

Dynamic Analysis and Classification of Android Malware

17th International School on Foundations of Security Analysis and Design (FOSAD)

University Residential Center of Bertinoro, Italy

Aug 28–Sep 2, 2017

Lorenzo Cavallaro

<lorenzo.cavallaro@rhul.ac.uk>

Research partially supported by the UK EPSRC grants EP/K033344/1 and EP/L022710/1





- ▶ Antifork Research
- ▶ s0ftpj



- ▶ Antifork Research
- ▶ s0ftpj



- ▶ BSc & MSc in Computer Science
- ▶ PhD in Computer Science (Computer Security)



- ▶ Antifork Research
- ▶ s0ftpj



- ▶ BSc & MSc in Computer Science
- ▶ PhD in Computer Science (Computer Security)



- ▶ 2006-2008: Visiting PhD Scholar – Prof. R. Sekar
- ▶ Systems security (mem err, taint tracking, anomaly detection)



- ▶ 2008-2010: PostDoc – Profs G. Vigna & C. Kruegel
- ▶ Malware analysis & detection (mostly botnet)



- ▶ 2010-2012: PostDoc – Prof. A. S. Tanenbaum
- ▶ OS Dependability (MINIX3) & Systems Security





► Reader (Associate Professor) of Information Security

► BSc & MSc in Computer Science

► Antifork Research





► Reader (Associate Professor) of Information Security

► BSc & MSc in Computer Science

► Antifork Research

Royal Holloway, University of London

► Founded in 1879 by Thomas Holloway

→ Entrepreneur and Philanthropist

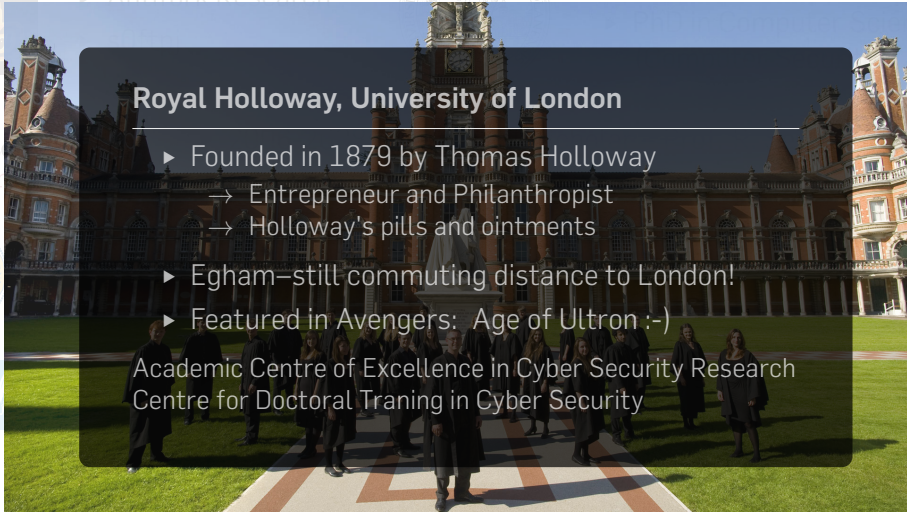
→ Holloway's pills and ointments

► Egham—still commuting distance to London!

► Featured in Avengers: Age of Ultron :-)

Academic Centre of Excellence in Cyber Security Research

Centre for Doctoral Training in Cyber Security



s2lab.isg.rhul.ac.uk

S²Lab

People

Projects

Publications

Reading Group

Teaching

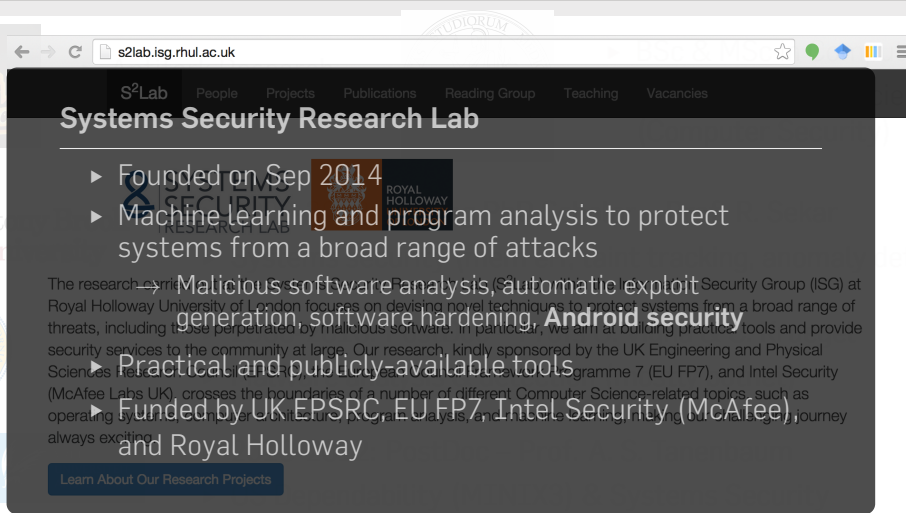
Vacancies



The research carried out at the Systems Security Research Lab (S²Lab) within the Information Security Group (ISG) at Royal Holloway University of London focuses on devising novel techniques to protect systems from a broad range of threats, including those perpetrated by malicious software. In particular, we aim at building practical tools and provide security services to the community at large. Our research, kindly sponsored by the UK Engineering and Physical Sciences Research Council (EPSRC), the European Council Framework Programme 7 (EU FP7), and Intel Security (McAfee Labs UK), crosses the boundaries of a number of different Computer Science-related topics, such as operating systems, computer architecture, program analysis, and machine learning, making our challenging journey always exciting.

[Learn About Our Research Projects](#)





The screenshot shows the homepage of the Systems Security Research Lab (S²Lab) at Royal Holloway University of London. The browser address bar shows 's2lab.isg.rhul.ac.uk'. The website has a dark header with navigation links: S²Lab, People, Projects, Publications, Reading Group, Teaching, and Vacancies. The main content area features the lab's name in large white text, followed by a list of bullet points and a paragraph of text. A 'Learn About Our Research Projects' button is at the bottom.

- ▶ Founded on Sep 2014
- ▶ Machine learning and program analysis to protect systems from a broad range of attacks

The research carried out in the Security Group (ISG) at Royal Holloway University of London focuses on devising novel techniques to protect systems from a broad range of threats, including those perpetrated by malicious software. In particular, we aim at building practical tools and provide security services to the community at large. Our research, kindly sponsored by the UK Engineering and Physical Sciences Research Council (EPSRC), the EU FP7, and Intel Security (McAfee Labs UK), crosses the boundaries of a number of different Computer Science-related topics, such as operating systems, computer architecture, program analysis, and machine learning, making our challenging journey always exciting.

and Royal Holloway

[Learn About Our Research Projects](#)

Google says there are now 1.4 billion active Android devices worldwide

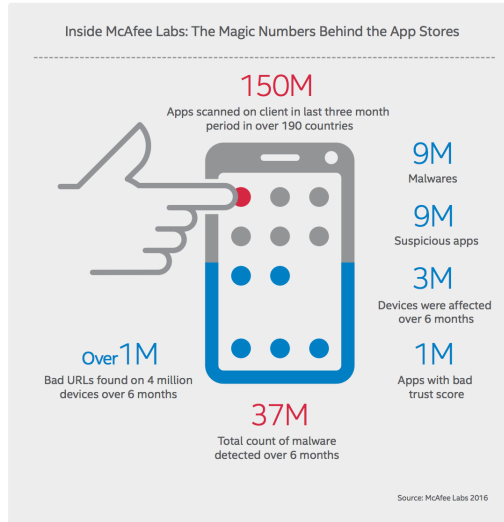
BY JOHN CALLAHAM 🕒 Tuesday, Sep 29, 2015 at 12:13 pm EDT



7 Comments



THE RISE IN ANDROID MALWARE



McAfee Labs routinely scans major app stores for infected systems as well as apps with suspicious behavior.

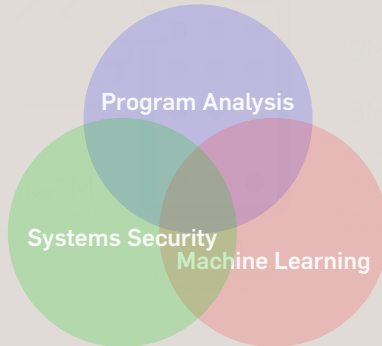
THE RISE IN ANDROID MALWARE

We need to automatically analyze programs

- ▶ Enable understanding of programs behaviors to aid analysts
- ▶ Facilitate additional automated analyses
(e.g., machine learning, security policy enforcement, hardening)

Inside McAfee Labs: The Magic Numbers Behind the App Stores

Apps scanned on client in last three month
period in over 190 countries



Program Analysis — the process of **automatically** analyzing the **behavior** of computer programs with respect to specific properties (e.g., correctness, robustness, safety, security, liveness)¹

¹Herbert Wiklicky—https://www.doc.ic.ac.uk/%7Eherbert/teaching/00_TasterPrint.pdf

Representation and Analysis of Software

- Control Flow Graphs

- Data Flow Information

- Data and Control Dependence Graphs

- Slicing

Dynamic Analysis for Android

Classification of Android Malware

Machine Learning and Malicious Software: Quo Vadis?

Disclaimer

Unless stated otherwise, all descriptions in this section (verbatim or rephrasing) and examples are taken from Harrold et al. Representation and Analysis of Software², a text introducing and summarizing core program analysis techniques. Any erroneous simplification or misinterpretation is my own mistake³.

²<http://www.ics.uci.edu/%7Elopes/teaching/inf212W12/readings/rep-analysis-soft.pdf>

³This is not strictly speaking a lecture on program analysis—that said, I hope there is none :-)

A control flow graph (CFG)⁴ is a directed graph in which each node represents a basic block and each edge represents the flow of control between basic blocks

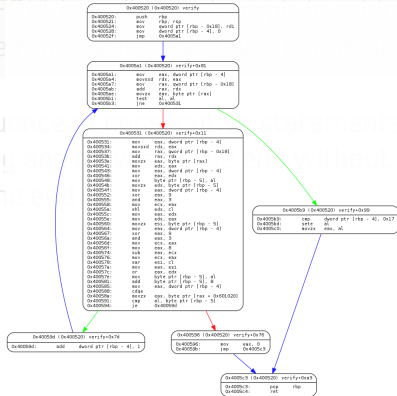
- ▶ A basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end

⁴A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, Reading, MA, 1986.

CONTROL FLOW GRAPHS

A control flow graph (CFG) is a directed graph where each node represents a basic block and each edge represents a control flow edge.

- A basic block is a sequence of instructions in which flow of control enters at the beginning and exits at the end without possibility of branching except at the end.



⁴A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, Reading, MA, 1986.

CONTROL FLOW GRAPHS

A control flow graph (CFG)⁴ is a directed graph in which each node represents a basic block and each edge represents the flow of control between basic blocks

- ▶ When analyzing source code, it is usually more convenient to consider each source code statement as a basic block

⁴A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, Reading, MA, 1986.



CONTROL FLOW GRAPHS

A control flow graph (CFG) is a directed graph in which each node represents a basic block and each edge represents the flow of control between basic blocks

- ▶ A basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end
- ▶ When analyzing source code, it is usually convenient to consider each source code statement as a basic block

Program Sums

```
1.  read(n);
2.  i = 1;
3.  sum = 0;
4.  while (i <= n) do
5.      sum = 0;
6.      j = 1;
7.      while (j <= i) do
8.          sum = sum + j;
9.          j = j + 1;
10.     endwhile;
11.     write(sum, i);
12.     i = i + 1;
13. endwhile;
14. write(sum, i);
15. end Sums
```

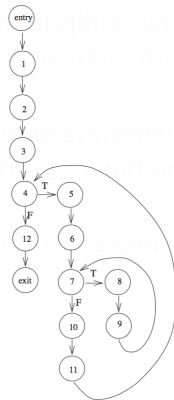


Figure 3: Program segment on the left, with its CFG on the right.

⁴A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, Reading, MA, 1986.

DATA FLOW INFORMATION: DEF-USE CHAINS

- ▶ We can classify each reference to a variable in a program as a definition or a use
 - A definition of a variable occurs whenever a variable gets a value
 - A use of a variable occurs whenever the value of the variable is fetched
 - A computation use (c-use) occurs whenever a variable affects a given computation or is output
 - A predicate use (p-use) occurs whenever a variable affects the control flow through the program
- ▶ When information is computed across across basic blocks (statements), we refer to it as intraprocedural data flow analysis

DATA FLOW INFORMATION: DEF-USE CHAINS

- ▶ We can classify each reference to a variable in a program as a definition or a use
 - A definition of a variable occurs whenever a variable gets a value
 - A use of a variable occurs whenever a value of the variable is fetched
 - A computation use (c-use) occurs whenever a variable affects a given computation or is output
 - A predicate use (p-use) occurs whenever a variable affects the control flow through the program
- ▶ When information is collected across basic blocks (statements), we refer to it as intrablock data flow analysis

Program Sums

```
1.  read(n);
2.  i = 1;
3.  sum = 0;
4.  while (i <= n) do
5.      sum = 0;
6.      j = 1;
7.      while (j <= i) do
8.          sum = sum + j;
9.          j = j + 1;
10.         endwhile;
11.     write(sum, i);
12.     i = i + 1;
13. endwhile;
14. write(sum, i);
15. end Sums
```

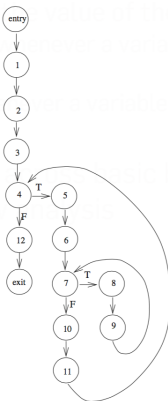


Figure 3: Program segment on the left, with its CFG on the right.



- ▶ We can classify each reference to a variable in a program as a definition or a use
 - A definition of a variable occurs whenever a variable gets a value
 - A use of a variable occurs whenever the value of the variable is fetched

Reaching Definitions

Given a program P with CFG G , a definition d reaches a point at p in G if there is a path in G from the point immediately following d to p such that d is not killed (e.g., redefined) along that path

A way to compute reaching definitions for all the variables in a program is to consider each definition d individually and propagate it along all paths from the definition until either d is killed or an exit from the program is reached^a

^aA more efficient approach is iterative dataflow analysis—see <http://www.ics.uci.edu/%7EElopes/teaching/inf212W12/readings/rep-analysis-soft.pdf>

DATA FLOW INFORMATION: DEF-USE CHAINS

- ▶ We can classify each reference to a variable in a program as a definition or a use
 - A definition of a variable occurs whenever a variable gets a value
 - A use of a variable occurs whenever a value of the variable is fetched
 - A computation use (c-use) occurs whenever a variable affects a given computation or is output
 - A predicate use (p-use) occurs whenever a variable affects the control flow through the program
- ▶ When information is passed across basic blocks (statements), we refer to it as intra-procedural data flow

Program Sums

```
1.  read(n);
2.  i = 1;
3.  sum = 0;
4.  while (i <= n) do
5.      sum = 0;
6.      j = 1;
7.      while (j <= i) do
8.          sum = sum + j;
9.          j = j + 1;
10.         endwhile;
11.     write(sum, i);
12.     i = i + 1;
13. endwhile;
14. write(sum, i);
15. end Sums
```

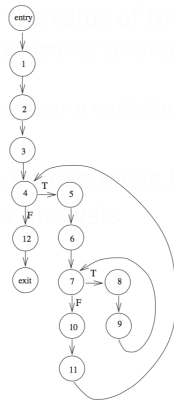


Figure 3: Program segment on the left, with its CFG on the right.

- ▶ A definition-use pair for variable v is an ordered pair (D,U)
 - D is a statement that contains a definition of v
 - U is a statement that contains a use of v
 - There is a subpath from D to U where v is not killed

- ▶ A definition-use pair for variable v is an ordered pair (D,U)
 - D is a statement that contains a definition of v
 - U is a statement that contains a use of v
 - There is a subpath from D to U where v is not killed
- ▶ Giving reaching definitions information, we can compute definition-use pairs
- ▶ Let (D,U) be a definition-use pair, then
 - The computation at U is dependent upon data computed at D (**Data Dependence**)
 - We can represent data dependences with a **Data Dependence Graph (DDG)** or we can add data dependence edge to the CFG

- ▶ A definition-use pair for variable v is an ordered pair (D,U)

→ D is a statement that contains a definition of v

→ U is a statement that contains a use of v

→ There is a control flow path from D to U where v is not killed

- ▶ Giving reachability information, we can compute definition-use pairs

- ▶ Let (D,U) be a definition-use pair

→ The control flow path from D to U is dependent upon data computed at D (Data Dependence)

→ We can construct a Data Dependence Graph (DDG) or we can add edges to a control flow graph

| Node | Definition-Use Pairs |
|-------|---|
| entry | (none) |
| 1 | (none) |
| 2 | (none) |
| 3 | (none) |
| 4 | [1:n,4:n],[2:i,4:i],[11:i,4:i] |
| 5 | (none) |
| 6 | (none) |
| 7 | [6:j,7:j],[9:j,7:j],[2:i,7:i],[11:i,7:i] |
| 8 | [5:sum,8:sum],[8:sum,8:sum],[6:j,8:j],[9:j,8:j] |
| 9 | [6:j,9:j],[9:j,9:j] |
| 10 | [2:i,10:i],[11:i,10:i],[5:sum,10:sum],[8:sum,10:sum] |
| 11 | [2:i,11:i] |
| 12 | [2:i,12:i],[11:i,12:i],[3:sum,12:sum],[5:sum,12:sum],[8:sum,12:sum] |
| exit | (none) |

Table 1: Definition-use pairs for program Sums.

DATA DEPENDENCE GRAPHS

- ▶ A definition-use pair for variable v is an ordered pair (D,U)

- D is a statement that contains a definition of v
- U is a statement that contains a use of v
- There is a statement S from D to U where v is not killed

- ▶ Giving reaching definitions information to a compiler to compute definition-use pairs

- ▶ Let (D,U) be a definition-use pair, then

- The computation of U depends on the value computed at D (Data Dependence)
- We can represent this as a Data Dependence Graph (DDG) or we can add data dependence edges to a control flow graph

Program Sums

```
1.  read(n);
2.  i = 1;
3.  sum = 0;
4.  while (i <= n) do
5.      sum = 0;
6.      j = 1;
7.      while (j <= i) do
8.          sum = sum + j;
9.          j = j + 1;
10.     endwhile;
11.     write(sum, i);
12.     i = i + 1;
13. endwhile;
14. write(sum, i);
15. end Sums
```

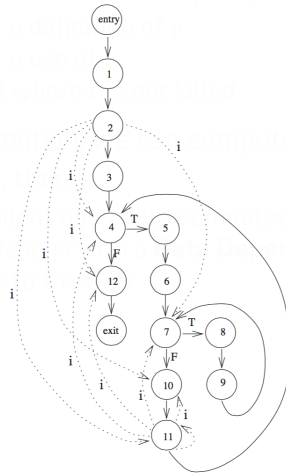


Figure 8: Control flow graph for Sums, with flow dependence edges for i added (dotted lines).

- ▶ A **Control Dependence Graph (CDG)** encodes control dependencies
- ▶ Let X and Y be nodes in CFG G ; then Y is control dependent on X iff
 1. There exists a directed path P from X to Y with any Z in P (excluding X and Y) postdominated by Y , and
 2. X is not postdominated by Y(If Y is control dependent on X then X must have two exits where one of the exits always reaches Y and the other not)

CONTROL DEPENDENCE GRAPHS

- ▶ A **Control Dependence Graph (CDG)** encodes control dependencies

- ▶ Let X and Y be nodes in CFG G ; then Y is control dependent on X if

1. There exists a directed path P from X to Y with any Z in P (excluding X and Y)

postdominated by X and

2. X is not postdominated by Y

(If Y is control dependent on X then X must have two exits where one of the

exists always reaches Y and the other does not)

Program Sums

```
1. read(n);
2. i = 1;
3. sum = 0;
4. while (i <= n) do
5.   sum = 0;
6.   j = 1;
7.   while (j <= i) do
8.     sum = sum + j;
9.     j = j + 1;
10.  endwhile;
11.  write(sum, i);
12.  i = i + 1;
13. endwhile;
14. write(sum, i);
15. end Sums
```

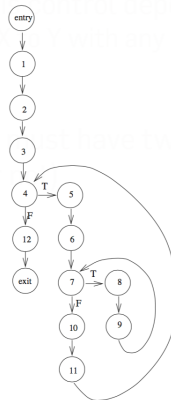


Figure 3: Program segment on the left, with its CFG on the right.

CONTROL DEPENDENCE GRAPHS

- ▶ A Control Dependence Graph (CDG) encodes control dependencies

- ▶ Let X and Y be nodes in CDG G , then Y is control dependent on X iff

1. There exists a path P from X to Y in G such that every node Z in P (excluding X and Y) is a predicate node.
2. X is not postdominated by Y .

(If Y is control dependent on X , then X must be a predicate node, and Y must be a statement node where one of the branches of X leads always reaches Y and the other not.)

Program Sums

```
1.  read(n);
2.  i = 1;
3.  sum = 0;
4.  while (i <= n) do
5.      sum = 0;
6.      j = 1;
7.      while (j <= i) do
8.          sum = sum + j;
9.          j = j + 1;
10.         endwhile;
11.         write(sum, i);
12.         i = i + 1;
13.     endwhile;
14.     write(sum, i);
15. end Sums
```

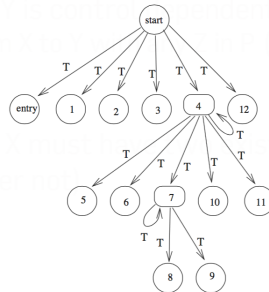


Figure 13: Program Sums on the left and its CDG without region nodes on the right. Circles represent statements in the program, and rounded rectangles represent predicates.

CONTROL DEPENDENCE GRAPHS

- ▶ A **Control Dependence Graph (CDG)** encodes control dependencies

- ▶ Let X and Y be nodes in a CDG G , then Y is control dependent on X iff

1. There exists a path P from X to Y in G such that X and Y are not in P (excluding X and Y)
2. X is not postdominated by Y

(If Y is control dependent on X , then X must always execute where one of the

exists always reaches the other node)

Program Sums

```
1.  read(n);
2.  i = 1;
3.  sum = 0;
4.  while (i <= n) do
5.      sum = 0;
6.      j = 1;
7.      while (j <= i) do
8.          sum = sum + j;
9.          j = j + 1;
10.         endwhile;
11.         write(sum, i);
12.         i = i + 1;
13.     endwhile;
14.     write(sum, i);
15. end Sums
```

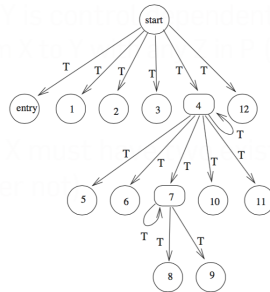


Figure 13: Program Sums on the left and its CDG without region nodes on the right. Circles represent statements in the program, and rounded rectangles represent predicates.

Program Dependence Graph

A program dependence graph embeds both control and data dependences

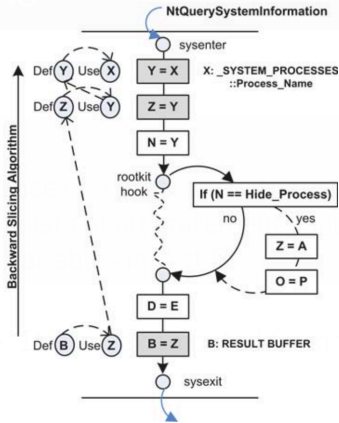
A slice of a program with respect to program point P and set of program variables V consists of all statements and predicates in the program that may affect the values of variables in V at P

- ▶ It can be computed from the program CFG and data flow analysis or its PDG



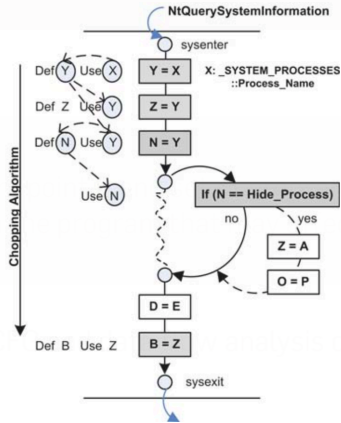
A slice of a program with respect to program point P and set of program variables V consists of all statements and predicates in the program that may affect the values of variables in V at P

- It can be computed from the program CFG and data flow analysis or its PDG



Backward slicing phase.

(a)



Chopping phase.

(b)

Figure 1. High-level overview of K-Tracer's approach.

a

^a<http://www.internet-society.org/doc/k-tracer-system-extracting-kernel-malware-behavior>

(A few) other Applications in Security

- ▶ Practical control flow integrity influenced by alias analysis precision (e.g., indirect calls)^a
- ▶ Programs graph isomorphism for malware detection^b
- ▶ Tainted graphs to detect and group similar malware in families^c
- ▶ ...
- ▶ And of course what we will see in the rest of the lecture!

^a<http://seclab.cs.sunysb.edu/seclab/pubs/oak13.pdf> and <https://www.usenix.org/node/190961>

^bhttps://www.cs.ucsb.edu/%7Exyan/papers/oakland20_malware.pdf and <http://sycurelab.ecs.syr.edu/%7Emu/Zhang-DroidSIFT-CCS14.pdf>

^chttps://www.usenix.org/legacy/event/sec09/tech/full_papers/sec09_malware.pdf and https://www.cs.ucsb.edu/~chris/research/doc/ndss09_cluster.pdf

| Year | Method | Venue | Type | | Feature | # Malware | DR/FP (%) | ACC (%) |
|------|---------------|-------------|------|-------|-------------------------------|-----------|-----------|---------|
| | | | Det | Class | | | | |
| 2014 | DroidAPIMiner | SecureComm | ✓ | — | API,PKG,PAR | 3,987 | 99/2.2 | — |
| 2014 | DroidMiner | ESORICS | ✓ | ✓ | CG,API | 2,466 | 95.3/0.4 | 92 |
| 2014 | Drebin | NDSS | ✓ | — | PER,STR,API,INT | 5,560 | 94.0/1.0 | — |
| 2014 | DroidSIFT | ACM CCS | ✓ | ✓ | API-F | 2,200 | 98.0/5.15 | 93 |
| 2014 | DroidLegacy | ACM PPREW | ✓ | ✓ | API | 1,052 | 93.0/3.0 | 98 |
| 2015 | AppAudit | IEEE S&P | ✓ | — | API-F | 1,005 | 99.3/0.61 | — |
| 2015 | MudFlow | ICSE | ✓ | — | API-F | 10,552 | 90.1/18.7 | — |
| 2015 | Marvin | ACM COMPSAC | ✓ | — | PER, INT, ST, PN | 15,741 | 98.24/0.0 | — |
| 2015 | RevealDroid | TR GMU | ✓ | ✓ | PER,API,API-F,INT,PKG | 9,054 | 98.2/18.7 | 93 |
| 2017 | MaMaDroid | NDSS | ✓ | — | Abstract APIs Markov Chain | 80,000 | 99/1 | — |
| 2017 | DroidSieve | ACM CODASPY | ✓ | ✓ | Syntactic- & Resource-centric | 100,000 | 99.7/0 | — |
| 2016 | Madam | IEEE TDSC | ✓ | — | SYSC, API, PER, SMS, USR | 2,800 | 96/0.2 | — |

| Year | Method | Venue | Type | | Feature | # Malware | DR/FP (%) | ACC (%) |
|------|---|--------------|------|-------|----------------------------|-----------|-----------|---------|
| | | | Det | Class | | | | |
| 2014 | DroidAPIMiner | SecureComm | ✓ | — | API,PKG,PAR | 3,987 | 99/2.2 | — |
| 2014 | DroidMiner | ESURICS | ✓ | ✓ | CG,API | 2,466 | 95.3/0.4 | 92 |
| 2014 | ▶ Most focus on statically-extracted features | ACM CCS | ✓ | ✓ | API,PKG,PAR | 5,560 | 94.0/1.0 | — |
| 2014 | DroidSIFT | ACM CCS | ✓ | ✓ | API-F | 2,200 | 98.0/5.15 | 93 |
| 2014 | → Issues with obfuscation, dynamically & native code | ACM CCS | ✓ | ✓ | API-F | 2,052 | 93.0/3.0 | 98 |
| 2015 | AppAudit | IEEE S&P | ✓ | — | API-F | 1,005 | 99.3/0.61 | — |
| 2015 | ▶ How far would dynamically-extracted features go? | ACM CCS | ✓ | ✓ | API-F | 10,552 | 90.1/18.7 | — |
| 2015 | Marvin | ACM COMP-SAC | ✓ | — | PER, INT, ST, PN | 15,741 | 98.24/0.0 | — |
| 2015 | RQ1 Automatic reconstruction of apps behaviors | ACM CCS | ✓ | — | API,PKG,PAR | 9,054 | 98.2/18.7 | 93 |
| 2017 | MaMaDroid | NDSS | ✓ | — | Abstract APIs Markov Chain | 80,000 | 99/1 | — |
| 2017 | (As fewer features as possible with rich semantics) | ACM CCS | ✓ | — | API,PKG,PAR | 10,000 | 99.7/0 | — |
| 2016 | RQ2 Machine learning with high-accuracy results | ACM CCS | ✓ | — | API,PKG,PAR | 2,800 | 96/0.2 | — |
| | (Challenging contexts: classification, sparse behaviors) | | | | | | | |
| | RQ3 Decaying machine learning models: concept drift | | | | | | | |
| | (Evaluate the quality of a classifier to identify drifting objects) | | | | | | | |

Representation and Analysis of Software

Dynamic Analysis for Android

- Semantic-Reconstruction Issues

- Understanding Android IPC

- Automatic Reconstruction of Binder Arguments

- (Open) Challenges and Living with them

Classification of Android Malware

Machine Learning and Malicious Software: Quo Vadis?



SYSTEMS
SECURITY
RESEARCH LAB



ROYAL
HOLLOWAY
UNIVERSITY
OF LONDON

WHAT IS DYNAMIC ANALYSIS?

Dynamic analysis

A technique for observing (and understanding) the runtime actions of an application

- ▶ Well-established technique to characterize process behaviors⁵
- ▶ Significantly better than static analysis when it comes to resilience against common obfuscation schemes⁶

⁵http://wenke.gtisc.gatech.edu/ids-readings/unix_process_self.pdf

⁶Thanks to Matthias Neugschwandtner for the bytecode examples in the next slides

ADVANTAGES: DETAIL OF INSIGHTS

```
.line 177
  .restart local v3      #file:Ljava/io/File;
  .restart local v4      #fos:Ljava/io/FileOutputStream;
  .restart local v5      #i:I
  .restart local v6      #is:Ljava/io/InputStream;
  .restart local v7      #temp:[B
  :cond_1
  const/4 v8, 0x0

  invoke-virtual {v4, v7, v8, v5}, Ljava/io/FileOutputStream;->write([BII)V
```

ADVANTAGES: RESILIENT AGAINST OBFUSCATION/ENCRYPTION

```
fill-array-data v0, :array_a
:array_a
.array-data 0x1,0x4t,0x4t,0x6t,0x8t,0xfat,0xe1t,0x2ct...,
const-class v0, Lo/abc;
invoke-virtual {v0}, Ljava/lang/Class;
    ->getClassLoader()Ljava/lang/ClassLoader;
move-result-object v0
```

ADVANTAGES: RESILIENT AGAINST REFLECTION

```
invoke-static {v0, v1, v2}, Lo/$CON;->(III)Ljava/lang/String;
move-result-object v0
invoke-static {v0}, Ljava/lang/Class;
    ->forName(Ljava/lang/String;)Ljava/lang/Class;
move-result-object v0
const/16 v1, 0x98
const/16 v2, -0x12b
const/16 v3, -0x21
invoke-static {v1, v2, v3}, Lo/$CONa;->(III)Ljava/lang/String;
move-result-object v1
const/4 v2, 0x0
invoke-virtual {v0, v1, v2}, Ljava/lang/Class;->getMethod(Ljava/lang/String;
    [Ljava/lang/Class;)Ljava/lang/reflect/Method;
move-result-object v0
const/4 v1, 0x0
invoke-virtual {v0, v12, v1}, Ljava/lang/reflect/Method;
    ->invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
```

RQ1—CopperDroid

Automatic Reconstruction of Apps Behaviors

- ▶ DroidScope/DECAF⁷
 - Dalvik VM method, asm insn, and system call tracing
 - 2-level VMI to get to Dalvik VM semantics
- ▶ Droidbox⁸ and TaintDroid⁹
- ▶ Other approaches generally built on top of Droidbox/DroidScope/TaintDroid

⁷<https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/yan> and <https://github.com/sycurelab/DECAF/tree/master/DroidScope/qemu>

⁸<https://github.com/pjlantz/droidbox>

⁹<http://www.appanalysis.org/>

- ▶ DroidScope/DECAF⁷
 - Dalvik VM method, asm insn, and system call tracing
 - 2-level VMI to get to Dalvik VM semantics
- ▶ Droidbox⁸ and TaintDroid⁹
- ▶ Other approaches generally built on top of Droidbox/DroidScope/TaintDroid

RQ

Is it necessary to look at Dalvik VM semantics or can we still reconstruct interesting behaviors from other observation points (perhaps a unique one)?

⁷<https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/yan> and <https://github.com/sycurelab/DECAF/tree/master/DroidScope/qemu>

⁸<https://github.com/pjlantz/droidbox>

⁹<http://www.appanalysis.org/>

- ▶ Established technique to characterize process behaviors¹⁰
- ▶ Identifying state-modifying actions crucial to analysis

¹⁰<https://www.cs.unm.edu/%2E/forrest/publications/acsac08.pdf>

- ▶ Established technique to characterize process behaviors¹⁰
- ▶ Identifying state-modifying actions crucial to analysis

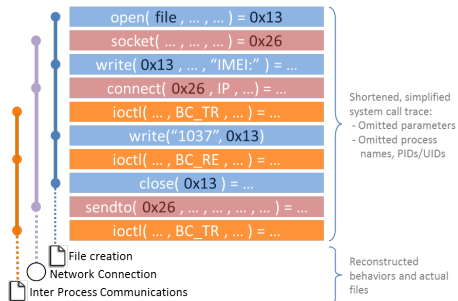
Can it be applied to Android?

- ▶ Android architecture is different to traditional devices
- ▶ State-modifying actions manifest at multiple abstractions
 - Traditional OS interactions (e.g., filesystem/network interactions)
 - Android-specific behaviors (e.g., SMS, phone calls)

¹⁰<https://www.cs.unm.edu/%2E/forrest/publications/acsac08.pdf>

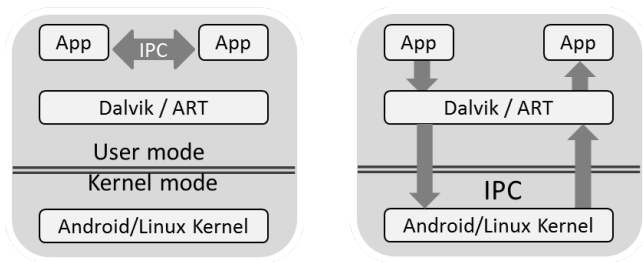
TRADITIONAL STATE-MODIFYING ACTIONS

► Traditional OS interactions



ANDROID'S ACTIONS

- ▶ Traditional OS interactions
- ▶ Android functionality (Send SMS, Phone Call etc.)
 - Largely achieved through IPC/ICC (ioctl)
 - The Binder protocol is crucial to this



▶ Traditional OS interactions

Key Insight

System calls provide the right semantic abstraction given the reconstruction of Inter-Component Communications (ICC) behaviors

- ▶ ICC (aka Binder transactions) are carried out as **ioctl** system calls
 - CopperDroid automatically unmarshalls such calls and reconstruct Android app behaviors^a
 - No modification to the OS
 - It works automatically across the Android fragmented ecosystem

^aKimberly Tam, Salahuddin J. Khan, Aristide Fattori, and Lorenzo Cavallaro. **CopperDroid: Automatic Reconstruction of Android Malware Behaviors**. In 22nd Annual Network and Distributed System Security Symposium (NDSS), 2015

IPC/RPC

- ▶ Binder protocols enable fast inter-process communication
- ▶ Allows apps to invoke other app component functions
- ▶ Binder objects handled by Binder Driver in kernel
 - Serialized/marshalled passing through kernel
 - Results in input output control (`ioctl`) system calls

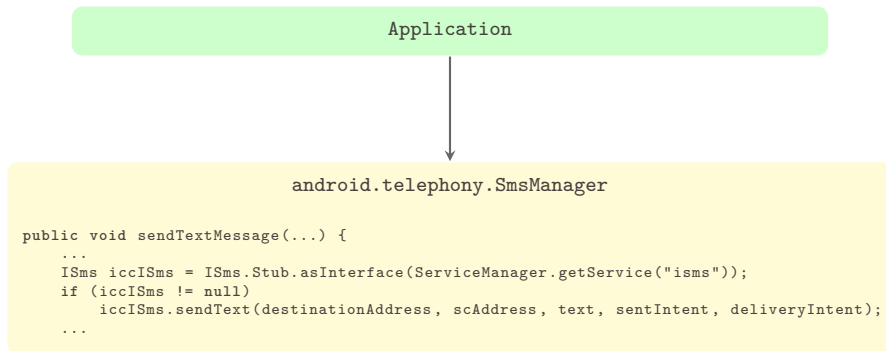
Android Interface Definition Language (AIDL)

- ▶ AIDL defines which/how services can be invoked remotely
- ▶ Describes how to marshal method parameters

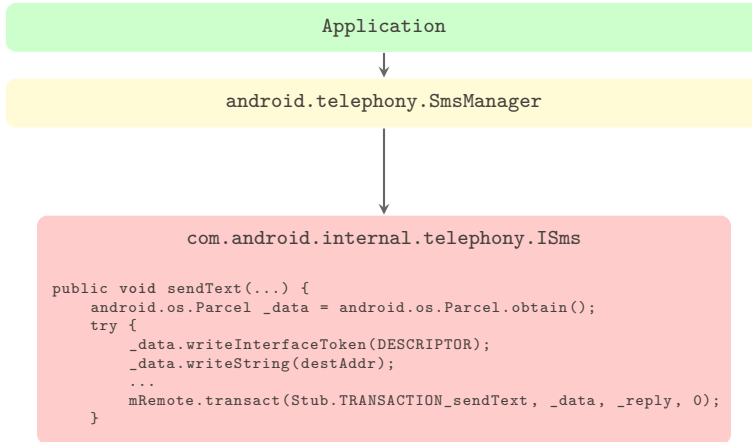
Application

```
PendingIntent sentIntent = PendingIntent.getBroadcast(SMS.this,  
0, new Intent("SENT"), 0);  
SmsManager sms = SmsManager.getDefault();  
sms.sendTextMessage("7855551234", null, "Hi_There", sentIntent, null);
```

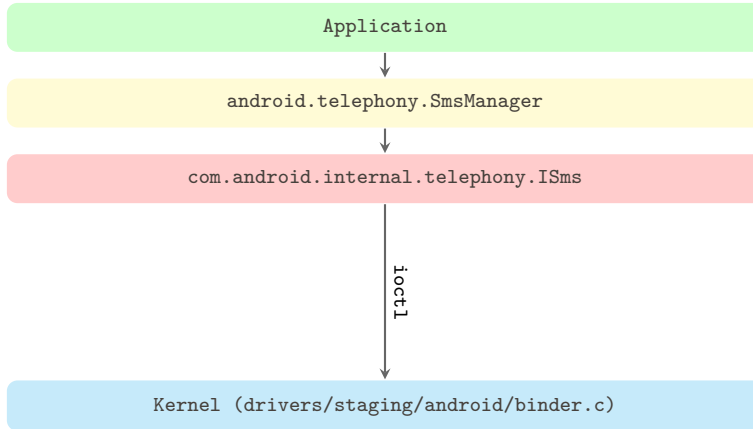
IPC BINDER: AN EXAMPLE



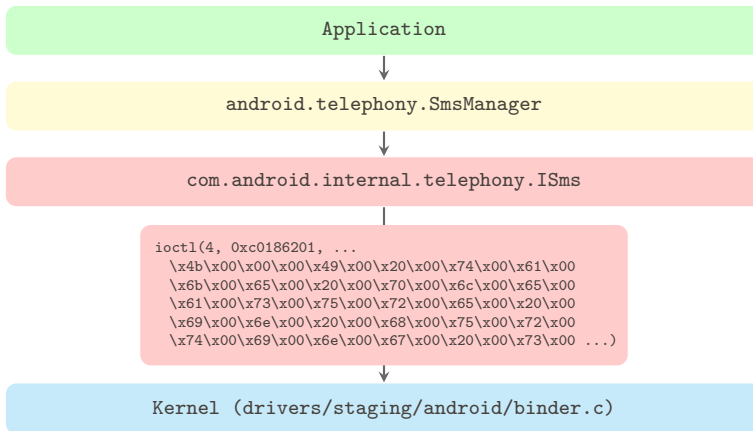
IPC BINDER: AN EXAMPLE



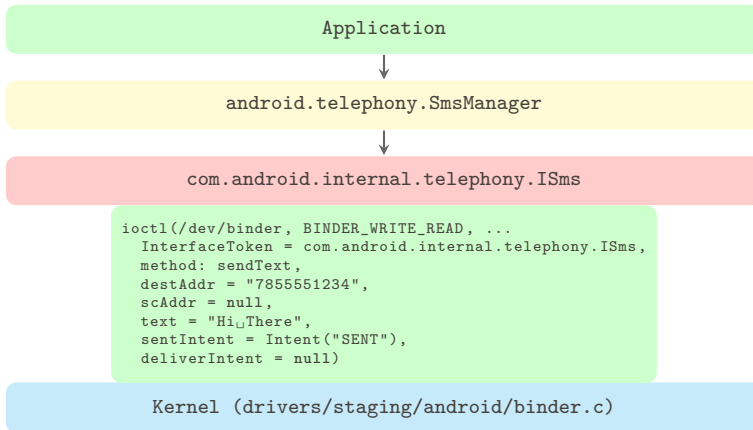
IPC BINDER: AN EXAMPLE



IPC BINDER: AN EXAMPLE



IPC BINDER: AN EXAMPLE



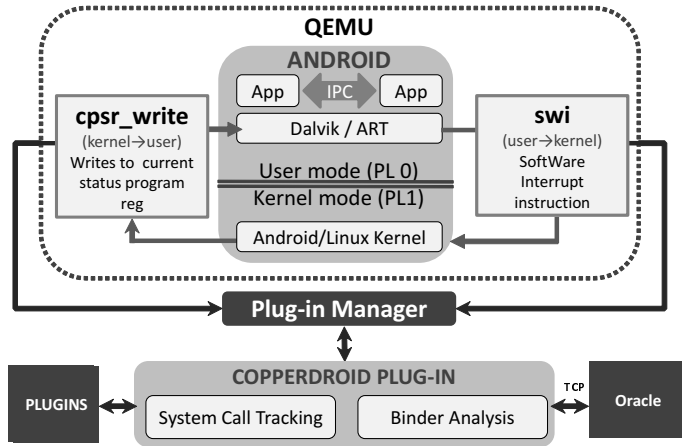
A system call induces a User -> Kernel transition

- ▶ On ARM invoked through the `swi` instruction (SoftWare Interrupt)
- ▶ `r7`: invoked system call number
- ▶ `r0-r5`: parameters
- ▶ `1r`: return address

CopperDroid's Approach

- ▶ instruments QEMU's emulation of the `swi` instruction
- ▶ instruments QEMU to intercept every `cpsr_write` (Kernel → User)
- ▶ Perform traditional VMI to associate system calls to threads

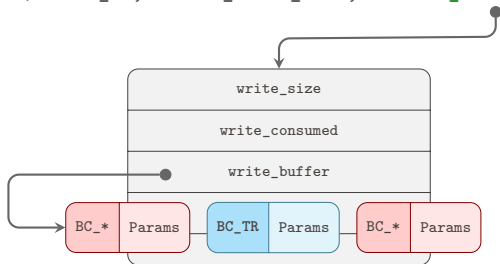
TRACING SYSTEM CALLS ON ANDROID ARM THROUGH QEMU



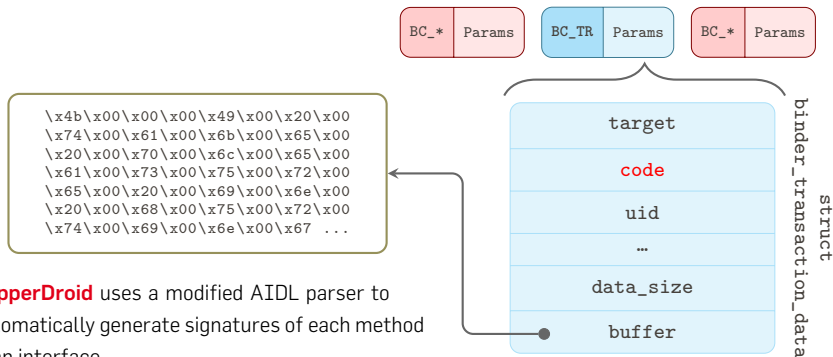
BINDER STRUCTURE WITHIN IOCTL

CopperDroid inspects the Binder protocol in detail by intercepting a subset of the `ioctl`s issued by userspace Apps.

```
ioctl(binder_fd, BINDER_WRITE_READ, &binder_write_read);
```

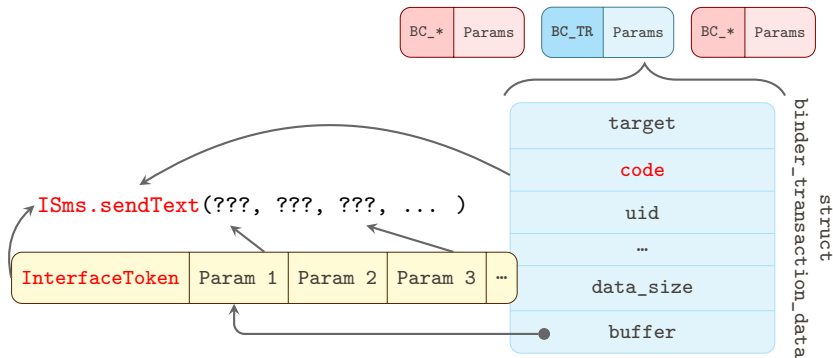


CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs

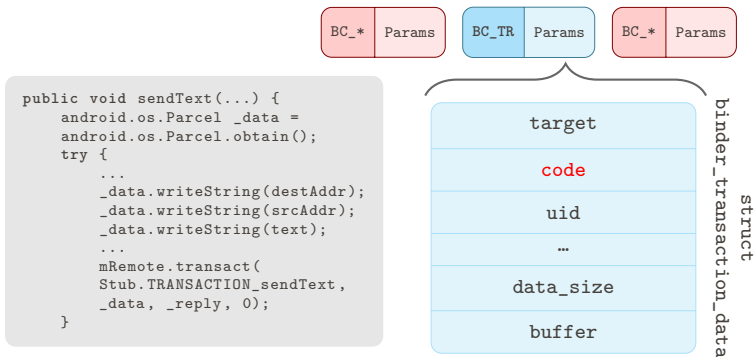


CopperDroid uses a modified AIDL parser to automatically generate signatures of each method in an interface.

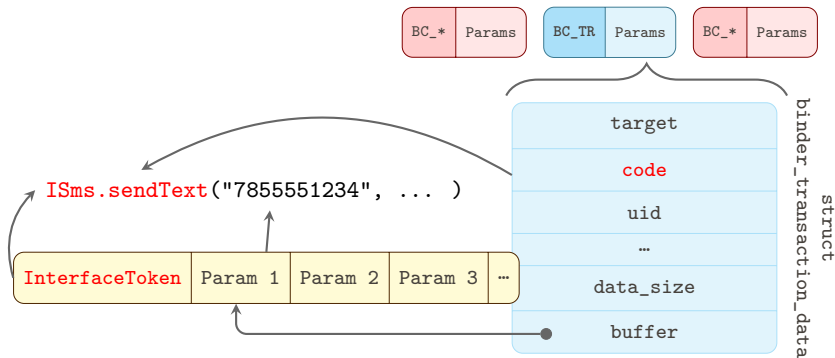
CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs



CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs



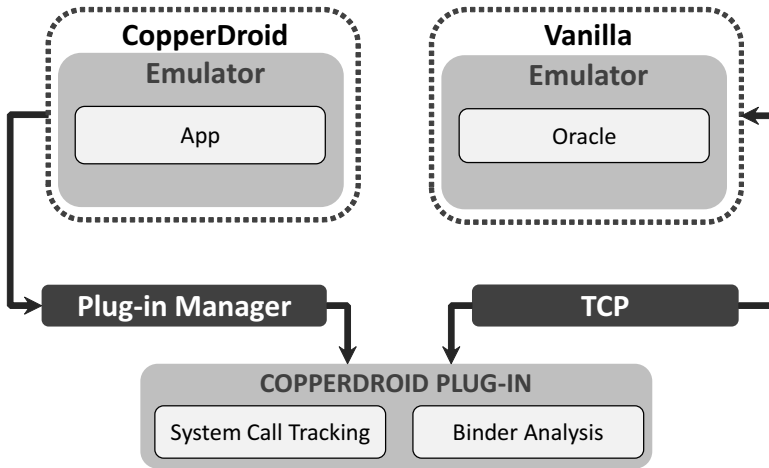
CopperDroid analyzes BC_TRANSACTIONs and BC_REPLYs



AUTOMATIC ANDROID OBJECTS UNMARSHALLING

- ▶ Primitive types (e.g., String text)
 - A few manually-written procedures
- ▶ Complex Android objects
 - 300+ Android objects–manual unmarshalling: does not scale & no scientific
 - Finds object CREATOR field
 - Use reflection (type introspection, then intercession)
- ▶ IBinder object reference
 - A handle (pointer) sent instead of marshalled object
 - Look earlier in trace to map each handle to an object

CopperDroid's Oracle unmarshalls all three automatically



AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

| | |
|--------|--|
| TYPE | "string", "string", "string", "PendingIntent", "PendingIntent" |
| DATA | <pre>\x0A \x00 \x00 \x00 \x37 \x00 \x38 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x31 \x00 \x32 \x00 \x33 \x00 \x34 \x00 \x00 \x00 \x00 \x08 \x00 \x00 \x00 \x48 \x00 \x69 \x00 \x20 \x00 \x74 \x00 \x68 \x00 \x65 \x00 \x72 \x00 \x65 \x00 \x85*hs \x7f \x00 \x00 \x00 \xa0 \x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00 ...</pre> |
| OUTPUT | |

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

| | |
|--------|--|
| TYPE | <code>"string", "string", "string", "PendingIntent", "PendingIntent"</code> |
| DATA | <code>\x0A \x00 \x00 \x00 \x37 \x00 \x38 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x31 \x00 \x32 \x00 \x33 \x00 \x34 \x00 \x00 \x00 \x00 \x00 \x08 \x00 \x00 \x00 \x48 \x00 \x69 \x00 \x20 \x00 \x74 \x00 \x68 \x00 \x65 \x00 \x72 \x00 \x65 \x00 \x85*hs \x7f \x00 \x00 \x00 \xa0 \x00 \x00 \x00 \x00 \x00 \x00 \x00 ...</code> |
| OUTPUT | <code>telephony.ISms.sendText(String destAddr = "7855551234", ...)</code> |

- ▶ `Type[0] = Primitive "string"`
- ▶ Use `ReadString()` (and increment data offset by length of string)

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

| | |
|--------|--|
| TYPE | "string", "string", "string", "PendingIntent", "PendingIntent" |
| DATA | <pre>\x0A \x00 \x00 \x00 \x37 \x00 \x38 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x31 \x00 \x32 \x00 \x33 \x00 \x34 \x00 \x00 \x00 \x00 \x00 \x08 \x00 \x00 \x00 \x48 \x00 \x69 \x00 \x20 \x00 \x74 \x00 \x68 \x00 \x65 \x00 \x72 \x00 \x65 \x00 \x85*hs \x7f \x00 \x00 \x00 \xa0 \x00 \x00 \x00 \x00 \x00 \x00 \x00 ...</pre> |
| OUTPUT | <pre>com.android.internal.telephony.ISms.sendText(String destAddr = "7855551234", String srcAddr = null, String text = "Hi there", ...)</pre> |

- ▶ Type[1] and Type[2] are also Primitive "string"
- ▶ Use ReadString() (and increment data offset by length of strings)

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

| | |
|--------|--|
| TYPE | "string","string", "string", "PendingIntent", "PendingIntent" |
| DATA | <pre>\x0A \x00 \x00 \x00 \x37 \x00 \x38 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x31 \x00 \x32 \x00 \x33 \x00 \x34 \x00 \x00 \x00 \x00 \x00 \x08 \x00 \x00 \x00 \x48 \x00 \x69 \x00 \x20 \x00 \x74 \x00 \x68 \x00 \x65 \x00 \x72 \x00 \x65 \x00 \x85*hs \x7f \x00 \x00 \x00 \xa0 \x00 \x00 \x00 \x00 \x00 \x00 \x00 ...</pre> |
| OUTPUT | <pre>com.android.internal.telephony.ISms.sendText(String destAddr = "7855551234", String srcAddr = null, String text = "Hi there", Intent sentIntent { type = BINDER_TYPE_HANDLE, flags = 0x7F FLAT_BINDER_FLAG_ACCEPT_FDS handle = 0xa, cookie = 0x0 }, ...)</pre> |

- ▶ Type[3] = IBinder "PendingIntent"
- ▶ Unmarshal using `com.Android.Intent` (AIDL) and increment buffer pointer
- ▶ Handle points to data to be unmarshalled in a previous Binder (ioctl) call

AUTOMATIC UNMARSHALLING ORACLE: SMS EXAMPLE

| | |
|--------|--|
| TYPE | "string","string", "string", "PendingIntent" , "PendingIntent" |
| DATA | <pre>\x0A \x00 \x00 \x00 \x37 \x00 \x38 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x35 \x00 \x31 \x00 \x32 \x00 \x33 \x00 \x34 \x00 \x00 \x00 \x00 \x00 \x08 \x00 \x00 \x00 \x48 \x00 \x69 \x00 \x20 \x00 \x74 \x00 \x68 \x00 \x65 \x00 \x72 \x00 \x65 \x00 \x85*hs \x7f \x00 \x00 \x00 \xa0 \x00 \x00 \x00 \x00 \x00 \x00 \x00 ...</pre> |
| OUTPUT | <pre>com.android.internal.telephony.ISms.sendText(String destAddr = "7855551234", String srcAddr = null, String text = "Hi there", Intent sentIntent { Intent("SENT") }, ...)</pre> |

- ▶ Each handle is paired with a parcelable object
- ▶ CopperDroid sends each handle and parcelable object to the Oracle

Outputs observed from CopperDroid

FILESYSTEM TRANSACTIONS

```
1 "class": "FS ACCESS",
2 "low": [
3   {
4     "blob": "{ 'flags': 131072, 'mode': 1, 'filename': u'/etc/media_codecs.xml' }",
5     "id": 187369,
6     "sysname": "open",
7     "ts": "1455718126.798",
8   },
9   {
10    "blob": "{ 'size': 4096L, 'filename': u'/etc/media_codecs.xml' }",
11    "id": 187371,
12    "sysname": "read",
13    "ts": "1455718126.798",
14    "xref": 187369
15  },
16  {
17    "blob": "{ 'filename': u'/etc/media_codecs.xml' }",
18    "id": 187389,
19    "sysname": "close",
20    "ts": "1455718126.799",
21    "xref": 187369
22  }
23 ],
24 "procname": "/system/bin/mediaserver"
```

NETWORK TRANSACTIONS

```
1 "class": "NETWORK ACCESS",
2 "low": [
3   {
4     "blob": "{ 'socket domain': 10, 'socket type': 1, 'socket protocol': 0 }",
5     "id": 62,
6     "sysname": "socket",
7     "ts": "1445024980.686",
8   },
9   {
10    "blob": "{ 'host': '::ffff:134.219.148.11', 'port': 80, 'returnValue': 0 }",
11    "id": 63,
12    "sysname": "connect",
13    "ts": "1445024980.687",
14  },
15  {
16    "blob": "=%22%27GET+%2Findex.html+HTTP%2F1.1%5C%5Cr%5C%5CnUser-Agent%3A+Dalvik%2F1.6.0+%28Linux%3B+U%3B+Android+4.4.4%3B+sdk+Build%2FKK%29%5C%5Cr%5C%5CnHost%3A+s2lab.isg.rhul.ac.uk%5C%5Cr%5C%5CnConnection%3A+Keep-Alive%5C%5Cr%5C%5CnAccept-Encoding%3A+gzip%5C%5Cr%5C%5Cn%5C%5Cr%5C%5Cn%27%22",
17    "id": 164,
18    "sysname": "sendto",
19    "ts": "1445024980.720",
20  },
21 ],
22 "procname": "com.cd2.nettest.nettest",
23 "subclass": "HTTP"
```

NETWORK TRANSACTIONS

```
1 "class": "NETWORK ACCESS",
2 "low": [
3   {
4     "blob": "{ 'socket domain': 10, 'socket type': 1, 'socket protocol': 0 }",
5     "id": 62,
6     "sysname": "connect",
7     "ts": "1440034800.000000",
8   },
9   {
10    "blob": "{ 'socket domain': 80, 'socket type': 1, 'socket protocol': 0 }",
11    "id": 63,
12    "sysname": "connect",
13    "ts": "1440034800.000000",
14  },
15  {
16    "blob": "{ 'socket domain': 80, 'socket type': 1, 'socket protocol': 0 }",
17    "id": 64,
18    "sysname": "connect",
19    "ts": "1440034800.000000",
20  },
21 ],
22 "procname": "com.cd2.nettest.nettest",
23 "subclass": "HTTP"
```

- ▶ Composite behaviors (e.g., filesystem and network transactions)
- ▶ We perform a value-based data flow analysis by building a system call-related DDG and def-use chains
 - Each observed system call is initially considered as an unconnected node
 - Forward slicing inserts edges for every inferred dependence between two calls
 - Nodes and edges are annotated with the system call argument constraints
 - Annotations needed for the creation of def-use chains
- Def-use chains relate the output value of specific system calls to the input of (non-necessarily adjacent) others

BINDER TRANSACTIONS

```
1 "class": "SMS SEND",
2 "low": [
3     {
4         "blob": {
5             "method": "sendText",
6             "params": [
7                 "callingPkg = com.load.wap",
8                 "destAddr = 3170",
9                 "scAddr = null",
10                "text = 999287346 418 Java (256) vip 2012-02-25 17:47:56 newoperastore.ru y"
11            ]
12        },
13        "method_name": "com.android.internal.telephony.ISms.sendText()",
14        "sysname": "ioctl",
15        "ts": "1444337887.816",
16        "type": "BINDER"
17    }
18 ],
19 "procname": "com.load.wap"
```

Challenges in Dynamic Analysis (for Android) and Living with it

1. Android apps are interactive and hard to stimulate
2. Evasive analyses (not further discussed here)¹¹

¹¹<http://roberto.greyhats.it/pubs/woot09.pdf> and
https://seclab.cs.ucsb.edu/media/uploads/papers/oakland10_hybrid.pdf and
<https://users.ece.cmu.edu/~tvidas/papers/ASIACCS14.pdf> and
<https://www.sba-research.org/wp-content/uploads/publications/mostAndroid.pdf>



COVERAGE IS PARTIAL IN DYNAMIC ANALYSIS

```
invoke-virtual {v7}, Ljava/lang/String;->length()I
move-result v1
const/16 v3, 0x12
if-eq v1, v3, :cond_0
iget-object v1, p0, Lcom/example/xxshenqi/RegisterActivity
:cond_0
```

Improving coverage

Coverage can be improved by generating test cases through sound stimulation strategies that cover all paths in the app.

STIMULATION: A HARD PROBLEM

- ▶ Automating interactions with apps meaningfully is hard
- ▶ We often do not have the context for a successful invocation and successful invocations may depend on a complex sequence of events
 - Imagine a FaceBook notification when someone logs into your account from a new device
 - How do we generate a test case to simulate this situation?

- ▶ A pragmatic approach is to stimulate what we can
- ▶ The most commonly used tool to stimulate apps is MonkeyRunner

MonkeyRunner

It is a program that provides APIs to control the execution of an Android app running on an emulator for testing purpose

- ▶ The onus is on the tester to provide the right inputs through MonkeyRunner to stimulate the app

- Install an apk

MONKEYRUNNER: WHAT CAN IT DO?

- ▶ Install an apk
- ▶ Invoke an activity with the installed apk

MONKEYRUNNER: WHAT CAN IT DO?

- ▶ Install an apk
- ▶ Invoke an activity with the installed apk
- ▶ Send keystrokes to type a text message

MONKEYRUNNER: WHAT CAN IT DO?

- ▶ Install an apk
- ▶ Invoke an activity with the installed apk
- ▶ Send keystrokes to type a text message
- ▶ Click on arbitrary locations on the screen

MONKEYRUNNER: WHAT CAN IT DO?

- ▶ Install an apk
- ▶ Invoke an activity with the installed apk
- ▶ Send keystrokes to type a text message
- ▶ Click on arbitrary locations on the screen
- ▶ Take screenshots (useful for feedback)



MONKEYRUNNER: WHAT CAN IT DO?

- ▶ Install an apk
- ▶ Invoke an activity with the installed apk
- ▶ Send keystrokes to type a text message
- ▶ Click on arbitrary locations on the screen
- ▶ Take screenshots (useful for feedback)
- ▶ Wake up device if it goes to sleep

- ▶ It is still notoriously difficult to write a good stimulation script

- ▶ It is still notoriously difficult to write a good stimulation script
- ▶ Android apps are highly interactive and you need to get right both the context and location of the stimulation in the GUI



- ▶ It is still notoriously difficult to write a good stimulation script
- ▶ Android apps are highly interactive and you need to get right both the context and location of the stimulation in the GUI
- ▶ A change in the GUI means a new test script needs to be developed

- ▶ It is still notoriously difficult to write a good stimulation script
- ▶ Android apps are highly interactive and you need to get right both the context and location of the stimulation in the GUI
- ▶ A change in the GUI means a new test script needs to be developed

MONKEYRUNNER VS. STATE-OF-THE-ART

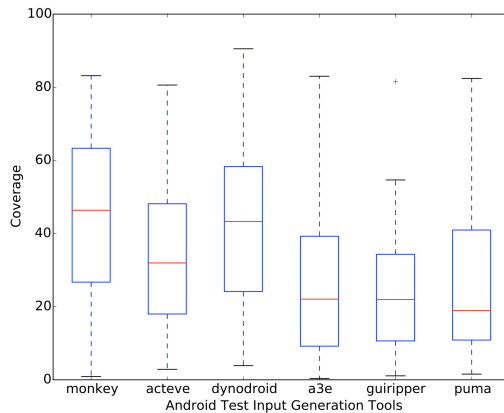


Figure: A comparison of Android test generation tools ¹²

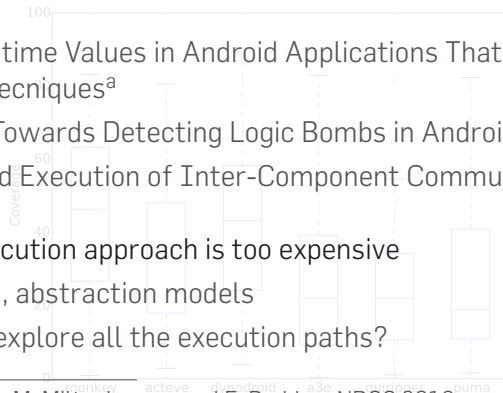
¹²Source: Automated Test Input Generation for Android: Are We There Yet? by Shauvik Roy Choudhary et. al., 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)

Quo Vadis?

- ▶ Harvesting Runtime Values in Android Applications That Feature Anti-Analysis Techniques^a
- ▶ TriggerScope: Towards Detecting Logic Bombs in Android Apps^b
- ▶ TeICC: Targeted Execution of Inter-Component Communications in Android^c

A pure symbolic execution approach is too expensive

- ▶ State explosion, abstraction models
- ▶ Do we need to explore all the execution paths?



^aS. Rasthofer, S. Arzt, M. Miltenberger, and E. Bodden. NDSS 2016.

^bY. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, G. Vigna. IEEE Symposium on Security & Privacy, 2016

^cMaqsood Ahmad, Valerio Costamagna, Bruno Crispo, and Francesco Bergadano. ACM Symposium on Applied Computing (SAC), 2017

Representation and Analysis of Software

Dynamic Analysis for Android

Classification of Android Malware

DroidScribe

Machine Learning and Malicious Software: Quo Vadis?

RQ2—DroidScribe

Classifying Android Malware with Runtime Behavior

TAKE AWAYS

- ▶ CopperDroid reproduces execution footprint of an App
- ▶ It re-constructs high-level semantics automatically
 - More meaningful actions
 - Less clutter from detailed OS-call trace

- ▶ CopperDroid reproduces execution footprint of an App
- ▶ It re-constructs high-level semantics automatically

CopperDroid: Automatic Reconstruction of Android Malware Behaviors

- ▶ <http://s2lab.isg.rhul.ac.uk/papers/files/ndss2015.pdf>
(Kimberly Tam, Salahuddin J. Khan, Aristide Fattori, and Lorenzo Cavallaro. NDSS 2015)
- ▶ Check it out @ <http://copperdroid.isg.rhul.ac.uk>
- ▶ Check MobSec out @ <http://s2lab.isg.rhul.ac.uk/projects/mobsec/>
- ▶ New roll out soon—stay tuned and drop me a line:
lorenzo.cavallaro@rhul.ac.uk

- ▶ CopperDroid reproduces execution footprint of an App
- ▶ It re-constructs high-level semantics automatically
 - More meaningful actions
 - Less clutter from detailed OS-call trace

Next steps...

- ▶ Can we use this information for classifying malware Apps?
- ▶ Is the level of reconstructed information/behaviour sufficient?
- ▶ What is the quality of classification that we can achieve?

RESEARCH OBJECTIVES

- ▶ Runtime behaviors as discriminator of maliciousness
 - Independent of any syntactic artifact
 - Visible in managed and native code alike
- ▶ Family Identification
 - Crucial for analysis of threats and mitigation planning

Goal Dynamic analysis for classification under challenging conditions

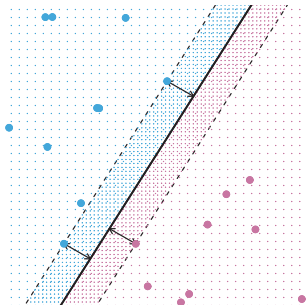
Our contributions¹³

- ▶ RQ2.1: What is the best level abstraction?
- ▶ RQ2.2: Can we deal with sparse behaviors?

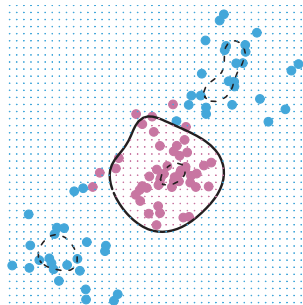
¹³Dash et al., "DroidScribe: Classifying Android Malware Based on Runtime Behavior" in IEEE S&P Workshop MoST 2016

MACHINE LEARNING COMPONENT

- ▶ Use existing malware classified into families as training data
- ▶ Use Support Vector Machines as the classification algorithm



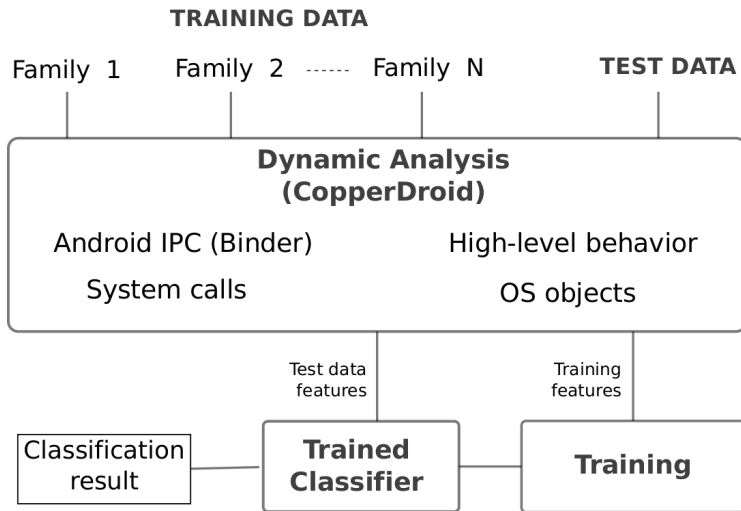
Linear function



Radial-basis function

Source: An Introduction to Statistical Learning–G. James et al.

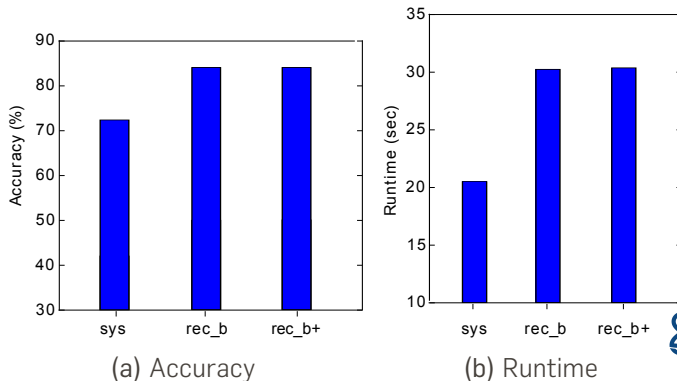
OVERVIEW OF THE CLASSIFICATION FRAMEWORK



SYSTEM-CALLS VS. ABSTRACT BEHAVIORS

RQ2.1 What is the best level of abstraction?

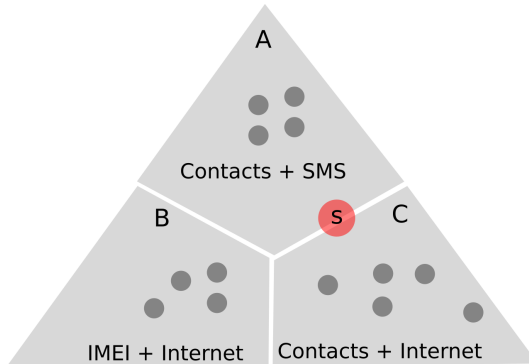
- ▶ Experiments on the Drebin dataset (5,246 malware samples).
- ▶ Reconstructing Binder calls adds 141 meaningful features.
- ▶ High level behaviors added 3 explanatory features.



- ▶ Dynamic analysis is limited by code coverage
- ▶ Classifier has only partial information about behaviors
- ▶ Identify when malware cannot be reliably classified into only one family
 - Based on a measure of the statistical confidence
- ▶ Helpful human analyst by identifying the top matching families, supported by statistical evidence

CLASSIFICATION FROM OBSERVED FEATURES

- ▶ When more than one choice of similar likelihood exists, ...
- ▶ ... traditional classification algorithms are prone to error



Contacts accessed by S

CLASSIFICATION WITH STATISTICALLY CONFIDENCE

Conformal Predictor (CP)

- ▶ A statistical learning algorithm tailored at classification tasks
- ▶ Provides statistical evidence on the results

Credibility

Supports how good a sample fits into a class

Confidence

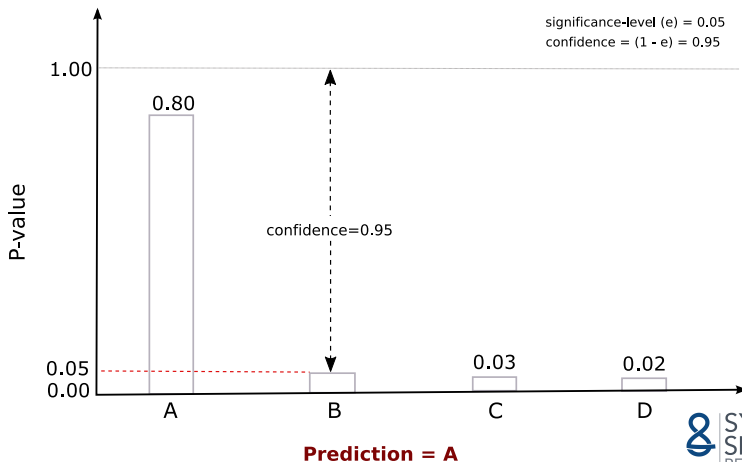
Indicates if there are other good choices

Robust Against Outliers

Aware of values from other members of the same class

IN AN IDEAL WORLD

Given a new object s , conformal predictor picks the class with the highest p-value and return a singular prediction.

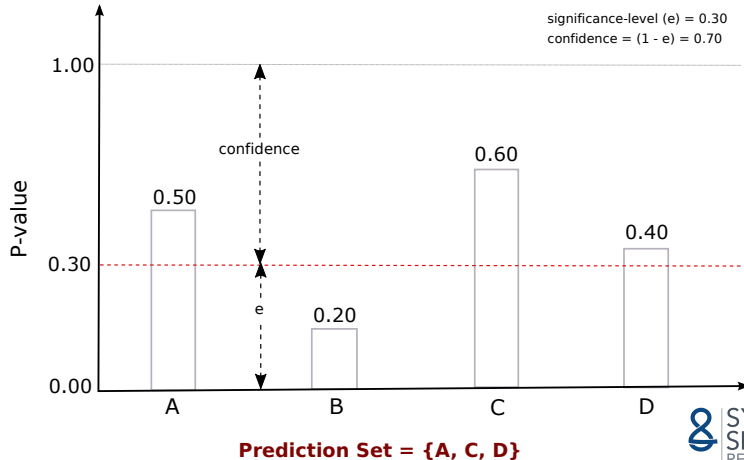


SYSTEMS
SECURITY
RESEARCH LAB



OBTAINING PREDICTION SETS

Given a new object s , we can set a significance-level e for p-values and obtain a prediction set Γ^e includes labels whose p-value is greater than e for the sample.



WHEN TO USE CONFORMAL PREDICTION?

- ▶ CP is an expensive algorithm
 - For each sample, we need to derive a p-value for each class
 - Computation complexity of $O(nc)$ where n is number of samples and c is the number of classes

WHEN TO USE CONFORMAL PREDICTION?

- ▶ CP is an expensive algorithm
 - For each sample, we need to derive a p-value for each class
 - Computation complexity of $O(nc)$ where n is number of samples and c is the number of classes

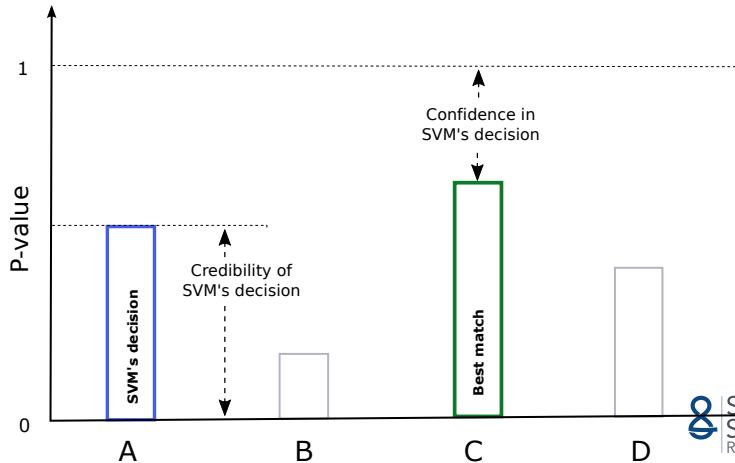
Conformal Evaluation¹

- ▶ Provide statistical evaluation of the quality of a ML algorithm
 - Quality threshold to understand when should be trusting SVM
 - Statistical evidences of the choices of SVM
 - Selectively invoke CP to alleviate runtime performance

¹Roberto Jordaney Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nourtdinov, and Lorenzo Cavallaro. "Transcend: Detecting Concept Drift in Malware Classification Models." In USENIX Security Symposium, Vancouver, CA, 2017

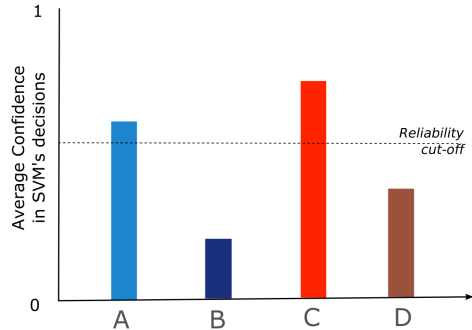
STEP 1. COMPUTING CONFIDENCE IN TRAINING DECISIONS

- ▶ During training, compute p-values for each sample for each class
- ▶ Compute the confidence in the decision for each sample



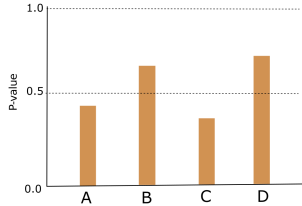
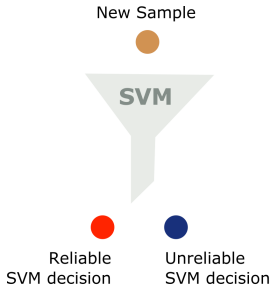
STEP 2. USING CLASS-LEVEL CONFIDENCE SCORES







- ▶ For each class, calculate the mean confidence for all decisions mapping to the class
- ▶ Use the median of the class-level confidence across all classes as a reliability threshold



Identifying reliability cut-off during training

STEP 3. INVOKING THE CONFORMAL PREDICTOR



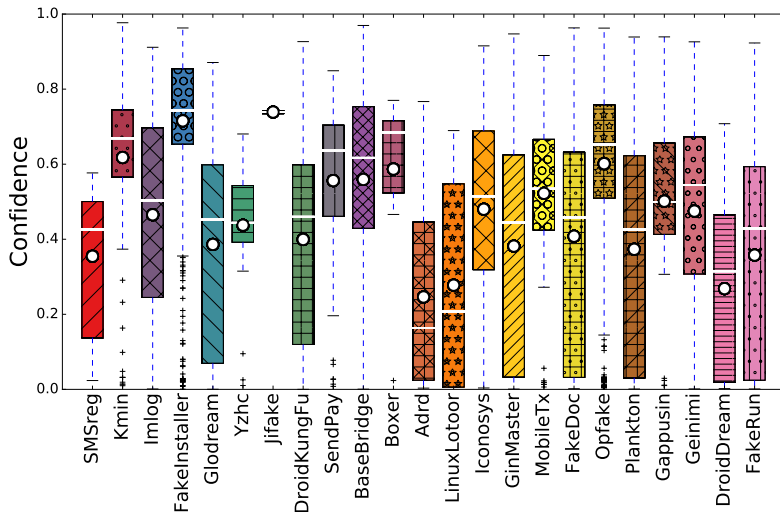
| Threshold | Prediction Set |
|-----------|---|
| 1.0 | \emptyset |
| 0.5 |   |
| 0.0 |     |

CONFORMAL PREDICTION

Threshold

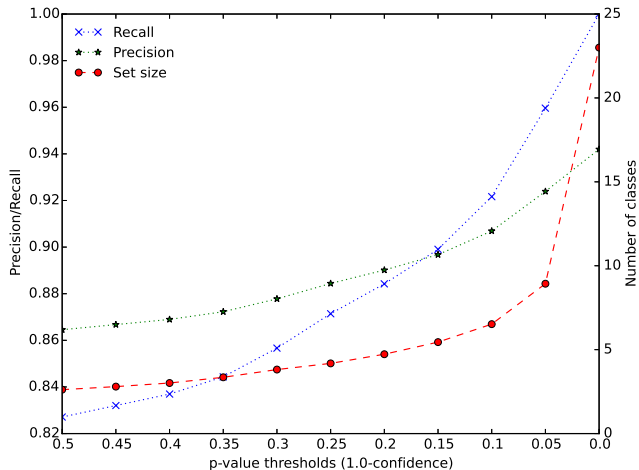
The threshold for picking prediction sets is fully tunable

CONFIDENCE OF CORRECT SVM DECISIONS



ACCURACY VS. PREDICTION SET SIZE

RQ2.2 Can we deal with sparse behaviors?



- Accuracy improves with the prediction set size



RQ3—Concept Drift

Statistical Evaluation of ML Classifiers

Representation and Analysis of Software

Dynamic Analysis for Android

Classification of Android Malware

Machine Learning and Malicious Software: Quo Vadis?

- Conformal Evaluator

- Experimental Results

- Take Aways

- ▶ Malware: pressing threat to the security of the Internet

MACHINE LEARNING AND MALICIOUS SOFTWARE: QUO VADIS?

- ▶ Malware: pressing threat to the security of the Internet
- ▶ Machine learning (often with program analysis): promising technique(s) to combat malware at scale

- ▶ Malware: pressing threat to the security of the Internet
- ▶ Machine learning (often with program analysis): promising technique(s) to combat malware at scale
 - Windows malware binary and multi-class classification
 - e.g., Holmes (IEEE S&P10), DIMVA08, CODASPY16
 - Malicious network traffic clustering and classification
 - e.g., BotMiner (USENIXSec08), FIRMA (RAID13)
 - Android malware binary and multi-class classification
 - e.g., Drebin (NDSS14), Marvin (COMPSAC15), DroidScribe (MoST16)

MACHINE LEARNING AND MALICIOUS SOFTWARE: QUO VADIS?

- ▶ Malware: pressing threat to the security of the Internet
- ▶ Machine learning (often with program analysis): promising technique(s) to combat malware at scale
 - Windows malware binary and multi-class classification
 - e.g., Holmes (IEEE S&P10), DIMVA08, CODASPY16
 - Malicious network traffic clustering and classification
 - e.g., BotMiner (USENIXSec08), FIRMA (RAID13)
 - Android malware binary and multi-class classification
 - e.g., Drebin (NDSS14), Marvin (COMPSAC15), DroidScribe (MoST16)

In general, high TPR and low FPR in k-fold cross validation settings (ROC curve)

- ▶ It looks like we have solved the problem...

Usually, a 2-phase process:

1. Training: build a model M , given labeled objects
2. Testing: given M , predict the labels of unknown objects

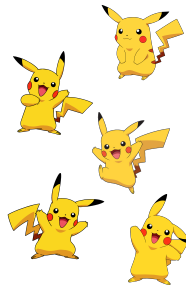
Objects are described as vectors of features

Usually, a 2-phase process:

1. Training: build a model M , given labeled objects
2. Testing: given M , predict the labels of unknown objects

Objects are described as vectors of features

Pikachu



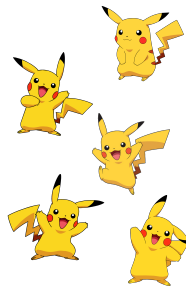
Charmander

Usually, a 2-phase process:

1. Training: build a model M , given labeled objects
2. Testing: given M , predict the labels of unknown objects

Objects are described as vectors of features

Pikachu



Charmander

MACHINE LEARNING CLASSIFICATION PROBLEM: CONCEPT DRIFT

- ▶ Concept drift is the **change in the statistical properties** of an object in unforeseen ways
- ▶ Drifted objects will likely be wrongly classified

Hitmonlee
(new pokémon family)



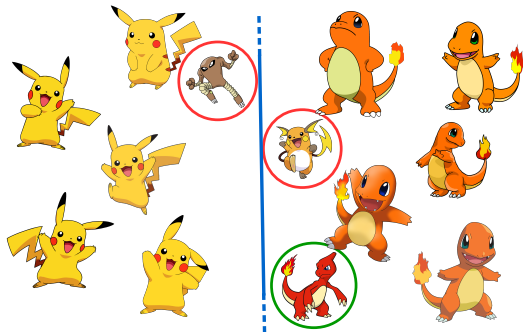
Raichu
(evolution of Pikachu)



Charmeleon
(evolution of Charmander)

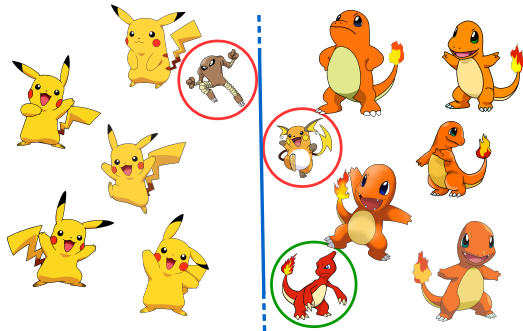
MACHINE LEARNING CLASSIFICATION PROBLEM: CONCEPT DRIFT

- ▶ Concept drift is the **change in the statistical properties** of an object in unforeseen ways
- ▶ Drifted objects will likely be wrongly classified



MACHINE LEARNING CLASSIFICATION PROBLEM: CONCEPT DRIFT

- ▶ Concept drift is the **change in the statistical properties** of an object in unforeseen ways
- ▶ Drifted objects will likely be wrongly classified



Of course, the problem exists in multiclass classification settings...



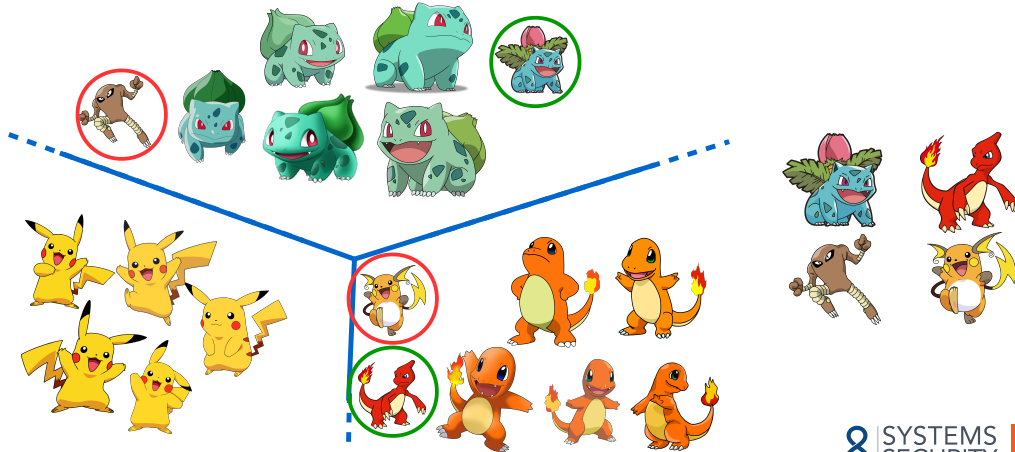
SYSTEMS
SECURITY
RESEARCH LAB







ROYAL
HOLLOWAY
UNIVERSITY
OF LONDON

MACHINE LEARNING CLASSIFICATION PROBLEM: CONCEPT DRIFT

- Multiclass classification is a generalization of the binary case



- ▶ In non-stationary contexts classifiers will suffer from concept drift due to:
 - malware evolution 
 - new malware families 
- ▶ Need a way to **assess the predictions** of classifiers
 - Ideally classifier-agnostic assessments
- ▶ Need to identify objects that fit a model and those drifting away

- ▶ In non-stationary contexts classifiers will suffer from concept drift due to:
 - malware evolution 
 - new malware families 
- ▶ Need a way to **assess the predictions** of classifiers
 - Ideally classifier-agnostic assessments

Our Contributions: identify objects that fit a model and those drifting away

- Conformal Evaluator: statistical evaluation of ML classifiers
- Per-class quality threshold to identify reliable and unreliable predictions

^aRoberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, and Lorenzo Cavallaro. "Transcend: Detecting Concept Drift in Malware Classification". USENIX Security Symposium, Vancouver, CA, Aug 2017.



- ▶ Assesses decisions made by a classifier
 - Mark each decision as reliable or unreliable
- ▶ Builds and makes use of p-value as assessment criteria
- ▶ Computes per-class thresholds to divide reliable decisions from unreliable ones

CONFORMAL EVALUATOR: P-VALUE?

- ▶ Used to measure "how well" a sample fits into a single class
- ▶ Conformal Evaluator computes a p-value for each class, for each test element

Definition

α_t = Non-conformity score for test element t

$\forall i \in \mathcal{K}, \alpha_i$ = Non-conformity score for train element i

$$\text{p-value} = \frac{|\{i : \alpha_i \geq \alpha_t\}|}{|\mathcal{K}|}$$

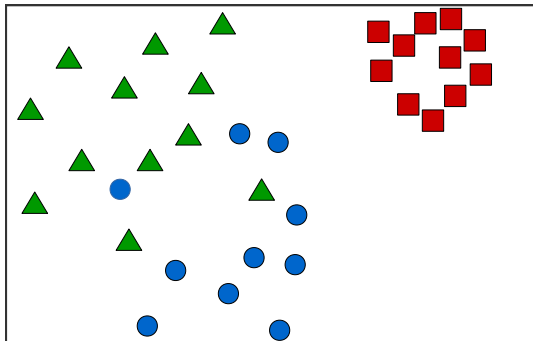
\mathcal{K} = Total number of element

P-value

Ratio between the number of training elements that are more dissimilar than the element under test

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

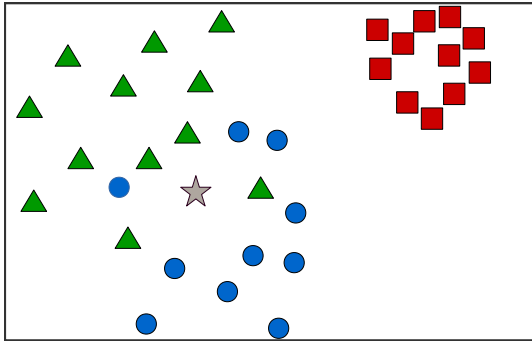
ML classifier:
distance from centroid



1. Setting: 3-class classification

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

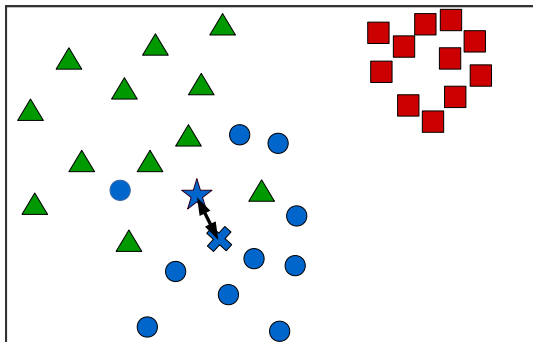
ML classifier:
distance from centroid



1. Setting: 3-class classification
2. Test object

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

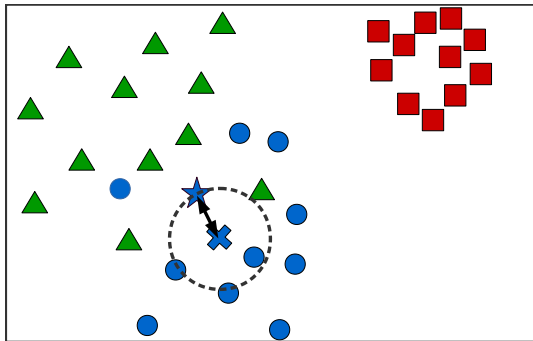
ML classifier:
distance from centroid



1. Setting: 3-class classification
2. Test object
 - 3.1 Compute distance to blue class

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

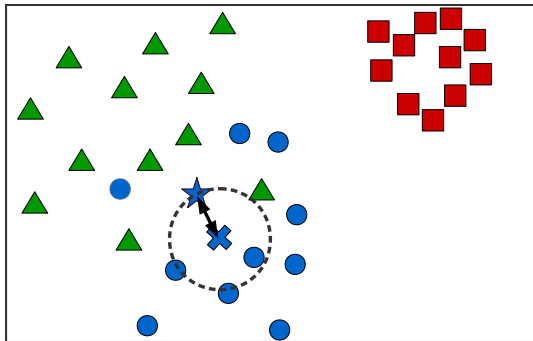
ML classifier:
distance from centroid



1. Setting: 3-class classification
2. Test object
 - 3.1 Compute distance to **blue** class
 - 3.2 How many objects are more dissimilar than the one under test?

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

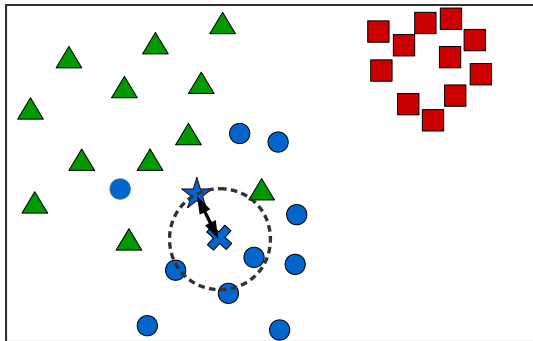
ML classifier:
distance from centroid



1. Setting: 3-class classification
2. Test object
 - 3.1 Compute distance to **blue** class
 - 3.2 How many objects are more dissimilar than the one under test?
 - 3.3 9

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

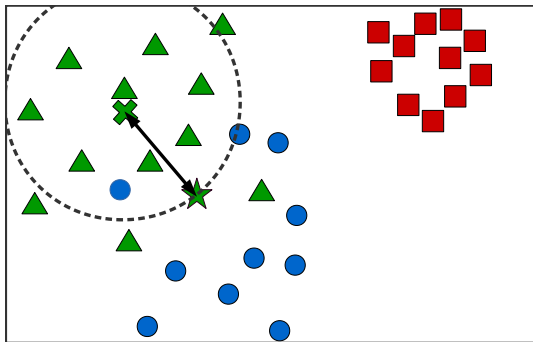
ML classifier:
distance from centroid



1. Setting: 3-class classification
2. Test object
 - 3.1 Compute distance to blue class
 - 3.2 How many objects are more dissimilar than the one under test?
 - 3.3 9
 - 3.4 P-value $\star = \frac{9}{10}$

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

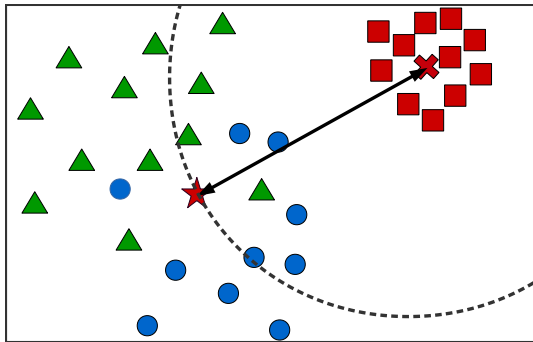
Machine learning classifier:
distance from centroid



1. Setting: 3-class classification
2. Test object
 - 4.1 Calculate distance to **green** class
 - 4.2 How many objects are more dissimilar than the one under test?
 - 4.3 4
 - 4.4 P-value $\star = \frac{4}{12}$

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

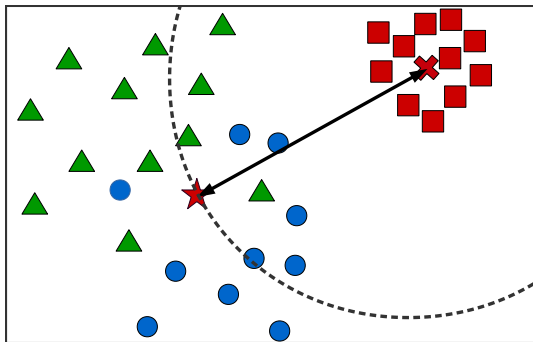
Machine learning classifier:
distance from centroid



1. Setting: 3-class classification
2. Test object
 - 5.1 Calculate distance to **red** class
 - 5.2 How many objects are more dissimilar than the one under test?
 - 5.3 0
 - 5.4 P-value $\star = \frac{0}{11}$

CONFORMAL EVALUATOR: P-VALUE EXAMPLE

Machine learning classifier:
distance from centroid

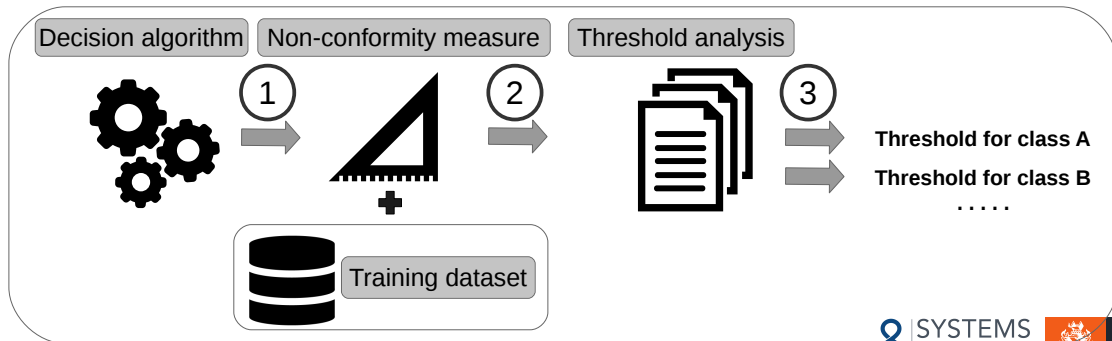


1. Setting: 3-class classification
2. Test object
 - 5.1 Calculate distance to **red** class
 - 5.2 How many objects are more dissimilar than the one under test?
 - 5.3 0
 - 5.4 P-value $\star = \frac{0}{11}$

Let's see how p-values are used within Conformal Evaluator.

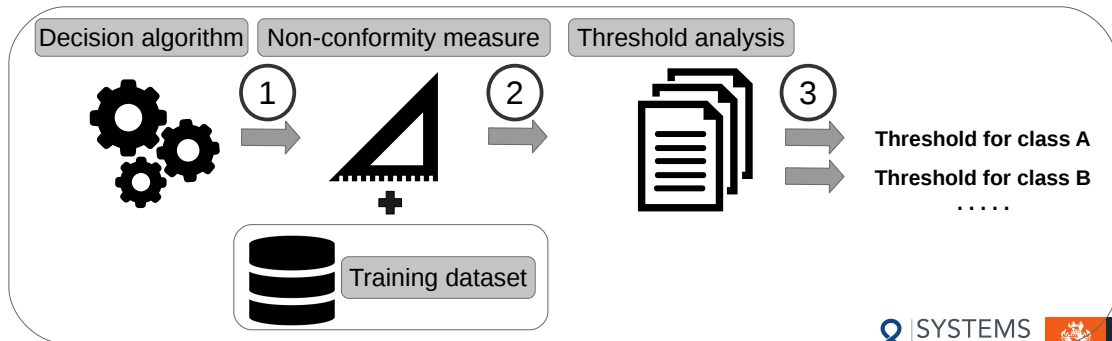
CONFORMAL EVALUATOR: HOW DOES IT WORK?

1. Extracts the non-conformity measure (NCM) from the decision making algorithm
 - NCM provides non-conformity scores for p-value computations
 - Example: distance from hyperplane, Random Forest probability (adapted to satisfy the non-conformity requirement)



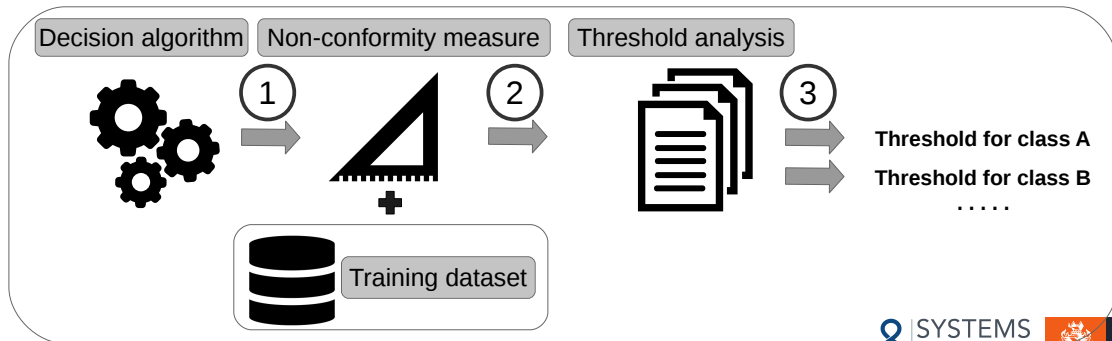
CONFORMAL EVALUATOR: HOW DOES IT WORK?

1. Extracts the non-conformity measure (NCM) from the decision making algorithm
2. Builds p-values for all training samples in a cross-validation fashion



CONFORMAL EVALUATOR: HOW DOES IT WORK?

1. Extracts the non-conformity measure (NCM) from the decision making algorithm
2. Builds p-values for all training samples in a cross-validation fashion
3. Computes per-class threshold to divide reliable predictions from unreliable ones



Customizable constraints:

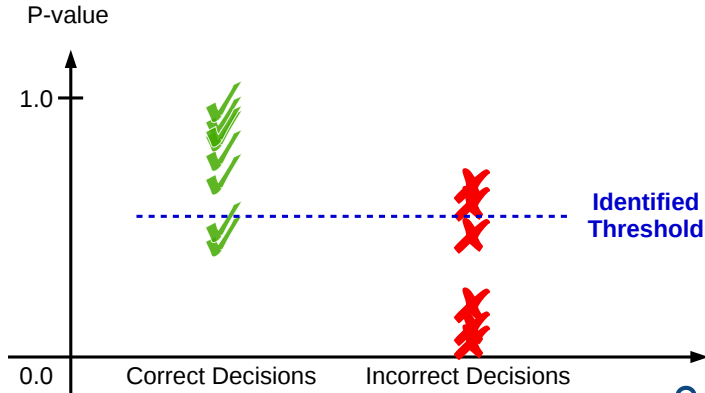
- ▶ Desired performance (of the predictions marked as reliable)
 - E.g.: high-level performance will raise the threshold
- ▶ Number of unreliable prediction tolerated
 - E.g.: low number of unreliable prediction will lower the threshold

Assumptions

- ▶ Performance of non-drifted elements are similar to the one declared by the algorithm
- ▶ Predictions with high confidence will have higher p-values

CONFORMAL EVALUATOR: IDENTIFYING PER-CLASS THRESHOLDS

- ▶ We use the p-values and prediction labels from training samples
- ▶ From the thresholds that satisfy the constraints we chose the one that maximize one or the other



- ▶ Binary case study: Android malware detection algorithm
 - Reimplemented Drebin¹⁴ algorithm with similar results (0.95-0.92 precision-recall on malicious apps and 0.99-0.99 precision-recall on benign apps)
 - Static features of Android apps, linear SVM (used as NCM)
 - Concept drift scenario: malware evolution
- ▶ Multiclass case study: Microsoft malware classification algorithm
 - Solution to Microsoft Kaggle competition¹⁵, ranked among the top ones
 - Static features from Windows PE binaries, Random Forest (used as NCM)
 - Concept drift scenario: family discovery

¹⁴ Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In 21st Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 23-26, 2014.

¹⁵ KAGGLE INC. Microsoft Malware Classification Challenge (BIG 2015). <https://www.kaggle.com/c/malware-classification>

- ▶ Binary case study: Android malware detection algorithm
 - Reimplemented Drebin¹⁴ algorithm with similar results (0.95-0.92 precision-recall on malicious apps and 0.99-0.99 precision-recall on benign apps)
 - Static features of Android apps, linear SVM (used as NCM)
 - Concept drift scenario: malware evolution

¹⁴Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In 21st Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 23-26, 2014.

¹⁵KAGGLE INC. Microsoft Malware Classification Challenge (BIG 2015). <https://www.kaggle.com/c/malware-classification>

EXPERIMENTAL RESULTS: BINARY CLASSIFICATION (MALWARE EVOLUTION)

- ▶ Drebin dataset: samples collected from 2010 to 2012
- ▶ Marvin dataset¹⁶: malware apps collected from 2010 to 2014 (no duplicates)
 - We expect some object to drift from objects in the Drebin dataset

| Drebin Dataset | |
|----------------|---------|
| Type | Samples |
| Benign | 123,435 |
| Malware | 5,560 |

| Marvin Dataset | |
|----------------|---------|
| Type | Samples |
| Benign | 9,592 |
| Malware | 9,179 |

¹⁶ Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. MARVIN: Efficient and Comprehensive Mobile App Classification Through Static And Dynamic Analysis. In 39th IEEE Annual Computer Software and Applications Conference (COMPSAC), Taichung, Taiwan, July 1-6, 2019.

Experiment: Drift Confirmation

- ▶ Training dataset: Drebin dataset
- ▶ Testing dataset: 4,500 benign and 4,500 malicious random samples from Marvin dataset

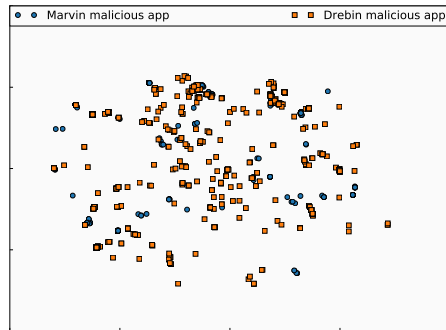
| Original label | Prediction label | | Recall |
|----------------|------------------|-----------|--------|
| | Benign | Malicious | |
| Benign | 4,498 | 2 | 1 |
| Malicious | 2,890 | 1,610 | 0.36 |
| Precision | 0.61 | 1 | |

EXPERIMENTAL RESULTS: BINARY CLASSIFICATION (MALWARE EVOLUTION)

Experiment: Drift Confirmation

- ▶ Training dataset: Drebin dataset
- ▶ Testing dataset: 4,500 benign and 4,500 malicious random samples from Marvin dataset

| Original label | Prediction label | | Recall |
|----------------|------------------|-----------|--------|
| | Benign | Malicious | |
| Benign | 4,498 | 2 | 1 |
| Malicious | 2,890 | 1,610 | 0.36 |
| Precision | 0.61 | 1 | |

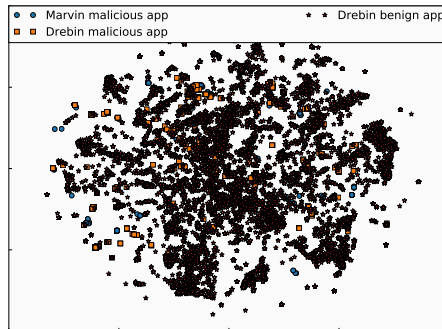


EXPERIMENTAL RESULTS: BINARY CLASSIFICATION (MALWARE EVOLUTION)

Experiment: Drift Confirmation

- ▶ Training dataset: Drebin dataset
- ▶ Testing dataset: 4,500 benign and 4,500 malicious random samples from Marvin dataset

| Original label | Prediction label | | Recall |
|----------------|------------------|-----------|--------|
| | Benign | Malicious | |
| Benign | 4,498 | 2 | 1 |
| Malicious | 2,890 | 1,610 | 0.36 |
| Precision | 0.61 | 1 | |



Experiment: Threshold Identification

- ▶ Training dataset: Drebin dataset
- ▶ Testing dataset: 4,500 benign and 4,500 malicious random samples from Marvin dataset
- ▶ Make use of Conformal Evaluator's prediction assessment algorithm
 - Constraints: F1-score of 0.99 and 0.76 of elements marked as reliable

| Original label | Prediction label | | Recall |
|----------------|------------------|-----------|--------|
| | Benign | Malicious | |
| Benign | 4,257 | 2 | 1 |
| Malicious | 504 | 1,610 | 0.76 |
| Precision | 0.89 | 1 | |

EXPERIMENTAL RESULTS: BINARY CLASSIFICATION (MALWARE EVOLUTION)

Experiment: Retraining

- ▶ Training dataset: Drebin dataset + samples marked as unreliable from previous experiment
- ▶ Testing dataset: 4,500 benign and 4,500 malicious random samples of Marvin dataset
(no sample overlap from previous experiment)

| Sample | Assigned label | | Recall |
|-----------|----------------|-----------|--------|
| | Benign | Malicious | |
| Benign | 4,413 | 87 | 0.98 |
| Malicious | 255 | 4,245 | 0.94 |
| Precision | 0.96 | 0.98 | |

EXPERIMENTAL RESULTS: BINARY CLASSIFICATION (MALWARE EVOLUTION)

Experiment: Threshold Comparison

- ▶ Compare probability- and p-value-based thresholds
 - Central tendency and dispersion points of true positive distribution
- ▶ Training dataset: Drebin dataset
- ▶ Testing dataset: 4,500 benign and 4,500 malicious apps from Marvin dataset (random sampling)

| | TPR (reliable predictions) | | TPR (unreliable predictions) | | FPR (reliable predictions) | | FPR (unreliable predictions) | |
|--------------|-------------------------------|-------------|---------------------------------|-------------|-------------------------------|-------------|---------------------------------|-------------|
| | p-value | probability | p-value | probability | p-value | probability | p-value | probability |
| 1st quartile | 0.9045 | 0.6654 | 0.0000 | 0.3176 | 0.0007 | 0.0 | 0.0000 | 0.0013 |
| Median | 0.8737 | 0.8061 | 0.3080 | 0.3300 | 0.0000 | 0.0 | 0.0008 | 0.0008 |
| Mean | 0.8737 | 0.4352 | 0.3080 | 0.3433 | 0.0000 | 0.0 | 0.0008 | 0.0018 |
| 3rd quartile | 0.8723 | 0.6327 | 0.3411 | 0.3548 | 0.0000 | 0.0 | 0.0005 | 0.0005 |



EXPERIMENTAL RESULTS: BINARY CLASSIFICATION (MALWARE EVOLUTION)

Experiment: Threshold Comparison

- ▶ Compare probability- and p-value-based thresholds
 - Central tendency and dispersion points of true positive distribution
- ▶ Training dataset: Drebin dataset
- ▶ Testing dataset: 4,500 benign and 4,500 malicious apps from Marvin dataset (random sampling)

| | TPR (reliable predictions) | | TPR (unreliable predictions) | | FPR (reliable predictions) | | FPR (unreliable predictions) | |
|--------------|-------------------------------|-------------|---------------------------------|-------------|-------------------------------|-------------|---------------------------------|-------------|
| | p-value | probability | p-value | probability | p-value | probability | p-value | probability |
| 1st quartile | 0.9045 | 0.6654 | 0.0000 | 0.3176 | 0.0007 | 0.0 | 0.0000 | 0.0013 |
| Median | 0.8737 | 0.8061 | 0.3080 | 0.3300 | 0.0000 | 0.0 | 0.0008 | 0.0008 |
| Mean | 0.8737 | 0.4352 | 0.3080 | 0.3433 | 0.0000 | 0.0 | 0.0008 | 0.0018 |
| 3rd quartile | 0.8723 | 0.6327 | 0.3411 | 0.3548 | 0.0000 | 0.0 | 0.0005 | 0.0005 |



EXPERIMENTAL RESULTS: BINARY CLASSIFICATION (MALWARE EVOLUTION)

Experiment: Threshold Comparison

- ▶ Compare probability- and p-value-based thresholds
 - Central tendency and dispersion points of true positive distribution
- ▶ Training dataset: Drebin dataset
- ▶ Testing dataset: 4,500 benign and 4,500 malicious apps from Marvin dataset (random sampling)

| | TPR (reliable predictions) | | TPR (unreliable predictions) | | FPR (reliable predictions) | | FPR (unreliable predictions) | |
|--------------|-------------------------------|-------------|---------------------------------|-------------|-------------------------------|-------------|---------------------------------|-------------|
| | p-value | probability | p-value | probability | p-value | probability | p-value | probability |
| 1st quartile | 0.9045 | 0.6654 | 0.0000 | 0.3176 | 0.0007 | 0.0 | 0.0000 | 0.0013 |
| Median | 0.8737 | 0.8061 | 0.3080 | 0.3300 | 0.0000 | 0.0 | 0.0008 | 0.0008 |
| Mean | 0.8737 | 0.4352 | 0.3080 | 0.3433 | 0.0000 | 0.0 | 0.0008 | 0.0018 |
| 3rd quartile | 0.8723 | 0.6327 | 0.3411 | 0.3548 | 0.0000 | 0.0 | 0.0005 | 0.0005 |



- Dataset: Microsoft Malware Classification Challenge (2015)

| Microsoft Malware Classification Challenge Dataset | | | |
|--|---------|----------------|---------|
| Malware | Samples | Malware | Samples |
| Ramnit | 1 541 | Obfuscator.ACY | 1 228 |
| Lollipop | 2 478 | Gatak | 1 013 |
| Kelihos_ver3 | 2 942 | Kelihos_ver1 | 398 |
| Vundo | 4 75 | Tracur | 751 |

EXPERIMENTAL RESULTS: MULTICLASS CLASSIFICATION (NEW FAMILY)

Experiment: Family Discovery

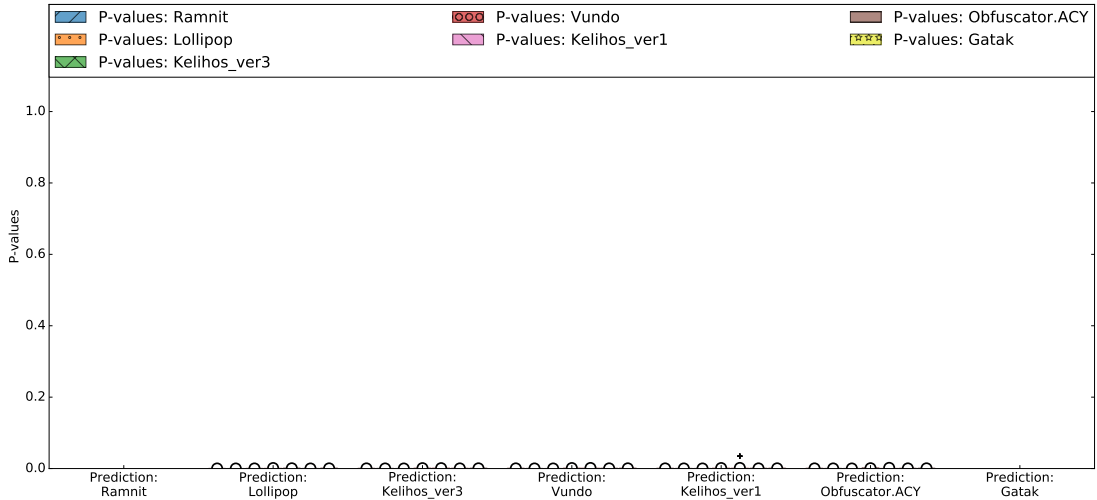
- ▶ Training families: Ramnit, Lollipop, Kelihos_ver3, Vundo, Obfuscator.ACY, Gatak, Kelihos_ver1
- ▶ Testing family: Tracur

Classification results:

| Lollipop | Kelihos_ver3 | Vundo | Kelihos_ver1 | Obfuscator.ACY |
|----------|--------------|-------|--------------|----------------|
| 5 | 6 | 358 | 140 | 242 |

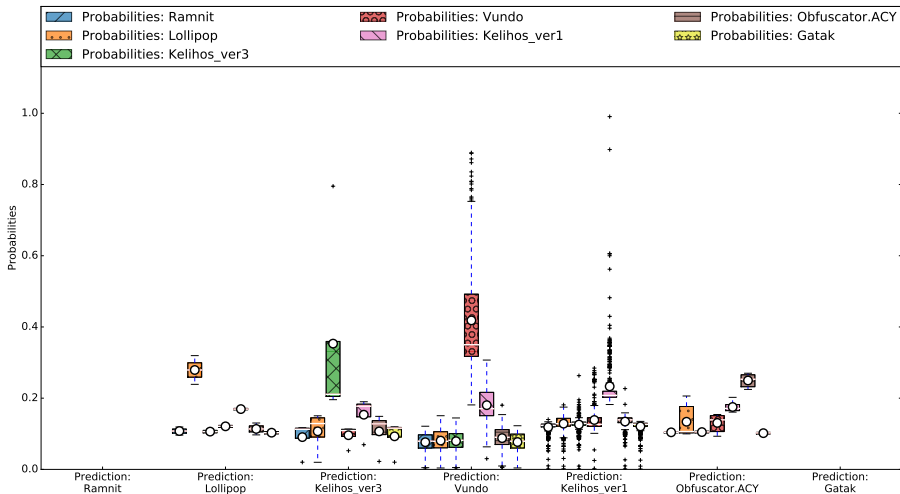
EXPERIMENTAL RESULTS: MULTICLASS CLASSIFICATION (NEW FAMILY)

P-value distribution for samples of Tracur family; as expected, the values are all close to zero.



EXPERIMENTAL RESULTS: MULTICLASS CLASSIFICATION (NEW FAMILY)

Probability distribution for samples of Tracur family; bounded to sum to one, the values are different than zero.



Conformal Evaluator (CE)

Statistical evaluation to assess predictions of ML classifiers and identify concept drift

Conformal Evaluator (CE)

Statistical evaluation to assess predictions of ML classifiers and identify concept drift

Algorithm Agnostic: Uses non-conformity measure (NCM) from the ML classifier
Statistical Support: Builds p-values from NCM to statistically-support predictions
Quality Thresholds: Builds thresholds from p-values to identify unreliable predictions

Conformal Evaluator (CE)

Statistical evaluation to assess predictions of ML classifiers and identify concept drift

Algorithm Agnostic: Uses non-conformity measure (NCM) from the ML classifier
Statistical Support: Builds p-values from NCM to statistically-support predictions
Quality Thresholds: Builds thresholds from p-values to identify unreliable predictions

- ▶ We evaluate the proposed solution on different ML classifiers and case studies
 - Android malware apps in binary classification settings
 - Windows PE binaries in multi-class classification settings
- ▶ Information on CE's python code and dataset availability at:

<https://s2lab.isg.rhul.ac.uk/projects/ce>

CONCLUSIONS

- ▶ CopperDroid: automatic reconstruction of apps behaviors¹⁷
 - System calls to abstract OS- and Android-specific behaviors
 - Resilient to changes to the runtime and Android versions

¹⁷<http://s2lab.isg.rhul.ac.uk/papers/files/ndss2015.pdf>

¹⁸<http://s2lab.isg.rhul.ac.uk/papers/files/most2016.pdf>

¹⁹<http://s2lab.isg.rhul.ac.uk/papers/files/aisec2016.pdf> and
<http://s2lab.isg.rhul.ac.uk/papers/files/usenixsec2017.pdf>

CONCLUSIONS

- ▶ CopperDroid: automatic reconstruction of apps behaviors¹⁷
 - System calls to abstract OS- and Android-specific behaviors
 - Resilient to changes to the runtime and Android versions
- ▶ Classification with such semantics: "It... Could... Work!"¹⁸
 - Selective set-based classification (CE/CP)
 - (WIP: binary classification and different feature engineering)

¹⁷<http://s2lab.isg.rhul.ac.uk/papers/files/ndss2015.pdf>

¹⁸<http://s2lab.isg.rhul.ac.uk/papers/files/most2016.pdf>

¹⁹<http://s2lab.isg.rhul.ac.uk/papers/files/aisec2016.pdf> and
<http://s2lab.isg.rhul.ac.uk/papers/files/usenixsec2017.pdf>

CONCLUSIONS

- ▶ CopperDroid: automatic reconstruction of apps behaviors¹⁷
 - System calls to abstract OS- and Android-specific behaviors
 - Resilient to changes to the runtime and Android versions
- ▶ Classification with such semantics: "It... Could... Work!"¹⁸
 - Selective set-based classification (CE/CP)
 - (WIP: binary classification and different feature engineering)
- ▶ Statistical evaluation of ML seems promising¹⁹
 - Identify concept drift and when to trust a prediction
 - TPR from **37.5%** to **92.7%** in realistic settings
 - Identifies previously-unknown classes or malicious samples

¹⁷<http://s2lab.isg.rhul.ac.uk/papers/files/ndss2015.pdf>

¹⁸<http://s2lab.isg.rhul.ac.uk/papers/files/most2016.pdf>

¹⁹<http://s2lab.isg.rhul.ac.uk/papers/files/aisec2016.pdf> and
<http://s2lab.isg.rhul.ac.uk/papers/files/usenixsec2017.pdf>