The Importance of Interpretability in Cyber Security Analytics

Juston Moore
Advanced Research in Cyber Systems Group
October 4, 2017
Cyber research group at large DOE national research laboratory

- Identify gaps, emerging threats
- Data collection and sensors
- Modelling and simulation
- Tool/product development

Our Prospective:

- We assume an initial compromise will inevitably happen. Can we detect and mitigate attacks within the network perimeter effectively?
Data Sources for Cyber Research

• To motivate a larger research effort focused on operational cyber data, LANL have released three publicly available datasets:
  – 9 month time-series user/computer bipartite, 2014
  – 58 day comprehensive, 2015
  – 90 day Netflow and Window Event Logs, 2017

• [https://csr.lanl.gov/data/](https://csr.lanl.gov/data/)

• Caveat: Human subject research implications
2017 Dataset Overview

### Event category density

- Process end
- System events
- Interactive
- Kerberos
- Authentication other
- Network Logoff
- Network Logon
- Process start

### Event count

- log_hosts
- comp_accounts
- users
- source_hosts
- processes

Density

Count
Modeling challenges

- Where collected, not necessarily for cybersecurity purposes
  - Debugging
- Where collected, not intended for analytics
  - Primary purpose is to support human operations
- Not useful unless combined with other data sets
  - Collection is often incomplete
Modeling Periodic Activity in Network Data

Raw authentication events for one user

Changepoints for one user
Relational User Behavior Analytics

REDUCE: Statistically-Guided Reverse Engineering

MalNet: Deep Learning for Cross-Compiler Code Similarity Search
Relational User Behavior Analytics

REDUCE: Statistically-Guided Reverse Engineering

MalNet: Deep Learning for Cross-Compiler Code Similarity Search
LANL Cyber Data

• Data is collected for security purposes from all LANL Windows machines

• LANL requires methods to monitor the network for malicious intrusions

• Important events include authentications and process-running activities
Internal Red Team Identification
Compromised Account Detection
Behavior-Based User Profiling

- **User**
  - activity
  - preference
  - counts

- **Item**
  - popularity
  - attribute

Clusters:
- Cluster 1
- Cluster 2
- Cluster 3
- Stats Wiki
- Ext. Chat
- Email
- Video
- Chat

Los Alamos National Laboratory

Oct. 4, 2017
Red Team Identification

Peer-based anomaly detection via collaborative filtering

[Graphs showing the performance of different collaborative filtering methods based on user authentication and process activity percentiles.]
Relational User Behavior Analytics

REDUCE: Statistically-Guided Reverse Engineering

MalNet: Deep Learning for Cross-Compiler Code Similarity Search
Detection Eventually Fails for Malware

Large-Scale Processing & Detection

Endpoint Agents

Firewalls

AntiVirus

Machine Learning Detectors

Missed Samples

Manual Malware Analysis

Missed Sample 1

Missed Sample 2

Missed Sample 3
Tools for Malware Similarity Analysis

• Tools exist for:
  – Large-scale sample-level similarity search
    • Kam1n0, VXClass, multiple tools from DARPA Cyber Genome
  – Comparison between two samples
    • BinDiff, Diaphora
  – Sharing information between malware analysts regarding one sample
    • CollabREate

• There is currently a lack of tools to identify shared code within a known malware family, and to share this information among a team
REDUCE System Overview

New Samples

Malware Samples

Clean Samples

YARA Signatures

REDUCE Static Code Analysis

Code Similarity Search

User Interface

Statistically Guided Signature Generation

Visual Similarity Exploration

Collaborative Code Commenting

REDUCE
Yara Signatures

• Efficient signatures for large-scale deployment
  – Malware repository support: VirusTotal, LANL’s CodeVision, Sandia’s FARM
  – Network-based defenses: bro, PaloAlto, …
  – Host-based defenses: CarbonBlack, Tanium, …
  – Translation for MIR, Encase, and other IOC formats

• Especially effective for in-memory search

rule MiniASP
{
  meta:
    author = "AlienVault Labs"
    info = "CommentCrew-threat-apt1"

  strings:
    SKEY = \{ 71 30 6E 63 39 77 38 65 64 61 6F 69 75 6B 32 6D 7A 72 66 79 33 78 74 31 70 35 6C 73 36 37 67 34 62 76 68 6A \} 
    $PDB = "MiniAsp.pdb" nocase wide ascii

  condition:
    any of them
}
Yara Signature Compilation

strings:
REDUCE Increases Situational Awareness

REDUCE developed at Los Alamos
Finding Unique APT Code

• REDUCE takes a simple approach and looks for function-level similarity
  – Remove memory references, registers
  – Match assembly opcodes
  – Identify unique functions using Mutual Information

• This basic technique works surprisingly well for a large range of APTs, but it is easily defeated by simple code modifications
  – We’re now deploying Locality Sensitive Hash random projections to identify similar functions with a few assembly-level modifications

• We’ll look at obfuscating compilers in just a few seconds
# Interactive Signature Generation

<table>
<thead>
<tr>
<th>Score</th>
<th>Type</th>
<th>Length</th>
<th>Value</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>function</td>
<td>14</td>
<td></td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>100.0</td>
<td>string</td>
<td>76</td>
<td>Content-Type: multi...</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>100.0</td>
<td>string</td>
<td>56</td>
<td>Content-Disposition:...</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>100.0</td>
<td>string</td>
<td>41</td>
<td>Content-Disposition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>string</td>
<td>32</td>
<td>--MULTI-PARTS--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>string</td>
<td>24</td>
<td>IPHONE8.5(host:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>string</td>
<td>12</td>
<td>%s\setup.exe</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Prioritized list of indicators**

Signature effectiveness is displayed in realtime.
Seeing Selected Features in Context
Collaborative Code Commenting
We are actively seeking partners to pilot REDUCE!
Relational User Behavior Analytics

REDUCE: Statistically-Guided Reverse Engineering

MalNet: Deep Learning for Cross-Compiler Code Similarity Search
APT6 Compiler Change

mov dl, [eax-1] mov dl, [eax-1] movzx ecx, byte ptr [eax-1] movzx ecx, byte ptr [eax-1]
shl dl, 2      shl dl, 2      sar dl, 4      sar dl, 4
sar bl, 4      sar cl, 4      and dl, 3      and dl, 3
and bl, 3      and cl, 3      add cl, cl
add dl, bl      add dl, cl      add cl, cl
mov [esi-1], dl mov [esi-1], dl mov [esi-1], dl mov [esi-1], dl
mov dl, [eax+1] mov dl, [eax+1] movzx edx, byte ptr [eax+1] movzx edx, byte ptr [eax+1]
mov bl, [eax]  mov cl, [eax]  movzx ecx, byte ptr [eax]  movzx ecx, byte ptr [eax]
sar dl, 2      sar dl, 2      sar dl, 2
and dl, 0Fh     and dl, 0Fh     and dl, 0Fh
shl bl, 4      shl cl, 4      shl cl, 4
xor dl, bl      xor dl, cl      xor dl, cl
mov [esi], dl   mov [esi], dl   mov [esi], dl   mov [esi], dl
...            ...            ...            ...
• Goal: A code index will allow analysts to find similarities and differences across vast malware collections at the source code level

• We learn function embeddings using recurrent neural networks, which are highly effective for human language translation

• We apply a bidirectional recurrent network to machine code found in compiled binaries
Model

• We train our embeddings to capture similarity between malware functions using a Siamese network design

• Left and right networks are identical
Instruction Substitution is a serious pain for distance metrics that count the usage of different assembly instructions!

- \( a = b + c \)
  - Ex: \( r = \text{rand}(); \ a = b + r; \ a = a + c; \ a = a - r \)
- \( a = b \& c \)
  - \((b \ ^ \ ^ \ ~c) \ & \ b\)
- \( a = b \mid c \)
  - \((b \ & \ c) \ | \ (b \ ^ \ ^ \ c)\)
- \( a = b ^ c \)
  - \((\sim b \ & \ c) \ | \ (b \ & \ \sim c)\)
• Control Flow Flattening

entry:
%retnval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%a = alloca i32, align 4
store i32 0, i32* %retnval
store i32 %argc, i32* %argc.addr, align 4
store i8** %argv, i8*** %argv.addr, align 8
%0 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8** %0, i64 1
%1 = load i8** %arrayidx, align 8
%call = call i32 @atoi(i8* %1)
store i32 %call, i32* %a, align 4
%2 = load i32* %a, align 4
%cmp = icmp eq i32 %2, 0
br i1 %cmp, label %if.then, label %if.else

T  F

if.then:
store i32 1, i32* %retnval
br label %return

if.else:
store i32 10, i32* %retnval
br label %return

return:
%3 = load i32* %retnval
ret i32 %3

CFG for 'main' function

loopEntry:
%switchVar = load i32* %switchVar
switch i32 %switchVar, label %switchDefault | %switch1 | %switch2, label %return

switchDefault:
br label %loopEnd

first:
%reload = load volatile i32* %reg2mem
%cmp = icmp eq i32 %reload, 0
%3 = select i1 %cmp, i32 1, i32 2
store i32 %3, i32* %switchVar
br label %loopEnd

if.then:
store i32 1, i32* %retnval
store i32 2, i32* %switchVar
br label %loopEnd

if.else:
store i32 10, i32* %retnval
store i32 2, i32* %switchVar
br label %loopEnd

return:
%4 = load i32* %retnval
ret i32 %4

CFG for 'main' function
Obfuscator-LLVM

• Bogus Control Flow

entry:
%retval = alloca i32, align 4
%argc.addr = alloca i32, align 4
%argv.addr = alloca i8**, align 8
%a = alloca i32, align 4
store i32 0, i32* %retval
store i32 %argc, i32* %argc.addr, align 4
store i8** %argv, i8*** %argv.addr, align 8
%0 = load i8*** %argv.addr, align 8
%arrayidx = getelementptr inbounds i8** %0, i64 1
%1 = load i8** %arrayidx, align 8
%call = call i32 @atoi(i8* %1)
store i32 %call, i32* %a, align 4
%2 = load i32* %a, align 4
%cmp = icmp eq i32 %2, 0
br i1 %cmp, label %if.then, label %if.else

T       F

if.then:
store i32 1, i32* %retval
br label %return

if.else:
store i32 10, i32* %retval
br label %return

return:
%3 = load i32* %retval
ret i32 %3

CFG for 'main' function
Experimental Dataset

• We compiled several open-source software libraries with various compilers, optimizations, and code obfuscation techniques.

<table>
<thead>
<tr>
<th>Library</th>
<th>Functions</th>
<th>Mean Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2-1.0.6</td>
<td>40</td>
<td>16,761</td>
</tr>
<tr>
<td>coreutils-8.25</td>
<td>1208</td>
<td>257,185</td>
</tr>
<tr>
<td>curl-7.50.3</td>
<td>1012</td>
<td>229,979</td>
</tr>
<tr>
<td>rsync-3.1.2</td>
<td>450</td>
<td>106,631</td>
</tr>
<tr>
<td>tiff-3.8.2</td>
<td>253</td>
<td>76,067</td>
</tr>
<tr>
<td>zlib-1.2.8</td>
<td>129</td>
<td>32,505</td>
</tr>
<tr>
<td>Total</td>
<td>3,092</td>
<td>719,128</td>
</tr>
</tbody>
</table>
Performance: Accuracy

- We significantly outperformed simple distance-based similarity search for identifying similar source codes.
**Challenge Problem: the movfuscator!**

- mov’s are Turing-complete. So let’s compile EVERYTHING to mov’s!

<table>
<thead>
<tr>
<th>GCC</th>
<th>M/o/Vfuscator</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;is_prime&gt;:</td>
<td></td>
</tr>
<tr>
<td>push ebp</td>
<td></td>
</tr>
<tr>
<td>mov ebp,esp</td>
<td></td>
</tr>
<tr>
<td>sub esp,0x10</td>
<td></td>
</tr>
<tr>
<td>cmp DWORD PTR [ebp+0x8].0x1</td>
<td></td>
</tr>
<tr>
<td>jne 884d6499 &lt;is_prime+0x13&gt;</td>
<td></td>
</tr>
<tr>
<td>mov eax,0x0</td>
<td></td>
</tr>
<tr>
<td>jmp 884d64c0 &lt;is_prime+0x52&gt;</td>
<td></td>
</tr>
<tr>
<td>cmp DWORD PTR [ebp+0x8].0x2</td>
<td></td>
</tr>
<tr>
<td>jne 884d6499 &lt;is_prime+0x28&gt;</td>
<td></td>
</tr>
<tr>
<td>mov eax,0x1</td>
<td></td>
</tr>
<tr>
<td>jmp 884d64c0 &lt;is_prime+0x52&gt;</td>
<td></td>
</tr>
<tr>
<td>mov DWORD PTR [ebp-0x4].0x2</td>
<td></td>
</tr>
<tr>
<td>jmp 884d64be &lt;is_prime+0x41&gt;</td>
<td></td>
</tr>
<tr>
<td>mov eax,DWORD PTR [ebp+0x8]</td>
<td></td>
</tr>
<tr>
<td>cdq</td>
<td></td>
</tr>
<tr>
<td>idiv DWORD PTR [ebp-0x4]</td>
<td></td>
</tr>
<tr>
<td>mov eax,edx</td>
<td></td>
</tr>
<tr>
<td>test eax,ecx</td>
<td></td>
</tr>
<tr>
<td>jne 884d64ba &lt;is_prime+0x3d&gt;</td>
<td></td>
</tr>
<tr>
<td>mov eax,0x0</td>
<td></td>
</tr>
<tr>
<td>jmp 884d64c0 &lt;is_prime+0x52&gt;</td>
<td></td>
</tr>
<tr>
<td>add DWORD PTR [ebp-0x4].0x1</td>
<td></td>
</tr>
<tr>
<td>mov eax,DWORD PTR [ebp-0x4]</td>
<td></td>
</tr>
<tr>
<td>imul eax,DWORD PTR [ebp-0x4]</td>
<td></td>
</tr>
<tr>
<td>cmp eax,DWORD PTR [ebp+0x8]</td>
<td></td>
</tr>
<tr>
<td>jle 884d64cb &lt;is_prime+0x29&gt;</td>
<td></td>
</tr>
<tr>
<td>mov eax,0x1</td>
<td></td>
</tr>
<tr>
<td>leave</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

- This certainly breaks existing instruction-based similarity search
Challenge Problem: the movfuscator!

• mov’s are Turing-complete. So let’s compile EVERYTHING to mov’s!

<table>
<thead>
<tr>
<th>GCC</th>
<th>M/o/Vfuscator</th>
</tr>
</thead>
<tbody>
<tr>
<td>![GCC Diagram]</td>
<td></td>
</tr>
<tr>
<td>![M/o/Vfuscator Diagram]</td>
<td></td>
</tr>
</tbody>
</table>

• Conditional execution also breaks current static control flow analysis
Thank You!

Cyber Physical

Data Analytics

Network Security

A-4 Hierarchy of Security

Protect
- Infrastructure Modification
- Signature Deployment

Analyze
- Human Analyst
- Statistical Exploration

Detect
- Rule-Based Systems
- Machine Learning

Monitor
- Network
- Host

COMPUTE

PHYSICAL WORLD

SENT

PHYSICAL WORLD

ACT

DIGITAL WORLD

INPUT

DIGITAL WORLD

OUTPUT

Juston Moore <jmoore01@lanl.gov>