Automated Synthesis of Semantic Malware Signatures using MaxSAT
Newly Found Malware can steal bank details on Android phones

by Ali Raza
9 months ago

Hundreds Of Operations Canceled After Malware Hacks Hospitals Systems

Mohit Kumar

Thursday, November 03, 2016

Malware Shuts Down Operations at Hospitals

Android Malware Used to Hack and Steal a Tesla Car

By Catalin Cimpanu

Statistics

https://www.symantec.com/content/dam/symantec/docs/reports/ISTR-21-2016-en.pdf
Statistics

37M
Total count of malware detected over 6 months

https://www.symantec.com/content/dam/symantec/docs/reports/ISTR-21-2016-en.pdf
Statistics

37M
Total count of malware detected over 6 months

295
# of Android malware families by 2016

Statistics

**37M**
Total count of malware detected over 6 months

**295**
# of Android malware families by 2016

Hundreds of signatures ≈ Millions of samples

Apposcopy Overview

Feng, et al. FSE’14
Apposcopy Overview

A high-level language for describing semantic properties of malware

Feng, et al. FSE’14
Apposcopy Overview

A high-level language for describing semantic properties of malware

A novel static analysis for deciding if an app matches the signature of a family

Feng, et al. FSE’14
Caveats
Caveats

Writing signatures is tedious
Caveats

Writing signatures is tedious

Vulnerable to semantic obfuscation
Goal

Signature DB

App

Apposcopy

[YES] [NO]
Goal

• Infer a signature from few samples of a malware family
Goal

- Infer a signature from few samples of a malware family
- Approximate matching algorithm that is resistant to semantic obfuscation
Our Signature in a Nutshell

Inter-Component Call Graph

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Inter-Component Call Graph

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Inter-Component Call Graph

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Inter-Component Call Graph

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Inter-Component Call Graph

DeviceID -> Internet

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Inter-Component Call Graph

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Inter-Component Call Graph

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Inter-Component Call Graph

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Inter-Component Call Graph

Solid line: control property
Dashed line: data property

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

GoldDream :- \textit{receiver}(r),
icc(SYSTEM, r, e, _), GDEvent(e),
service(s), \textit{icc}^*(r, s),
flow(s, DeviceId, s, Internet),
flow(s, SubscriberId, s, Internet).

**GoldDream Signature**

Solid line: control property
Dashed line: data property

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

**Component Predicate**

\[
\text{GDEvent(SMS RECEIVED).} \\
\text{GDEvent(NEW OUTGOING CALL).} \\
\text{GoldDream} \ :- \ \text{receiver(r),} \\
\text{icc(SYSTEM, r, e, _), GDEvent(e),} \\
\text{service(s),} \ \text{icc*(r, s),} \\
\text{flow(s, DeviceId, s, Internet),} \\
\text{flow(s, SubscriberId, s, Internet).}
\]

**GoldDream Signature**

**Inter-Component Call Graph**

Solid line: control property
Dashed line: data property

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

Activity1 → Service1 → Activity2

DeviceId -> Internet

Inter-Component Call Graph

Component Predicate
GDEvent(SMS RECEIVED).
GDEvent(NEW OUTGOING CALL).
GoldDream :- receiver(r),
icc(SYSTEM, r, e, _), GDEvent(e),
service(s), icc*(r, s),
flow(s, DeviceId, s, Internet),
flow(s, SubscriberId, s, Internet).

Control Predicate

GoldDream Signature

Solid line: control property
Dashed line: data property

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Our Signature in a Nutshell

GoldDream Signature

Component Predicate
GDEvent(SMS RECEIVED).
GDEvent(NEW OUTGOING CALL).
GoldDream :- receiver(r),
icc(SYSTEM, r, e, _), GDEvent(e),
service(s), icc*(r, s),
flow(s, DeviceId, s, Internet),
flow(s, SubscriberId, s, Internet).

Control Predicate

Flow Predicate

Inter-Component Call Graph

Solid line: control property
Dashed line: data property

Feng, et al. FSE’14
Feng, et al. OOPSLA’15
Signature Inference
Signature Inference

Given \( n \) malware samples from family \( F \), compute its signature \( S \).
Signature Inference

Given $n$ malware samples from family $F$, compute its signature $S$

Any signature that matches $n$ samples
Signature Inference

Given $n$ malware samples from family $F$, compute its signature $S$

Any signature that matches $n$ samples

Empty signature could also be a solution!
Insight
Given $n$ malware samples from family $F$, compute its signature $S$. 

Insight
Insight

Given $n$ malware samples from family $F$, compute its signature $S$.

- Our candidate $S$ should be
Insight

Given $n$ malware samples from family $F$, compute its signature $S$

• Our candidate $S$ should be
  • A common subgraph to minimize false negatives
Insight

Given $n$ malware samples from family $F$, compute its signature $S$

• Our candidate $S$ should be
  • A common subgraph to minimize false negatives
  • Maximally suspicious to minimize false positives
Insight

Given $n$ malware samples from family $F$, compute its signature $S$

• Our candidate $S$ should be
  • A common subgraph to minimize false negatives
  • Maximally suspicious to minimize false positives

Infer signatures by finding a Maximally Suspicious Common Subgraph of $n$ malware samples
Example
Example
Example
Example

Common subgraph
Example
Example

Common subgraph

Maximally suspicious
Example

Common subgraph

Maximally suspicious
How to infer the signature

Infer signatures by finding a Maximally Suspicious Common Subgraph of n malware samples
How to infer the signature

Infer signatures by finding a Maximally Suspicious Common Subgraph of n malware samples
How to infer the signature

Infer signatures by finding a Maximally Suspicious Common Subgraph of n malware samples
How to infer the signature

Infer signatures by finding a \textbf{Maximally Suspicious Common Subgraph} of $n$ malware samples

Signature Inference \quad MSCS \quad MaxSat
MaxSat in a nutshell
MaxSat in a nutshell

MaxSat: Given a UNSAT boolean formula in CNF, determine the maximum number of satisfied clauses

\[
( x_0 \lor x_1 ) \land ( \neg x_0 \lor x_1 ) \land ( x_0 \lor \neg x_1 ) \land ( \neg x_0 \lor \neg x_1 )
\]
MaxSat in a nutshell

**MaxSat**: Given a UNSAT boolean formula in CNF, determine the maximum number of satisfied clauses

\[(x_0 \lor x_1) \land (\neg x_0 \lor x_1) \land (x_0 \lor \neg x_1) \land (\neg x_0 \lor \neg x_1)\]

**Hard Clause**: has to be satisfied
MaxSat in a nutshell

MaxSat: Given a UNSAT boolean formula in CNF, determine the maximum number of satisfied clauses

\[(x_0 \lor x_1) \land (\neg x_0 \lor x_1) \land (x_0 \lor \neg x_1) \land (\neg x_0 \lor \neg x_1)\]

Hard Clause: has to be satisfied

Soft Clause: preferable to be satisfied but could be UNSAT. Each has different weight since some are more important than the others
MaxSat in a nutshell

MaxSat: Given a UNSAT boolean formula in CNF, determine the maximum number of satisfied clauses

\[(x_0 \lor x_1) \land (\neg x_0 \lor x_1) \land (x_0 \lor \neg x_1) \land (\neg x_0 \lor \neg x_1)\]

Hard Clause: has to be satisfied

Soft Clause: preferable to be satisfied but could be UNSAT. Each has different weight since some are more important than the others

Find an assignment s.t. the total weight of satisfied clauses is maximized

\[\{x_0 \mapsto 0, x_1 \mapsto 0\}\]
Synthesis using MaxSat
Synthesis using MaxSat

- **Hard Clause**: common subgraph (control-flow property)
Synthesis using MaxSat

- **Hard Clause**: common subgraph (control-flow property)
- **Soft Clause**: maximally suspiciousness (data-flow property)
Synthesis using MaxSat

- **Hard Clause**: common subgraph (control-flow property)
- **Soft Clause**: maximally suspiciousness (data-flow property)
- **Weight** for each clause
  - Inverse frequency from benign samples
  - Higher weight to features that are commonly found in malware

\[
O = \sum_{v,v' \in V} x_0(v, v') + \sum_{v,v' \in V} \sum_{d \in \mathcal{D}} w(v,v',d)y_0(v, v', d).
\]

\( O = \text{Hard} \quad \text{Soft} \)
Example, cont.

\[ O = \sum_{v,v' \in V} x_0(v, v') + \sum_{v,v' \in V} \sum_{d \in \mathcal{D}} w(v,v',d)y_0(v, v', d). \]
Example, cont.

\[ O = \sum_{v,v' \in V} x_0(v, v') + \sum_{v,v' \in V} \sum_{d \in D} w(v,v',d) y_0(v, v', d). \]
Example, cont.

\[ O = \sum_{v,v' \in V} x_0(v,v') + \sum_{v,v' \in V} \sum_{d \in D} w_{(v,v',d)} y_0(v,v',d). \]
Example, cont.

\[ O = \sum_{v,v' \in V} x_0(v,v') + \sum_{v,v' \in V} \sum_{d \in D} w(v,v',d) y_0(v,v',d). \]

**Control properties**

**Data properties**

\[ O_1 = 6 \]
Example, cont.

\[ \mathcal{O} = \sum_{v,v' \in V} x_0(v, v') + \sum_{v,v' \in V} \sum_{d \in \mathcal{D}} w(v,v',d) y_0(v, v', d). \]

**Control properties**

**Data properties**

\( \mathcal{O}_1 = 6 \)

\( \mathcal{O}_2 = 4 \)
Example, cont.

\[
O = \sum_{v,v' \in V} x_0(v, v') + \sum_{v,v' \in V} \sum_{d \in D} w(v,v',d)y_0(v, v', d).
\]

**Control properties**

**Data properties**

\[
O_1 = 6
\]

\[
O_2 = 4
\]

\[
O_3 = 3
\]
Example, cont.

\[
O = \sum_{v,v' \in V} x_0(v, v') + \sum_{v,v' \in V} \sum_{d \in D} w(v,v',d)y_0(v, v', d).
\]

Control properties

Data properties

\[O_1 = 6\]

\[O_2 = 4\]

\[O_3 = 3\]
Approximate matching

Now that we have the signature...
Approximate matching

Now that we have the signature...

Utilize existing signature inference algorithm to decide if a sample A belongs to a family F:
Approximate matching

Now that we have the signature...

Utilize existing signature inference algorithm to decide if a sample $A$ belongs to a family $F$:

$$\delta(A, F) = \frac{f(\text{INFER_SIGNATURE}(A, S_F))}{f(S_F)}$$

$f(S)$: Weighted sum of the number of nodes and edges in $S$
Example, cont.
Example, cont.
Example, cont.
Example, cont.
Example, cont.
Example, cont.

Resistant to semantic obfuscation!
Evaluation
Evaluation

• RQ1: How do the signatures synthesized by Astroid compare with manual version?
Evaluation

• RQ1: How do the signatures synthesized by Astroid compare with manual version?

• RQ2: How effective is Astroid at detecting zero-day malware?
Evaluation

• RQ1: How do the signatures synthesized by Astroid compare with manual version?

• RQ2: How effective is Astroid at detecting zero-day malware?

• RQ3: How does Astroid compare against state-of-the-art malware detectors?
Manual v.s. Automated
Manual v.s. Automated

Malware Families from Android Genome Benchmarks

Detection Rate (%)

Manual (Feng, et al. FSE’14)  Astroid
Manual v.s. Automated

Detection Rate (%)

Malware Families from Android Genome Benchmarks

Manual (Feng, et al. FSE’14)
Manual v.s. Automated

Malware Families from Android Genome Benchmarks

Manual (Feng, et al. FSE’14)  Astroid
Manual v.s. Automated Detection Rate (%)

Malware Families from Android Genome Benchmarks

Manual (Feng, et al. FSE’14)  Astroid

Detection Rate (%)
Manual v.s. Automated

Detection Rate (%)

Malware Families from Android Genome Benchmarks

- Manual (Feng, et al. FSE'14)
- Astroid

Manual: 90%, 94%
Astroid: 94%, 90%
Manual v.s. Automated

Outperform manual version!

Detection Rate (%)

Malware Families from Android Genome Benchmarks
Zero-day malware
Zero-day malware

- 160 malware samples from Symantec and McAfee of which we have no signature

- Astroid: 92%, MassVet (Security’15): 81%
Zero-day malware

• 160 malware samples from Symantec and McAfee of which we have no signature

• Astroid: 92%, MassVet (Security’15): 81%

• Identify 22 Google Play apps that can’t be detected by AV tools but are actually malicious after manual inspection
Zero-day malware

- 160 malware samples from Symantec and McAfee of which we have no signature

- Astroid: 92%, MassVet (Security’15): 81%

- Identify 22 Google Play apps that can’t be detected by AV tools but are actually malicious after manual inspection

*Our approximate matching is effective!*
Comparison with other tools

False positive rate: Drebin(NDSS’14): 1%, MassVet (Security’15): 175/503, Astroid: 0.04%
Comparison with other tools

False positive rate: Drebin (NDSS'14): 1%, MassVet (Security'15): 175/503, Astroid: 0.04%
Comparison with other tools

False positive rate: Drebin (NDSS ’14): 1%, MassVet (Security ’15): 175/503, Astroid: 0.04%
### Comparison with other tools

False positive rate:
- Drebin (NDSS’14): 1%,
- MassVet (Security’15): 175/503,
- Astroid: 0.04%

<table>
<thead>
<tr>
<th>Tool</th>
<th>Detection Rate</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drebin</td>
<td>0.83</td>
<td>89%</td>
</tr>
<tr>
<td>MassVet</td>
<td>0.858</td>
<td>84%</td>
</tr>
<tr>
<td>Astroid</td>
<td>0.885</td>
<td>0.04%</td>
</tr>
<tr>
<td></td>
<td>0.913</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.94</td>
<td></td>
</tr>
</tbody>
</table>
Comparison with other tools

False positive rate: Drebin (NDSS’14): 1%, MassVet (Security’15): 175/503, Astroid: 0.04%

Detection Rate

- Drebin: 89%
- MassVet: 84%
- Astroid: 94%
Comparison with other tools

False positive rate: Drebin (NDSS’14): 1%, MassVet (Security’15): 175/503, Astroid: 0.04%

**Astroid achieves high detection rate with low FP!**
Conclusion
Conclusion

• Automatically infer semantic malware signature from very few samples
Conclusion

• Automatically infer semantic malware signature from very few samples

• Our approximate matching is resilient to semantic obfuscations
Thank you!

Automated Synthesis of Semantic Malware Signatures using Maximum Satisfiability.  
