Designing Robust Software Analysis and Artificial Intelligence Approaches For Cybersecurity

Giacomo Iadarola
Research fellow (Assegnista di Ricerca) at IIT-CNR
PhD student at Department of Computer Science (University of Pisa)

TUTOR: Fabio Martinelli (IIT-CNR)

Interests: Software Testing and Analysis - Mobile Security
Machine Learning - Cryptography (Blockchain)

ToDo: Adversarial Learning - Explicable AI
Outline

- Introduction

- Let’s talk about:
  - Software Testing and Analysis
  - Mobile Security

- Future Works
  - Adversarial Learning

- Conclusion
Software Testing and Analysis

Code for 5 minutes

WOW, THAT'S FUN

Debug for 5 hours
Introduction

All software have bugs, we know that...

Number of bugs per kLOC:
Between 57.02 bugs/kLOC
and 10.09 bugs/kLOC

Time to Fix:
Between 5 and 340 days

... and also the smallest vulnerability may trigger a domino effect!

Goal of GrapPa

*Design and implement a generic bug finder that uses machine learning to learn from buggy examples*

- Static analysis
  - from source code to graph
- Train graph-based classifier
- Classify graphs of previously unseen code
What is “buggy”?

1. `void` example(User user){
2.     String name = user.getName();
3.     if(user != null){
4.         int id = user.getId();
5.     }
6. }

    Buggy example

1. `void` example(User user){
2.     if(user != null){
3.         String name = user.getName();
4.         int id = user.getId();
5.     }
6. }    

    Non-Buggy example
What is “buggy”?

Buggy example

```java
void example(User user) {
    String name = user.getName();
    if (user != null) {
        int id = user.getId();
    }
}
```

Non-Buggy example

```java
void example(User user) {
    if (user != null) {
        String name = user.getName();
        int id = user.getId();
    }
}
```
Background

• **Code Property graph (CPG)**
  ➢ Merges classical graph representation into one data structure

• **Contextual Graph Markov Model (CGMM)**
  ➢ Neural network approach for processing graph data

• **Multilayer Perceptron (MLP)**
  ➢ Classical neural network model
Background - CPG

```c
1. void Example()
2. {
3.     int x = 0;
4.     if (x < 3){
5.         x = x + 1;
6.     }
7. }
```

Code example
An unsupervised model able to encode graphs of varying size and topology to a fixed dimension vector.

Background - CGMM

Feedforward artificial neural network.

**Dropout**

The dropout layer randomly selects a fraction rate of input neurons that are ignored during training.
Methodology

Approach steps

• Database of source code samples
• Static analysis and graph generation
• Graph vectorization
• Classification
Approach - The Dataset

Free Database with tons of examples

Wow, look! Nothing!
Approach - The Dataset

1. **.java** → **Major Mutation Framework** → **.java**
2. **.java** → **Run Test Suite on mutated code** → **.LOG**
3. **.LOG** → **Analyze log file, group mutants by kind of error raised up**
Approach - The Dataset

Arithmetic Operator Replacement
a + b ⟷ a - b
Logical Operator Replacement
a ∧ b ⟷ a | b
Conditional Operator Replacement
a || b ⟷ a && b
Relational Operator Replacement
a == b ⟷ a >= b
Shift Operator Replacement
a >> b ⟷ a << b
Expression Value Replacement
return a ⟷ return 0
int a = b ⟷ int a = 0
Literal Value Replacement
0 ⟷ 1
1 ⟷ -1
1 ⟷ 0
true ⟷ false
false ⟷ true
"Hello" ⟷ ""

Statement Deletion
Deletes (omits) a single statement:
Approach - Generate CPGs

```
ENTRY
<table>
<thead>
<tr>
<th>Int x = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>If (x &lt; 3)</td>
</tr>
<tr>
<td>x = x + 1</td>
</tr>
<tr>
<td>EXIT</td>
</tr>
</tbody>
</table>
```

- DATA: `Int x = 0`
- CONTROL: `If (x < 3) True False x = x + 1`

AST Node | CFG Node | CGF Edge | PDG Edges
---|---|---|---
Static analysis, generates simplified CPGs
Buggy method graph
Not-buggy method graph
Approach - Graphs vectorization

Dataset of a bug pattern

Train a CGMM model with the graph dataset

CGMM Trained

Dataset of unclassified graphs

Vectorize graphs using the trained model

VECTORS DATASET
Approach presented by Gal Y. and Ghahramani Z. to calculate the uncertainty of the model predictions.

Output for each sample:
- **Prediction value** in range [0, 1]
- **Uncertainty value** in range [0, 1.8]

---

Approach - Classification

We define uncertainty as:

\[
\text{uncertainty}(\text{vect}) = |\text{pred}(\text{vect}) - \text{avg\_pred\_dropout}(\text{vect})| + \text{std\_pred\_drop}(\text{vect})
\]

**Final step**: removing graphs/vector:

\[
\text{filter\_vectors}(\text{vect}) = \begin{cases} 
\text{remove} & \text{if } \text{uncertainty}(\text{vect}) > T \\
\text{store} & \text{otherwise}
\end{cases}
\]
Model trained on a specific bug pattern

Approach - Classification

- MLP Trained
- Analyzes and Classifies every source code file
- RESULTS
  - Predictions and subset of methods
Implementation - GrapPa

- **Mutate code**
  - Major: mutation framework

- **Generate CPG**
  - Soot: analyzing Java applications

- **Graph to Vector**
  - CGMM tool: Github by Errica F. (@diningphil)

- **Classification**
  - Weka
  - Keras
  - Tensorflow
Results - NPE Example #1

- Classified by the model as: **BUGGY**
- Manual check classified as: **BUGGY**

```java
public void draw(Graphics2D g2, XYPlot plot, Rectangle2D dataArea,
                 ValueAxis domainAxis, ValueAxis rangeAxis,
                 int rendererIndex,
                 PlotRenderingInfo info) {

    PlotOrientation orientation = plot.getOrientation();
    AxisLocation xAxisLocation = plot.getDomainAxisLocation();
    AxisLocation yAxisLocation = plot.getRangeAxisLocation();
    RectangleEdge xEdge = Plot.resolveDomainAxisLocation(xAxisLocation,
                                                          orientation);
    RectangleEdge yEdge = Plot.resolveRangeAxisLocation(yAxisLocation,
                                                          orientation);

    ...  
}
```
Results - NPE Example #2

- Classified by the model as: **BUGGY**
- Manual check classified as: **NON-BUGGY**

```java
1. public void add(Block block, Object key) {
2.     // since the flow layout is relatively straightforward,
3.     // no information needs to be recorded here
4. }
5.
```
Results - NPE Example #2

- Classified by the model as: **BUGGY**
- Manual check classified as: **NON-BUGGY**

```java
public void add(Block block, Object key) {
  // since the flow layout is relatively straightforward,
  // no information needs to be recorded here
}
```

People with no idea about AI saying it will take over the world:  

My Neural Network:
Take-home points for GrapPa

**Novel and general approach**
- Use of recent works
- Useful for developers in improving code security
- Not need prior-knowledge on code (neither on the bug pattern)

**The tool GrapPa** (https://github.com/Djack1010/GrapPa)
- Three trained models available
- Easy to include more bug patterns

Simplified version of the CPG

Three datasets of **syntetich bugs** available online
- https://github.com/Djack1010/BUG_DB
Mobile Security

When you restart the router and the internet magically starts working
Motivation

• Mobile devices handle huge amount of sensitive data
  ➢ really lucrative and attractive for attackers

• Mobile malware abuse of the “weakest link” of security
  ➢ malware detection techniques to mitigate

• Banking malware are critical
  ➢ significant exposure to every infected device
Formal methods in a nutshell

➢ Formal Model

Calculus of Communicating Systems of Milner (CCS)

\[
\text{doing\_shopping} = \text{init} \land \text{empty\_cart} \land \text{not\_empty\_cart}
\]

\[
\text{init} = \text{init.} <\text{start}>\text{empty\_cart}
\]

\[
\text{empty\_cart} = \text{empty\_cart.}<\text{add\_item}>\text{not\_empty\_cart}
\]

\[
\text{not\_empty\_cart} = \text{not\_empty\_cart.}<\text{add\_item}>\text{not\_empty\_cart} \lor \text{not\_empty\_cart.}<\text{pay}>\text{true}
\]
The Method

➢ Formal Model

- Java Bytecode-to-CCS transformation
  ➢ defined for each instruction

App under analysis

Transformed .class files

Transformation Function

CCS specification

Labelled Transition System

➢ Temporal Logics

- Specify set of properties
  ➢ describing malware behaviours

Manual inspection and current literature

Properties

\[
\begin{align*}
\phi_{TIGERBOT} &= \mu X. (\text{invokegetData}) \phi_1 \lor (\text{~invokegetData}) X \\
\phi_1 &= \mu X. (\text{pushpackName}) \phi_2 \lor (\text{~pushpackName}) X \\
\phi_2 &= \mu X. (\text{invokeString}) \phi_3 \lor (\text{~invokeString}) X \\
\phi_3 &= \mu X. (\text{invokeData}) \phi_4 \lor (\text{~invokeData}) X \\
\phi_4 &= \mu X. (\text{pushprodName}) \phi_5 \lor (\text{~pushprodName}) X \\
\phi_5 &= \mu X. (\text{invokeString}) \phi_6 \lor (\text{~invokeString}) X \\
\phi_6 &= \mu X. (\text{invokeData}) \phi_7 \lor (\text{~invokeData}) X \\
\phi_7 &= \mu X. (\text{pushcoortype}) \phi_8 \lor (\text{~pushcoortype}) X \\
\phi_8 &= \mu X. (\text{invokeString}) \phi_9 \lor (\text{~invokeString}) X \\
\phi_9 &= \ldots
\end{align*}
\]
The Method

Android APK ➔ Model

build

Model Checker

Malicious Behaviors ➔ Properties

specify

TRUE or FALSE
Features and Pros of the Method

- Use of formal methods
- Inspection directly on Java Bytecode
- Capture of malicious behaviours at finer granularity
- Method independent of source programming language
- Identification payload without decompilation
The Experiment on the Overlay family

1. Intercepting SMS messages

2. Stealing money in background

3. Password resetting


The Experiment on the Overlay family

```java
public void onReceive(Context var1, Intent var2){
    Bundle var5=var2.getExtras();
    int var3=var5.getInt('const_id_send_sms');
    int var4=var5.getInt('const_task_id_send_sms');
    M.d('receiverStatusSms', 'smsId': ' + var3 \\
    + "'; smsTaskId: " + var4 \\
    + "'; getResultCode(): " \\
    + this.getResultCode());
    switch(this.getResultCode()) {
    case -1:
        if (var4 >= 0) {
            SocketService.access$000(this.this$0, var4);
        }
        SocketService.access$100(this.this$0, var3, \\
        Const._SMS_STATUS_SEND);
    }
```

Malicious Behaviour in mu-calculus formulae

\[
\begin{align*}
\psi &= \mu X. \langle \text{pushconstidsendsms} \rangle \psi_1 \lor \langle \neg \text{pushconstidsendsms} \rangle X \\
\psi_1 &= \mu X. \langle \text{invokeGetInt} \rangle \psi_2 \lor \langle \neg \text{invokeGetInt} \rangle X \\
\psi_2 &= \mu X. \langle \text{store} \rangle \psi_3 \lor \langle \neg \text{store} \rangle X \\
\psi_3 &= \mu X. \langle \text{load} \rangle \psi_4 \lor \langle \neg \text{load} \rangle X \\
\psi_4 &= \mu X. \langle \text{pushconsttaskidsendsms} \rangle \psi_5 \lor \langle \neg \text{pushconsttaskidsendsms} \rangle X \\
\psi_5 &= \mu X. \langle \text{invokeGetInt} \rangle \psi_6 \lor \langle \neg \text{invokeGetInt} \rangle X \\
\psi_6 &= \mu X. \langle \text{store} \rangle \psi_7 \lor \langle \neg \text{store} \rangle X \\
\psi_7 &= \mu X. \langle \text{pushreceiverStatusSms} \rangle \psi_8 \lor \langle \neg \text{pushreceiverStatusSms} \rangle X \\
\psi_8 &= \mu X. \langle \text{pushsmsId} \rangle \psi_9 \lor \langle \neg \text{pushsmsId} \rangle X \\
\psi_9 &= \mu X. \langle \text{pushsmsTaskId} \rangle \psi_{10} \lor \langle \neg \text{pushsmsTaskId} \rangle X \\
\psi_{10} &= \mu X. \langle \text{pushgetResultCode} \rangle \ttt \lor \langle \neg \text{pushgetResultCode} \rangle X
\end{align*}
\]
The Experiment on the Overlay family

Malicious Behaviour in Java Code

```java
public void onReceive(Context var1, Intent var2) {
    Bundle var5 = var2.getExtras();
    int var3 = var5.getInt('const_id_send_sms');
    int var4 = var5.getInt('const_task_id_send_sms');
    M.d('receiverStatusSms', 'smsId': ' + var3 \ + "; smsTaskId": " + var4 \ + "; getResultCode(): " \ + this.getResultCode());
    switch (this.getResultCode()) {
        case -1:
            if (var4 >= 0) {
                SocketService.access$000(this.this$0, var4);
            }
    }
    SocketService.access$100(this.this$0, var3, Const._SMS_STATUS_SEND);
}
```

Malicious Behaviour in mu-calculus formulae

- \( \psi = \mu X.\langle pushconstidsendsms\rangle \psi_1 \lor \langle\neg pushconstidsendsms\rangle X \)
- \( \psi_1 = \mu X.\langle invokegetInt\rangle \psi_2 \lor \langle\neg invokegetInt\rangle X \)
- \( \psi_2 = \mu X.\langle store\rangle \psi_3 \lor \langle\neg store\rangle X \)
- \( \psi_3 = \mu X.\langle load\rangle \psi_4 \lor \langle\neg load\rangle X \)
- \( \psi_4 = \mu X.\langle pushconsttaskidsendsms\rangle \psi_5 \lor \langle\neg pushconsttaskidsendsms\rangle X \)
- \( \psi_5 = \mu X.\langle invokegetInt\rangle \psi_6 \lor \langle\neg invokegetInt\rangle X \)
- \( \psi_6 = \mu X.\langle store\rangle \psi_7 \lor \langle\neg store\rangle X \)
- \( \psi_7 = \mu X.\langle pushreceiverStatusSms\rangle \psi_8 \lor \langle\neg pushreceiverStatusSms\rangle X \)
- \( \psi_8 = \mu X.\langle pushsmsId\rangle \psi_9 \lor \langle\neg pushsmsId\rangle X \)
- \( \psi_9 = \mu X.\langle pushsmsTaskId\rangle \psi_{10} \lor \langle\neg pushsmsTaskId\rangle X \)
- \( \psi_{10} = \mu X.\langle pushgetResultCode\rangle \top \lor \langle\neg pushgetResultCode\rangle X \)
The Experiment on the Overlay family

public void onReceive(Context var1, Intent var2) {
    Bundle var5 = var2.getExtras();
    int var3 = var5.getInt('const_id_send_sms');
    int var4 = var5.getInt('const_task_id_send_sms');
    M.d('receiverStatusSms', 'smsId: ' + var3 + ';
        smsTaskId: ' + var4 + ';
        getResultCode(): ''
        this.getResultCode();
    switch(this.getResultCode()) {
    case -1:
        if (var4 >= 0) {
            SocketService.access$000(this.this$0, var4);
        }
    SocketService.access$100(this.this$0, var3, \n                                Const._SMS_STATUS_SEND);
}

Malicious Behaviour in mu-calculus formulae

ψ = μX.(pushconstidsendsms) ψ1 ∨ (¬pushconstidsendsms) X
ψ1 = μX.(invokeGetInt) ψ2 ∨ (¬invokeGetInt) X
ψ2 = μX.(store) ψ3 ∨ (¬store) X
ψ3 = μX.(load) ψ4 ∨ (¬load) X
ψ4 = μX.(pushconsttaskidsendsms) ψ5 ∨ (¬pushconsttaskidsendsms) X
ψ5 = μX.(invokeGetInt) ψ6 ∨ (¬invokeGetInt) X
ψ6 = μX.(store) ψ7 ∨ (¬store) X
ψ7 = μX.(pushreceiverStatusSms) ψ8 ∨ (¬pushreceiverStatusSms) X
ψ8 = μX.(pushsmsId) ψ9 ∨ (¬pushsmsId) X
ψ9 = μX.(pushsmsTaskId) ψ10 ∨ (¬pushsmsTaskId) X
ψ10 = μX.(pushgetResultCode) ψ11 ∨ (¬pushgetResultCode) X

Pesaresi Seminar – 16th Mar 2020
The Dataset

+ 75 malware Overlay family
+ 250 malware from Drebin [1]*
+ 50 trusted samples

= 375 real world samples

*25 randomly selected samples from each of the top 10 Drebin Malware Families

Evaluation Result

<table>
<thead>
<tr>
<th>#Malware ∈ Overlay</th>
<th>#Malware ∉ Overlay</th>
<th>#Trusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>250</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>True Positive</th>
<th>False Positive</th>
<th>False Negative</th>
<th>True Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>0</td>
<td>0</td>
<td>300</td>
</tr>
</tbody>
</table>
Take-home points

Short experimental paper: applied known technique[1,2] on a specific malware classification problem

- **Methodology:**
  ➢ model checking to detect Overlay malware

- **Database:**
  ➢ 350 real world applications

- **Experiment result:**
  ➢ achieved precision and recall values equal to 1

Limitations and Future Works

- Extend analysis to more malware (families)
  - Image classification and Deep Learning

- Take into account obfuscation
  - Check robustness model

- Using preliminary static analysis to automatize malicious behaviour extraction (GrapPa)
Research topics and publications

- **Software Testing and Analysis**
  - Graph-based classification for detecting instances of bug patterns → Master’s degree thesis TU Darmstadt

- **Mobile Security (Android OS)**
  - Improving robustness and efficiency in malware classification → Work in progress with F. Mercaldo
  - Formal Methods for Android Banking Malware Analysis and Detection → Published IOTSMS19

- **Machine Learning** (towards Adversarial Learning)
  - Image-based Malware Family Detection: An Assessment between Feature Extraction and Classification Techniques → submitted IoTBDS20
Thanks for the attention

Questions?
Literature for specifying malware behaviours as logic property:

Applied techniques based on Formal Methods:

Database:
Results GrapPa

JfreeChart project as test dataset (7555 methods)

Frequency of predictions without dropout (on the left) and the average of predictions with dropout (on the right).
Results GrapPa

- Selected 2675 methods out of 7555

Threshold = 0.05
Manually checked 80 methods of the 2675 selected by the tool
➢ 40 buggy predictions
➢ 40 non-buggy predictions

We agreed with the tool predictions in 70% of the cases.

<table>
<thead>
<tr>
<th>PREDICTION</th>
<th>AGREED with the model</th>
<th>NOT AGREED with the model</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Possible NPE</td>
<td>60%</td>
<td>40%</td>
</tr>
<tr>
<td></td>
<td>23 cases</td>
<td>17 cases</td>
</tr>
<tr>
<td>(0) NPE not-possible</td>
<td>80%</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>32 cases</td>
<td>8 cases</td>
</tr>
</tbody>
</table>

Result manual check
Intercepting SMS messages behaviour

```java
public void onReceive(Context var1, Intent var2) {
    String var3 = var2.getAction();
    this.app = \n        (Application)var1.getApplicationContext();
    if (var3.equals('android.intent.action.BOOT_COMPLETED')) {
        M.d('Receiver', 'ACTION_BOOT_COMPLETED');
        this.app.startSocket();
    } else if (var3.equals("alarm_check_connected")) {
        this.cardActivity(var1);
        this.app.startSocket();
    } else if (var3.equals('android.provider.Telephony.SMS_RECEIVED')) {
        M.d('Receiver', 'android.provider.Telephony.SMS_RECEIVED');
        new Thread(new 1(this, var2)).start();
    } try {
        this.abortBroadcast();
    } catch (Exception var4) {
        var4.printStackTrace();
    }
}
```
Password resetting behaviour

```java
void showReq() {
    this.deviceManger = (DevicePolicyManager)
    this.getSystemService('device_policy')
    this.deviceAdmin = new ComponentName(this,
            MyAdmin.class);
    Intent var1 = new Intent( \n        'android.app.action.ADD_DEVICE_ADMIN');
    var1.putExtra('android.app.extra.' + \n        'DEVICE_ADMIN', this.deviceAdmin);
    var1.putExtra('android.app.extra.' + \n        'ADD_EXPLANATION', 'Explanation Message';
    this.startActivityForResult(var1, 1);
    if ((this.deviceManger.isAdminActive( \n            this.compName))
        this.app.setAdmin(1);
}
```

\[
\begin{align*}
\zeta &= \mu X. (\text{pushdevicepolicy}) \; \zeta_1 \lor (\neg \text{pushdevicepolicy}) \; X \\
\zeta_1 &= \mu X. (\text{invokegetSystemService}) \; \zeta_2 \lor (\neg \text{invokegetSystemService}) \; X \\
\zeta_2 &= \mu X. (\text{checkcastandroidappadminDevicePolicyManager}) \; \zeta_3 \lor \neg \text{checkcastandroidappadminDevicePolicyManager} \; X \\
\zeta_3 &= \mu X. (\text{pushandroidappactionADDDEVICEADMIN}) \; \zeta_4 \lor (\neg \text{pushandroidappactionADDDEVICEADMIN}) \; X \\
\zeta_4 &= \mu X. (\text{pushandroidappextraDEVICEADMIN}) \; \zeta_5 \lor (\neg \text{pushandroidappextraDEVICEADMIN}) \; X \\
\zeta_5 &= \mu X. (\text{invokeputExtra}) \; tt \lor (\neg \text{invokeputExtra}) \; X
\end{align*}
\]
Why Formal Methods?

• The checking process is automatic, there is no need to construct a correctness proof

• The possibility of using the diagnostic counterexamples

• Temporal logic can easily and correctly express the behaviour of a malware

• Formal verification allows evaluating all possible scenarios, the entire state space all at once