Harnessing Intelligence from Malware Repositories

Arun Lakhotia and Vivek Notani
Software Research Lab
University of Louisiana at Lafayette
arun@louisiana.edu, vxn4849@louisiana.edu
Self Introduction

☐ Software Research Lab
  □ 10 years research on Malware
  □ Graduate course on malware analysis
  □ Active interaction with industry
  □ Funded by AFOSR, ARO, DARPA, ONR, and State of Louisiana

☐ Research Focus
  □ How does malware evade detection?
  □ How to detect stealthy malware?
  □ Malware analysis in the large

☐ Results
  □ Papers: 50+ peer-reviewed
  □ Patents: one granted
  □ Degrees: 6 Ph.D., 8 M.Sc.
  □ Research Funding: $5MM+
Targeted Attacks

Average Number of Spear-Phishing Attacks Per Day

Source: Symantec :: MARCH 2014 — FEBRUARY 2015
Machine Generation of Malware
CyberSecurity Paradox

Physical world: FAILED attempts are INVESTIGATED

Cyber World: FAILED attempts are CELEBRATED
Extract Intelligence from Malware

- Connect actors
- Discover trends
- Connect families
- Malware evolution

Person, Group, or Campaign
- Inferred Relationship
- Observed Relationship
On October 14, 2011, a threat that Symantec dubbed the "Duqu" [dyū-kō] because it was an anomaly, was one of several powerful computer worms that had provided us with samples recovered from computers in the Middle East, Europe, and the United States. Duqu appears to have been created as a follow-on to Stuxnet, which we recently described.

Duqu is essentially the precursor to Stuxnet. Some researchers believe that the threat was written by the same authors (or using Stuxnet source code) but with a completely different purpose.

Parts of Duqu are nearly identical to Stuxnet, but with a completely different purpose.
Requirement: “Google” for Malware
Challenges

1. Remove obfuscation
2. Map to document
3. Create index
4. Search
Key Innovation: VM Introspection in the Cloud
**Key Innovation: Semantic Fingerprints**

**STATE OF PRACTICE**

- **VIRUSBATTLE**
  - Bit shreds
  - Semantics
  - Fingerprint

- **Susceptible to obfuscation**

---

(C) 2015 U. Louisiana at Lafayette
Semantics Enabled: Connecting Malware through Code

Semantically similar binaries between malware families

Aldibot

Ponyloader

Smokeloader

Darkcomet
Unpacking Malware
Challenge 1: Packing
Classes of Packers (Protectors)

• Classification parameter
  • Based on execution behavior
    • When and how much of the original code is decrypted

• Traditional Packer
  • Entire original code is decrypted at one time
    • Entire original code is in clear text before it is executed

• Paged Packer
  • Just-In-Time decryption of a page when it is executed
    • Only a ‘page’ of the original code is in clear text at any time

• Virtual Machine Protectors
  • Decrypt a single instruction at a time
    • None of the original code is ever in clear text
Innovation: Unpacking using VM Introspection

Observe malware below ring 0
Unpacking Traditionally Packed Malware

1. Execute Malware
2. Detect When to Stop
3. Extract Revealed Code
When to Stop: Hump and Dump

- Traditional Packer
  - Decryption in a loop
  - High instruction execution frequency
  - Spike in frequency graph
- Hump & Dump Algorithm
  - Detect spike – hump
  - Detect end – flat

Addresses Ordered by last execution time
When to Stop: TimeOut

• What if Hump is never detected?
  • TimeOut
  • Limits execution time
Constructing PE

- Modify OEP using last PC value
- Fix Section Headers
- Copy Memory Contents to new PE
Extracting Memory Contents: Challenge

• Extracting memory through hypervisor
• Memory contents may be paged out by GuestOS

Solution:
• Determine memory is paged out
• Analyze execution profile
• Re-run unpacker with new parameters
  • Catch before memory is paged out
Case Study

Dataset Description:
- File Type: PE-32
- Source: FBI
- Availability: Upon request
- Collection period: 1 year

<table>
<thead>
<tr>
<th>Bot Family</th>
<th># Executables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aldibot</td>
<td>19</td>
</tr>
<tr>
<td>Armageddon</td>
<td>1</td>
</tr>
<tr>
<td>Blackenergy</td>
<td>65</td>
</tr>
<tr>
<td>Darkcomet</td>
<td>339</td>
</tr>
<tr>
<td>Darkshell</td>
<td>379</td>
</tr>
<tr>
<td>Ddoser</td>
<td>5</td>
</tr>
<tr>
<td>Illusion</td>
<td>17</td>
</tr>
<tr>
<td>Nitol</td>
<td>11</td>
</tr>
<tr>
<td>Optima</td>
<td>160</td>
</tr>
<tr>
<td>Ponyloader</td>
<td>1,312</td>
</tr>
<tr>
<td>Smokeloader</td>
<td>31</td>
</tr>
<tr>
<td>Umbraloader</td>
<td>25</td>
</tr>
<tr>
<td>Yzf</td>
<td>4</td>
</tr>
<tr>
<td>Zeus</td>
<td>41</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2,409</strong></td>
</tr>
</tbody>
</table>

(C) 2015 U. Louisiana at Lafayette
Case Study: Results

- Input : 2,409
- Unpacked : 2,354
- Output : 2,185

<table>
<thead>
<tr>
<th>HEURISTIC</th>
<th>Original</th>
<th>Unique Unpacked</th>
<th>Poor Unpacking</th>
<th>Poor Unpacking (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hump and Dump</td>
<td>1,671</td>
<td>1,523</td>
<td>205</td>
<td>12.27</td>
</tr>
<tr>
<td>TimeOut</td>
<td>515</td>
<td>500</td>
<td>46</td>
<td>8.93</td>
</tr>
<tr>
<td>Self-tuning</td>
<td>168</td>
<td>163</td>
<td>23</td>
<td>13.69</td>
</tr>
<tr>
<td>TOTAL</td>
<td>2,354</td>
<td>2,186</td>
<td>274</td>
<td>11.64</td>
</tr>
</tbody>
</table>

Unpacked Binary “very similar” to Original => Poor Unpacking
Unpacker’s Impact: Analysis Cost Reduction

Multiple malware produce the same unpacked result

**Malware Binaries**

<table>
<thead>
<tr>
<th></th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original binaries</td>
<td>2,409</td>
<td></td>
</tr>
<tr>
<td>Unpacked successfully</td>
<td>2,354</td>
<td>97.7%</td>
</tr>
<tr>
<td>Unique unpack result</td>
<td>2,185</td>
<td>92.8%</td>
</tr>
<tr>
<td>Reduction in analyst work</td>
<td>169</td>
<td>7.2%</td>
</tr>
</tbody>
</table>
Matching Code
Challenge 2: Code Obfuscation

push ecx
mov ecx,ebp
add ecx,33
mov [ecx-36],eax
pop ecx

push ecx
mov ecx,ebp
add ecx,33
push esi
mov esi,ecx
sub esi,34
mov [esi-2],eax
pop esi
pop ecx

push ecx
mov ecx,ebp
push eax
mov eax, 33
add ecx, eax
pop eax

push ecx
mov ecx, ebp
push eax
mov eax, 33
add ecx, eax
pop eax
mov eax, esi
push eax
mov esi, ecx
push edx
xor edx, 778f
mov edx, 34
sub esi, edx
pop edx
mov [esi - 2], eax
pop esi
pop ecx

push ecx
mov ecx, [ebp + 10]
mov ecx, ebp
push eax
add eax, 2342
mov eax, 33
add ecx, eax
pop eax
mov eax, esi
push eax
mov esi, ecx
push edx
xor edx, 778f
mov edx, 34
sub esi, edx
pop edx
mov [esi-2], eax
pop esi
pop ecx
Requirements

• Scale requirement
  • Search in collection of thousands to millions of malware

• Performance requirement
  • Provide results in seconds, or less

• Quality requirement
  • Error rates should be comparable to pairwise matching
Representations for Matching Binaries

Map binary to ‘document’

Disassemble

Abstracted Bytecode

Word = N-Bytes

(je push) (push mov) (mov pop) (pop xor)

(380091df)
(0091df96)
(91df96f6)
(df96f633)

(C) 2015 U. Louisiana at Lafayette
VirusBattle Strategy

Map binary to CFG to Document

Binary → Disassembly → CFG

Abstracted Bytecode
Abstracted Disassembly
Semantics
Juice

Word = Block
Code to Semantics

Code

• Sequential
• Focus on operations

push ebp
mov ebp,esp
sub esp,4
mov eax, DWORD ebp+4
mov DWORD ebp+8,eax
mov eax, DWORD ebp
mov DWORD ebp-4,eax

Semantics

• Parallel
• Captures affect

eax = def(ebp)
ebp = -4+def(esp)
esp = -8+def(esp)
memdw(-8+def(esp))= def(ebp)
memdw(-4+def(esp))= def(ebp)
memdw(4+def(esp)) = def(memdw(def(esp)))

Interpret: seq(Instruction) -> State -> State
State = LValue -> RValue
LValue = Register + Mem
RValue = Int

Value in previous state
Unsimplified

+ def(RValue)
+ RValue op RValue
+ op RValue
Limitations of (Block) Semantics

• Does not capture:
  • Register renaming
  • Memory address reassignment
  • Code motion between blocks
  • Evolutionary changes
    • Hashes good for strict equality

• Solution:
  • Generalize semantics
    • Juice
  • Use n-Block semantics
  • Use fuzzy hashes
Semantics to ‘words’

• Challenge:
  • How to map equal semantics to the same ‘word’?

• Solution:
  • Define canonical ordering
    • RValue structures are ground
    • Use ordering over symbols
    • Account for commutativity
    • Sum-of-product form
    • Simplify
  • Word = Hash (md5, SHA1) of linearized semantics

\[
\text{RValue} = \text{Int} + \text{def(RValue)} + \text{RValue op Rvalue} + \text{op RValue}
\]
Problem: Establish constraints between induced variables?

Solution

1. Track simplification steps

2. Generalize simplification steps
Semantics and Juice

**Code**

```
push ebp
mov ebp,esp
sub esp,4
mov eax, DWORD ebp+4
mov DWORD ebp+8, eax
mov eax, DWORD ebp
mov DWORD ebp-4, eax
```

**Semantics**

```
eax = def(ebp)
ebp = -4+def(esp)
esp = -8+def(esp)
memdw(-8+def(esp)) = def(ebp)
memdw(-4+def(esp)) = def(ebp)
memdw(4+def(esp)) = def(memdw(def(esp)))
```

**Juice**

- Inductive Generalization
  - Replace registers and constants by variables

```
RValue = Int
  + def(RValue)
  + RValue op RValue
  + op RValue
  + Variable
```

A = def(B),
B = N1+def(C),
C = N2+def(C),
memdw(N2+def(C)) = def(B)
memdw(N1+def(C)) = def(B)
memdw(N3+def(C)) = def(memdw(def(C)))

where A, B, C are ‘registers’
N1, N2, N3 are ‘Int’
Challenge 3: Scalable Search
Featurization Process

- Binary
- Unpack
- Disassembly
- Procedure
- Hash
- MinHash
- Procedure
- Hash
- MinHash
- Compiler Attributes
- Feature Vector
MinHash: A form of LSH

<table>
<thead>
<tr>
<th>A</th>
<th>Feature</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light Brown</td>
<td>Hair Color</td>
<td>Dark Brown</td>
</tr>
<tr>
<td>Long</td>
<td>Hair Length</td>
<td>Long</td>
</tr>
<tr>
<td>Brown</td>
<td>Eye Color</td>
<td>Brown</td>
</tr>
</tbody>
</table>

Feature = MinHash Function
Set of Features = MinHash Signature

Compose for Deterministic manipulations
Architecture
VirusBattle Webservice Architecture
Empirical Results
Dataset

Bots harvested in 2013

0 500 1000 1500 2000 2500 3000

unpacked
original

aldibot
armageddon
blackenergy
darkcomet
darkshell
ddoser
illusion
nitol
optima
ponyloader
smokeloader
umbroloader
vzf
zeus
“Interesting” Procedures
Libraries ID’ed by IDA
Transitive Library via Semantics

![Bar chart showing transitive library via semantics](chart.png)
RE Cost Reduction

1.7 Million+ procedures

# of procedures in binaries

105K+ procedures

# of IDA unique procedures

32K+ semantically unique procedures

<table>
<thead>
<tr>
<th>Procedures</th>
<th>All</th>
<th>IDA Unique</th>
<th>Juice Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lib Procs</td>
<td>65,113</td>
<td>11,482</td>
<td>4,382</td>
</tr>
<tr>
<td>Non Lib Procs</td>
<td>1,644,355</td>
<td>93,916</td>
<td>27,859</td>
</tr>
<tr>
<td>Total</td>
<td>1,709,468</td>
<td>105,398</td>
<td>32,241</td>
</tr>
</tbody>
</table>
Intelligence: Connecting Families

Semantically similar binaries between malware families

Aldibot
Smokeloader
Darkcomet
Ponyloader
Intelligence: Code Sharing

Non-Lib Unpacked Procedure

Optima

Shared in 13/159 samples

DarkComet

Shared in 65/319 samples
Intelligence: Code Evolution

ddoser: 10 samples

Percent binaries

# Procedures Shared

100 in 10%

200 in 100%
Intelligence: Needle in Haystack

- darkcomet: 649 samples
- 930 in 0-5%
- 20 in 60-65%
Performance

Distribution of analysis time

Legend
SRLUnpacker
Binjuice

95% percentile
Semantic analysis: < 15s
Unpacking: < 30s
Malware Search: < 7s
Procedure Search: < 100ms
VirusBattle: In a nutshell

Key Innovations

• Automated unpacking using VM introspection
• Semantic fingerprints, as against bits-based fingerprints
• Innovative 2-tier search algorithm for fast searches
• Search at various granularity:
  Whole binary, procedures, blocks, strings
• Interfaces with Palantir’s Forensic Investigation platform

Application
Rapidly extract intelligence from malware

• Order of magnitude improvement in malware analysts capability
  Unpacking time:
  Reduced from days/weeks to minutes
  Analysis work:
  Reduce efforts from weeks/months to minutes
  New capability:
  Build knowledge base of analysis indexed on similar code
  Share analysts’ experience across malware families

<table>
<thead>
<tr>
<th>Component</th>
<th>Time(*)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpacker</td>
<td>30 sec</td>
<td>97%</td>
</tr>
<tr>
<td>Semantic Juice</td>
<td>15 sec</td>
<td></td>
</tr>
<tr>
<td>Binary search</td>
<td>7 sec</td>
<td>95%</td>
</tr>
<tr>
<td>Procedure search</td>
<td>100ms</td>
<td></td>
</tr>
</tbody>
</table>

* Based on analysis of 2,500 botnets binaries; ** Max time to process 95% of files
Blackhat Sound Bytes

- Malware repositories are great source of intelligence
- Semantic juice peers through code obfuscation
- Semantic hashing enables fast search over large repositories
- VM Introspection gives you X-Ray vision over malware
- VirusBattle.com: Malware Intelligence Mining in the Cloud
Contact

Prof. Arun Lakhotia
Vivek Notani
University of Louisiana at Lafayette, Lafayette, LA, USA
arun@louisiana.edu
vxn4849@louisiana.edu
MinHash: A form of LSH

Consider Set A and Set B

Let $h(x) \rightarrow \text{int}$ be a function that takes a member of A or B and gives an integer.

Let $h_{\text{min}}(s)$ represent minimum member of set s w.r.t. h.

Then,

$$\Pr(h_{\text{min}}(A) = h_{\text{min}}(B)) = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Problem: High Variance!
MinHash Signatures:

• Compose d minhash functions:
  • Signature Match then implies each of the d functions agree on match
  • $\Pr(\text{sig}(A)=\text{sig}(B)) = J(A,B)^d$

Problem: Too many False Negatives!

• Check r minhash signatures:
  • A Match then implies atleast one of the r signatures agree on match
  • $\Pr(\text{match}(A,B)) = 1 - (1 - J(A,B)^d)^r$