Statistical Similarity of Binaries

Yaniv David, Nimrod Partush, Eran Yahav @

*I*The research leading to these results has received funding from the European Union's - Seventh Framework Programme (FP7) under grant agreement n° 615688 – ERC- COG-PRIME.*
Motivation

Network time protocol (ntpd)

<table>
<thead>
<tr>
<th>Version</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntp-4.2.8</td>
<td>Dec 2014</td>
</tr>
<tr>
<td>ntp-4.2.6</td>
<td>Dec 2009</td>
</tr>
<tr>
<td>ntp-4.2.4</td>
<td>Dec 2006</td>
</tr>
<tr>
<td>ntp-4.2.2</td>
<td>Jun 2006</td>
</tr>
<tr>
<td>ntp-4.2.0</td>
<td>Oct 2003</td>
</tr>
<tr>
<td>ntp-4.1.2</td>
<td>Jul 2003</td>
</tr>
<tr>
<td>ntp-4.1.1</td>
<td>Feb 2002</td>
</tr>
<tr>
<td>ntp-4.1.0</td>
<td>Aug 2001</td>
</tr>
<tr>
<td>ntp-4.0.99</td>
<td>Jan 2000</td>
</tr>
<tr>
<td>ntp-4.0.90</td>
<td>Nov 1998</td>
</tr>
<tr>
<td>ntp-4.0.73</td>
<td>Jun 1998</td>
</tr>
<tr>
<td>ntp-4.0.72</td>
<td>Feb 1998</td>
</tr>
<tr>
<td>ntp-4.0</td>
<td>Sep 1997</td>
</tr>
<tr>
<td>xntp-3.86.5</td>
<td>Oct 1996</td>
</tr>
<tr>
<td>xntp3.5f</td>
<td>Apr 1996</td>
</tr>
<tr>
<td>xntp3.3wy</td>
<td>Jun 1994</td>
</tr>
<tr>
<td>xntp3</td>
<td>Jun 1993</td>
</tr>
<tr>
<td>xntp2</td>
<td>Nov 1989</td>
</tr>
</tbody>
</table>

RedHat’s Linux distribution

Apple’s OSX

CISCO’s 5900X (switches)

(table source: https://queue.acm.org/detail.cfm?id=2717320)
Semantic Similarity Wish List

• Given q (query) and set T (targets) rank targets based on similarity to q
• **Precise** - avoid false positives
• **Flexible** – find similarities across
  • Different compiler versions
  • Different compiler vendors
  • Different versions of the same code
• Work on **stripped** binaries
Challenge: Finding Similar Procedures

```
shr   eax, 8
lea   r14d, [r12+13h]
mov   r13, rbx
lea   rcx, [r13+3]
mov   [r13+1], al
mov   [r13+2], r12b
mov   rdi, rcx

mov   r9, 13h
mov   r12, rbx
add   rbp, 3
mov   rsi, rbp
lea   rdi, [r12+3]
mov   [r12+2], bl
lea   r13d, [rcx+r9]
shr   eax, 8
```

🔥 Heartbleed, gcc v.4.9  🔥 Heartbleed, clang v.3.5
Similarity by Composition - Irani et al. [2006]

- *image1* is similar to a *image2* if you can compose *image1* from the segments of *image2*

- Segments can be transformed
  - rotated, scaled, moved

- Segments of (statistical) significance, give more evidence
  - black background should be much less accounted for
Similarity of Binaries: 3 Step Recipe

1. Decomposition

2. Pairwise Semantic Similarity

3. Statistical Similarity Evidence
Similarity of Binaries: 3 Step Recipe

1. Decomposition

2. Pairwise Semantic Similarity

3. Statistical Similarity Evidence
Step 1 - Procedure Decomposition

We need to decompose procedures into comparable units

```
shr eax, 8
lea r14d, [r12+13h]
mov r13, rbx
lea rcx, [r13+3]
mov [r13+1], al
mov [r13+2], r12b
mov rdi, rcx
```
Step 1 - Procedure Decomposition

```
shr   eax, 8
lea   r14d, [r12+13h]
mov   r13, rbx
lea   rcx, [r13+3]
mov   [r13+1], al
mov   [r13+2], r12b
mov   rdi, rcx
```
Step 1 - Procedure Decomposition

```
shr eax, 8
lea r14d, [r12+13h]
mov r13, rbx
lea rcx, [r13+3]
mov [r13+1], al
mov [r13+2], r12b
mov rdi, rcx
```
Step 1 - Procedure Decomposition

```
shr    eax, 8  
lea    r14d, [r12+13h]  
mov    r13, rbx  
lea    rcx, [r13+3]  
mov    [r13+1], al  
mov    [r13+2], r12b  
mov    rdi ← rcx
```
Step 1 - Procedure Decomposition

shr    eax, 8
lea    r14d, [r12+13h]
mov    r13, rbx
lea    rcx, [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
Step 1 - Procedure Decomposition

```
shr eax, 8
lea r14d, [r12+13h]
mov r13, rbx
lea rcx, [r13+3]
mov [r13+1], al
mov [r13+2], r12b
mov rdi, rcx
```
Step 1 - Procedure Decomposition

```
shr    eax, 8
lea    r14d, [r12+13h]
mov    r13, rbx
lea    rcx, [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
```
Step 1 - Procedure Decomposition

```
shr   eax, 8
lea   r14d, [r12+13h]
mov   r13 ← rbx
lea   rcx, [r13+3]
mov   [r13+1], al
mov   [r13+2], r12b
mov   rdi, rcx
```
Step 1 - Procedure Decomposition

shr    eax,  8
lea    r14d,  [r12+13h]
mov    r13,  rbx
lea    rcx,  [r13+3]
mov    [r13+1], al
mov    [r13+2], r12b
mov    rdi, rcx
Step 1 - Procedure Decomposition

```
shr     eax, 8
lea     r14d, [r12+13h]
mov     r13, rbx
lea     rcx, [r13+3]
mov     [r13+1], al
mov     [r13+2], r12b
mov     rdi, rcx
```
Step 1 - Procedure Decomposition

**Inputs:** rbx

```
mov r13, rbx
lea rcx, [r13+3]
mov rdi, rcx
```

**Vars:** rdi, rcx, r13
Step 1 - Procedure Decomposition

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>shr eax, 8</code></td>
</tr>
<tr>
<td>2</td>
<td><code>lea r14d, [r12+13h]</code></td>
</tr>
<tr>
<td>3</td>
<td><code>mov r13, rbx</code></td>
</tr>
<tr>
<td>4</td>
<td><code>lea rcx, [r13+3]</code></td>
</tr>
<tr>
<td>5</td>
<td><code>mov [r13+1], al</code></td>
</tr>
<tr>
<td>6</td>
<td><code>mov [r13+2], r12b</code></td>
</tr>
<tr>
<td>7</td>
<td><code>mov rdi, rcx</code></td>
</tr>
</tbody>
</table>
Step 1 - Procedure Decomposition

• Applying program slicing on the basic-block level until all variables are covered

```
1: shr eax, 8
2: lea r14d, [r12+13h]
3: mov r13, rbx
4: lea rcx, [r13+3]
5: mov [r13+1], al
6: mov [r13+2], r12b
7: mov rdi, rcx
```

• We call these basic-block slices *Strands*
Similarity of Binaries: 3 Step Recipe

1. Decomposition

2. Pairwise Semantic Similarity

3. Statistical Similarity Evidence

Heartbleed, gcc v.4.9 -03

Heartbleed, clang v.3.5 -03

CORPUS
Step 2 – Pairwise Semantic Similarity

Strand 3
@Heartbleed, gcc v.4.9-03

Strand 3
in Boogie representation

```plaintext
mov    r13, rbx
lea    rcx, [r13+3]
mov    rdi, rcx

BAP + Smack

v1    := rbx
r13   := v1
v2    := r13 + 3
v3    := int_to_ptr(v2)
rcx   := v3
v4    := rcx
rdi   := v4
```
Step 2 – Pairwise Semantic Similarity

Heartbleed, gcc v.4.9 -03
Strand 6

v1 := r12
v2 := 13h + v1
v3 := int_to_ptr(v2)
r14 := v3
v4 := 18h
rsi := v4
v5 := v4 + v3
rax := v5

≈

Heartbleed, clang v.3.5 -03
Strand 11

v1 := 13h
r9 := v1
v2 := rbx
v3 := v2 + v3
v4 := int_to_ptr(v3)
r13 := v4
v5 := v1 + 5
rsi := v5
v6 := v5 + v4
rax = v6
Step 2 – Pairwise Semantic Similarity

Heartbleed, gcc v.4.9
Strand 6

Heartbleed, clang v.3.5
Strand 11

v1 := r12
v2 := 13h + v1
v3 := int_to_ptr(v2)
r14 := v3
v4 := 18h
rsi := v4
v5 := v4 + v3
rax := v5

v1 := 13h
r9 := v1
v2 := rbx
v3 := v2 + v3
v4 := int_to_ptr(v3)
r13 := v4
v5 := v1 + 5
rsi := v5
v6 := v5 + v4
rax := v6
Step 2 – Pairwise Semantic Similarity

Strand \( s_q \in q \)

Inputs: \( r12_q \)

\[
\begin{align*}
v1_q & := r12_q \\
v2_q & := 13h + v1_q \\
v3_q & := \text{int}_\text{to}_\text{ptr}(v2_q) \\
r14_q & := v3_q \\
v4_q & := 18h \\
r_{si_q} & := v4_q \\
v5_q & := v4_q + v3_q \\
r_{ax_q} & := v5_q
\end{align*}
\]

Variables: \( v1_q, v2_q, v3_q, r14_q, v4_q, r_{si_q}, v5_q, r_{ax_q} \)

Strand \( s_t \in t \in T \)

Inputs: \( rbx_t \)

\[
\begin{align*}
v1_t & := 13h \\
r9_t & := v1_t \\
v2_t & := rbx_t \\
v3_t & := v2_t + v3_t \\
v4_t & := \text{int}_\text{to}_\text{ptr}(v3_t) \\
r13_t & := v4_t \\
v5_t & := v1_t + 5 \\
r_{si_t} & := v5_t \\
v6_t & := v5_t + v4_t \\
r_{ax_t} & := v6
\end{align*}
\]

Variables: \( v1_t, r9_t, v2_t, v3_t, v4_t, r13_t, v5_t, r_{si_t}, v6_t, r_{ax_t} \)
Step 2 – Pairwise Semantic Similarity

Assume:
\[ r_{12q} = rb_{x_t} \]
\[ s_q; s_t; \]

Assert:
\[ v_{1q} = v_{1t} \]
... 
\[ rax_q = rax_{t} \]

Max number of equal variables
Step 2 – Pairwise Semantic Similarity

assume $r_{12_q} == rbx_t$

\[
\begin{align*}
v_{1_q} & := r_{12_q} \\
v_{2_q} & := 13h + v_{1_q} \\
v_{3_q} & := \text{int}_\text{to}_\text{ptr}(v_{2_q}) \\
r_{14_q} & := v_{3_q} \\
v_{4_q} & := 18h \\
r_{si_q} & := v_{4_q} \\
v_{5_q} & := v_{4_q} + v_{3_q} \\
r_{ax_q} & := v_{5_q}
\end{align*}
\]

\[
\begin{align*}
v_{1_t} & := 13h \\
r_{9_t} & := v_{1_t} \\
v_{2_t} & := rbx_t \\
v_{3_t} & := v_{2_t} + v_{3_t} \\
v_{4_t} & := \text{int}_\text{to}_\text{ptr}(v_{3_t}) \\
r_{13_t} & := v_{4_t} \\
v_{5_t} & := v_{1_t} + 5 \\
r_{si_t} & := v_{5_t} \\
v_{6_t} & := v_{5_t} + v_{4_t} \\
r_{ax_t} & := v_{6_t}
\end{align*}
\]

assert

$v_{1_q}=v_{2_t} \land v_{2_q}=v_{3_t} \land v_{3_q}=v_{4_t} \land r_{14_q}=r_{13_t} \\
v_{4_q}=v_{5_t} \land r_{si_q}=r_{si_t} \land v_{5_q}=v_{6_t} \land r_{ax_q}=r_{ax_t}$
Step 2 - Quantify Semantic Similarity

- $VCP(s_q, s_t) = \frac{\text{MaxEqualVars}(s_q, s_t)}{|s_q|}$
  - Variable Containment Proportion
  - An asymmetric relation
  - Using dataflow information and optimizations make this calculation feasible
Step 2 – Pairwise Semantic Similarity

assume \( r_{12} \equiv rbx_t \)

\[
\begin{align*}
  v_{1q} & = r_{12} \\
v_{2q} & = 13h + v_{1q} \\
v_{3q} & = \text{int}_\to\text{ptr}(v_{2q}) \\
r_{14} & = v_{3q} \\
v_{4q} & = 18h \\
r_{si} & = v_{4q} \\
v_{5q} & = v_{4q} + v_{3q} \\
r_{ax} & = v_{5q}
\end{align*}
\]

\[
\begin{align*}
  v_{1t} & = 13h \\
r_{9} & = v_{1t} \\
v_{2t} & = rbx_t \\
v_{3t} & = v_{2t} + v_{3t} \\
v_{4t} & = \text{int}_\to\text{ptr}(v_{3t}) \\
r_{13} & = v_{4t} \\
v_{5t} & = v_{1t} + 5 \\
r_{si} & = v_{5t} \\
v_{6t} & = v_{5t} + v_{4t} \\
r_{ax} & = v_{6t}
\end{align*}
\]

\( VCP(s_q; s_t) = 8/8 \)

assert
\[
\begin{align*}
  v_{1q} &= v_{2t},& v_{2q} &= v_{3t},& v_{3q} &= v_{4t},& r_{14} &= r_{13} \\
v_{4q} &= v_{5t},& r_{si} &= r_{si_t},& v_{5q} &= v_{6t},& r_{ax} &= r_{ax_t}
\end{align*}
\]
Step 2 – Pairwise Semantic Similarity

\[
\text{assume } r_{12q} = rb_{xt}
\]

\[
\begin{align*}
v_{1q} & = r_{12q} \\
v_{2q} & = 13h + v_{1q} \\
v_{3q} & = \text{int\_to\_ptr}(v_{2q}) \\
r_{14q} & = v_{3q} \\
v_{4q} & = 18h \\
r_{siq} & = v_{4q} \\
v_{5q} & = v_{4q} + v_{3q} \\
r_{axq} & = v_{5q}
\end{align*}
\]

\[
\begin{align*}
v_{1t} & = 13h \\
r_{9t} & = v_{1t} \\
v_{2t} & = rb_{xt} \\
v_{3t} & = v_{2t} + v_{3t} \\
v_{4t} & = \text{int\_to\_ptr}(v_{3t}) \\
r_{13t} & = v_{4t} \\
v_{5t} & = v_{1t} + 5 \\
r_{siti} & = v_{5t} \\
v_{6t} & = v_{5t} + v_{4t} \\
r_{axt} & = v_{6t}
\end{align*}
\]

\[
\text{VCP}(s_q; s_t) = 8/8
\]

\[
\text{VCP}(s_t; s_q) = 8/10
\]

\[
\begin{align*}
\text{assert} \\
v_{1q} & = v_{2t}, v_{2q} = v_{3t}, v_{3q} = v_{4t}, r_{14q} = r_{13t} \\
v_{4q} & = v_{5t}, r_{siq} = r_{siti}, v_{5q} = v_{6t}, r_{axq} = r_{axt}
\end{align*}
\]
Similarity of Binaries: 3 Step Recipe

1. Decomposition

2. Pairwise Semantic Similarity

3. Statistical Similarity Evidence

Heartbleed, gcc v.4.9 -03

Heartbleed, clang v.3.5 -03

CORPUS
Step 3 – Statistical Evidence

- We need to turn VCP into a probability that $s_q$ is input-output equivalent to $s_t$

$Pr(s_q|s_t) = \text{sigmoid} \left( VCP(s_q, s_t) \right) = \frac{1}{1 + e^{-k(VCP(s_q, s_t) - 0.5)}}$

"Throw" bad results close to 0

"Throw" good results close to 1
Step 3 – Statistical Evidence

- We need to know how significant is $s_q$
- To do that we use all the comparison data available

\[ \Pr(s_q | H_0) = \sum_{s_t \in T} \frac{\Pr(s_q | s_t)}{|T|} \]
Step 3 – Statistical Evidence

- Define a *Local Evidence Score* to quantify the statistical significance of matching each strand

\[
LES(s_q|t) = \log \frac{\max_{s_t \in t} \Pr(s_q|s_t)}{\Pr(s_q|H_0)}
\]
Step 3 – Statistical Evidence

\[
\max_{s_t \in t} \frac{\Pr(s_q | s_t)}{\Pr(s_q | H_0)} = \frac{1}{0.08} = 12.5
\]

\[
\max_{s_t \in t} \frac{\Pr(s_q | s_t)}{\Pr(s_q | H_0)} = \frac{1}{0.001} = 1000
\]
Step 3 - Global Similarity

• Procedures are similar if one can be composed using non-trivial, significantly similar parts of the other

\[ GES(q|t) = \sum_{s_q \in q} LES(s_q|t) \]
Similarity of Binaries: Recap

1. Decomposition

2. Pairwise Semantic Similarity

3. Statistical Similarity Evidence
Evaluation - Vulnerabilities

● Corpus
  ● Real-world code packages
    ● open-ssl, bash, qemu, wget, ws-snmp, ffmpeg, coreutils
  ● Spanning across product versions
    ● e.g. openssl-1.0.1{e,f,g}
  ● Compiled with clang 3.{4,5}, gcc 4.{6,8,9} and icc {14,15}
  ● 1500 procedures picked at random

● Queries
  ● Focused on vulnerabilities (for motivation’s sake)
Results - *Finding Heartbleed*

- OpenSSL 1.0.1f, compiled using clang 3.5
- Heartbleed procedure
- Full 1500 corpus

Compiler version (top), and vendor (bottom)
Results - *Finding Heartbleed*

Full GES Scores for Experiment #1
Results - Vulnerabilities

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>False Positives</th>
<th>False positives rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Heartbleed</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 Shellshock</td>
<td>3</td>
<td>0.002</td>
</tr>
<tr>
<td>3 Venom</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4 Clobberin' Time</td>
<td>19</td>
<td>0.0126</td>
</tr>
<tr>
<td>5 Shellshock #2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6 ws-snmp</td>
<td>1</td>
<td>0.0006</td>
</tr>
<tr>
<td>7 wget</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8 ffmpeg</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Low FP rate
  - Crucial to the vulnerability search scenario
- Previous methods fail at cross-{version,compiler} scenario or produce too many FPs (see paper)
Evaluation – All vs All

- Verified with randomly picked procedures
  - For example – when ff_rev34_decode@ffmpeg-2.4.6 is selected

```
<table>
<thead>
<tr>
<th></th>
<th>clang</th>
<th>gcc</th>
<th>icc</th>
</tr>
</thead>
<tbody>
<tr>
<td>clang</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icc</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Evaluation – All vs All

- Verified with randomly picked procedures
  - For example – when `ff_rev34_decode@ffmpeg-2.4.6` is selected

<table>
<thead>
<tr>
<th></th>
<th>clang</th>
<th>gcc</th>
<th>icc</th>
</tr>
</thead>
<tbody>
<tr>
<td>clang</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>gcc</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>icc</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Results – All vs All

All v. All comparison

- parse_integer()
- dev_ino_compare()
- default_format()
- print_stat()
- cached_umask()
- create_hard_link()
- i_write()
- compare_nodes()
- ff_rv34_decode_init_thread_copy()
- wget-1.8:
- ftp_syst()
www.binsim.com
(code+demo)
Summary

• Clear motivation
  • Finding vulnerable code, detecting clones, etc.

• Challenging scenario
  • Finding similarity cross-{compiler, version} in stripped binaries

• Applied to real-world code

• Take home:
  • A semantic approach, yet feasible
  • Accuracy achieved with statistical framework