Similarity of Binaries through re-Optimization

By
Yaniv David, Nimrod Partush & Eran Yahav
Technion, Israel
Motivation

Developer

OpenSSL

httpd

APACHE®
Motivation

Security Researcher

OpenSSL 💔
Problem Definition

\[ |T| \geq 10^6 \]
Challenge

OpenSSL's
dtls1_buffer_message()

ARM
mov x0, x20
mov x20, 3
add x0, x0, 1
sub x21, x21, x0
cmn x21, 2

intel
lea r15, [rax+1]
sub r13, r15
xor rax, rax
add rax, 3
cmp r13, -2

gcc 4.8
-00

icc 15.0.3
-03

OpenSSL's dtls1_buffer_message()
Our Approach

For finding Similarity of Binaries
Our Approach: **What**

Query $q$: dtls1_buffer_message()

Compiler: icc 15.0.3 –O3

Architecture: Intel

```
push r12
push rbx
push rbp
...
```

```
lea r15, [rax+1]
sub r13, r15
xor rax, rax
add rax, 3
cmp r13, -2
...
```

```
mov x0, x20
mov x20, 3
add x0, x0, 1
sub x21, x21, x0
cmn x21, 2
...
```

```
leal r1, [rax+1]
sub r13, r15
xor rax, rax
add rax, 3
cmp r13, -2
...
```

```
...  
```
Our Approach: **What**

**Query** $q$: dtls1_buffer_message()
**Compiler:** icc 15.0.3 –O3
**Architecture:** intel

**Query** $t_1$: dtls1_buffer_message()
**Compiler:** gcc 4.8 –O0
**Architecture:** ARM
Our Approach: What

Procedure $t_2$: unrelated(dtls1_buffer_message())

Compiler: icc 15.0.3 -O3

Architecture: intel

```
push r12
push rbx
push rbp
...
lea r15, [rax+1]
sub r13, r15
xor rax, rax
add rax, 3
cmp r13, -2
...
```
Our Approach: How

- **Decompose** procedure to fragments
- **Transform** fragments to canonical form

```
  mov  x0,  x20
  add  x0,  x0,  1
  sub  x21, x21, x0
  cmn  x21,  2

  tmp0 = register0 + 1
  register1 = tmp0
  tmp1 = register2 - tmp0
  register2 = tmp1
  tmp2 = tmp1 - 2
```

```
  lea  r15, [rax+1]
  sub  r13, r15
  cmp  r13, -2

  tmp0 = register0 + 1
  register1 = tmp0
  tmp1 = register2 - tmp0
  register2 = tmp1
  tmp2 = tmp1 - 2
```

- Count shared fragments while weighing in their **statistical significance**
Decomposing Assembly Procedures

- The procedure is broken at **basic block** level
  - Block ordering is ignored

```
push r12
push rbx
push rbp
sub rsp, 10
...
mov edi, 10
...
lea r15, [rax+1]
sub r13, r15
...
```

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We use slicing to break basic blocks into separate data-independent computations:

```
mov x0, x20
mov x20, 3
add x0, x0, 1
sub x21, x21, x0
cmn x21, 2
```

```
mov x0, x20
add x0, x0, 1
sub x21, x21, x0
cmn x21, 2
mov x20, 3
```
Moving to Canonical Form (re-Optimizing)

• “Out-of-context” re-optimization

\[
\begin{align*}
\text{mov} & \quad x0, x20 \\
\text{add} & \quad x0, x0, 1 \\
\text{sub} & \quad x21, x21, x0 \\
\text{cmn} & \quad x21, 2
\end{align*}
\]

\[
\begin{align*}
t0 & = \text{load} x20 \\
t1 & = \text{load} x0 \\
t2 & = 1 \\
t3 & = \text{add} t1, t2 \\
t0 & = \text{load} x20 \\
\end{align*}
\]

\[
\begin{align*}
t1 & = \text{add} t0, 1 \\
\text{store} & \quad t1, r1 \\
\end{align*}
\]

\[
\begin{align*}
t & = \text{store} t3, x0 \\
\end{align*}
\]
dtls1_buffer_message()

\[
\begin{align*}
\text{push} & \quad r12 \\
\text{push} & \quad rbx \\
\text{push} & \quad rbp \\
\text{sub} & \quad rsp, 10 \\
\ldots & \\
\text{mov} & \quad edi, 10 \\
\ldots & \\
\text{lea} & \quad r15, [rax+1] \\
\text{sub} & \quad r13, r15 \\
\ldots & \\
\ldots & 
\end{align*}
\]

\[
R(\text{dtls1_buffer_message}) = \{
\begin{align*}
t0 & = \text{load} \ r0 \\
t1 & = \text{add} \ t0, 1 \\
\text{store} & \ t1, r1 \\
t2 & = \text{load} \ r2 \\
t3 & = \text{sub} \ t2, t1 \\
\text{store} & \ t3, r2 \\
\ldots & \\
t0 & = \text{load} \ r0 \\
t1 & = \text{sub} \ t0, 10 \\
t2 & = \text{load} \ r1 \\
t3 & = \text{add} \ t2, 3 \\
t3 & = \text{mul} \ t3, t1 \\
\text{store} & \ t3, r2 \\
\ldots & 
\end{align*}
\}
Computing Similarity

\[ \text{Similarity}(q, t) = |R(q) \cap R(t)| \]

• Reminder:

```
unrelated() , icc 15.0.3, -O3
dtls1_buffer_message() , icc 15.0.3, -O3
```

```
push   r12  
push   rbx  
push   rbp  
sub    rsp, 10
```

```
push   r12  
push   rbx  
push   rbp  
sub    rsp, 10
```

=
Statistical Significance

- Given a canonical fragment $s$, we need to determine its significance.

$$Pr(s) = \frac{|\{ p \in P \mid s \in R(p) \}|}{|P|}$$

- We estimate $W$ with a bound, random sample of procedures $P$ – a “Global Context”
Computing Similarity

\[ \text{Similarity}(q, t) = \sum_{s \in R(q) \cap R(t)} \frac{1}{\text{Pr}_P(s)} \]

- A sum ranging over the \textit{shared canonical fragments}
- A distinctive fragment with \(\text{Pr}_P(s) = 0.001\) will contribute 1000
- A common fragment with \(\text{Pr}_P(s) = \frac{1}{5}\) will contribute 5
Evaluation

Prototype Implementation: GitZ
**GitZ Output**

**Procedure** \( q: \text{OpenSSL’s dtls1_buffer_message()} \)

1. **Procedure** \( t_{42} \)
   - Similarity: 170.34

2. **Procedure** \( t_{13} \)
   - Similarity: 168.91

3. **Procedure** \( t_{900} \)
   - Similarity: 130.41

4. **Procedure** \( t_{218,777} \)
   - Similarity: 101.11

5. **Procedure** \( t_{43,081} \)
   - Similarity: 13.19

\[ \cdots \]
Evaluation Corpus

- Real-world code packages
  - OpenSSL, BASH, , , git, QEMU, wget, ffmpeg, Coreutils
  - Containing ~11,000 procedures

- Compiled to:
  - x86_64 with CLang 3.{4,5}, gcc 4.{6,8,9} and icc {14,15} x 9
  - ARM-64 with aarch64-gcc 4.8 and aarch64-Clang 4

- Optimization levels -0{0,1,2,3,s} x 5

- Corpus size: $|T| = 45 \times \sim 11,000 = \sim 500,000$

- Queries: 9 procedures from notable CVEs
GitZ Accuracy

Here, we report accuracy as the number of FPs ranked above the lowest TP.

1. **Procedure** $t_{42}$
   - Similarity: 170.34
   - **Positive**

2. **Procedure** $t_{13}$
   - Similarity: 168.91
   - **Positive**

3. **Procedure** $t_{900}$
   - Similarity: 130.41
   - **Negative**

4. **Procedure** $t_{218777}$
   - Similarity: 101.11
   - **Positive**

5. **Procedure** $t_{43081}$
   - Similarity: 13.19
   - **Negative**

...  

500,000. **Procedure** $t_{81}$
- Similarity: 0.0
- **Negative**

#FP = 1

$FPr = \frac{1}{500,000}$
## GitZ Scalability

- How useful is GitZ in the vulnerability search scenario?

<table>
<thead>
<tr>
<th>#</th>
<th>Alias</th>
<th>CVE</th>
<th>#FPs</th>
<th>FP Rate</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Heartbleed</td>
<td>2014-0160</td>
<td>52</td>
<td>0.000104</td>
<td>15m</td>
</tr>
<tr>
<td>2</td>
<td>Shellshock</td>
<td>2014-6271</td>
<td>0</td>
<td>0</td>
<td>17m</td>
</tr>
<tr>
<td>3</td>
<td>Venom</td>
<td>2015-3456</td>
<td>0</td>
<td>0</td>
<td>16m</td>
</tr>
<tr>
<td>4</td>
<td>Clobberin'</td>
<td>2014-9295</td>
<td>0</td>
<td>0</td>
<td>16m</td>
</tr>
<tr>
<td>5</td>
<td>Shellshock #2</td>
<td>2014-7169</td>
<td>0</td>
<td>0</td>
<td>12m</td>
</tr>
<tr>
<td>6</td>
<td>WS-snmp</td>
<td>2011-0444</td>
<td>0</td>
<td>0</td>
<td>14m</td>
</tr>
<tr>
<td>7</td>
<td>wget</td>
<td>2014-4877</td>
<td>0</td>
<td>0</td>
<td>10m</td>
</tr>
<tr>
<td>8</td>
<td>ffmpeg</td>
<td>2015-6826</td>
<td>0</td>
<td>0</td>
<td>17m</td>
</tr>
<tr>
<td>9</td>
<td>WS-statx</td>
<td>2014-8710</td>
<td>0</td>
<td>0</td>
<td>18m</td>
</tr>
</tbody>
</table>

0.12s for query-target pair, single core
Evaluating Solution Components

• How does each component of our solution affect the accuracy of GitZ?

<table>
<thead>
<tr>
<th>Query</th>
<th>Corpus Size</th>
<th>#Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heartbleed</td>
<td>10000</td>
<td>45</td>
</tr>
</tbody>
</table>
Evaluating the Global Context

Reminder: \[ \mathcal{Pr}_P(s) = \frac{|\left\{ p \in P \middle| s \in R(p) \right\}|}{|P|} \]
Evaluating Solution Vectors

False Positive Rate

<table>
<thead>
<tr>
<th>0.12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
</tr>
<tr>
<td>0.08</td>
</tr>
<tr>
<td>0.06</td>
</tr>
<tr>
<td>0.04</td>
</tr>
<tr>
<td>0.02</td>
</tr>
<tr>
<td>0.01</td>
</tr>
<tr>
<td>0.00</td>
</tr>
</tbody>
</table>
Evaluating Solution Vectors

False Positive Rate

Lifted (Only) LLVM Fragments
Evaluating Solution Vectors

False Positive Rate

0.12
0.10
0.08
0.06
0.04
0.02
0.00

LLVM
LLVM w/ Global Context

No Global Context
With Global Context
Evaluating Solution Vectors

**False Positive Rate**

- **LLVM**
- **LLVM w/ GC**
- **Normalized LLVM Fragments**

**No Global Context**

- 0.073
- 0.046
- 0.022

**With Global Context**

- 0.092
- 0.046
- 0.022
Evaluating Solution Vectors

False Positive Rate

- LLVM
- LLVM w/ GC
- Normalized LLVM
- Optimized LLVM
- Canonical Fragments

No Global Context
With Global Context
Take Aways

• A procedure can be identified using a set of statistically significant fragments
  • The statistical data can be collected over a relatively small set

• Applying an optimizer “out-of-context” is useful at transforming fragments to canonical form
  • A form that allows finding similarity
Questions

• **Canonical Form: The Good**

• **Canonical Form: The Bad**

• **Previous Work**

• **BinDiff**

• **More Experiments!**

• **You’re over-thinking this**

• **Out-of-Context?**

• **Limitations**

• **Evaluating Binary Classifiers**

• **All v. All**

• **Is Pr(s) a probability even?**

• **The Global Context**
Canonical Form: The Good

The Good:

```assembly
mov  rax, -2
sub  rbx, rax
```

Canonical

```assembly
t0 = load r0
t1 = add 2, t0
store t1, r0
```

The Bad:

```assembly
add  r12, 2
```

Canonical

```assembly
t0 = load r0
t1 = add 2, t0
store t1, r0
```

```assembly
add  rax, 1
add  rbx, 2
add  rbx, rax
```

Canonical

```assembly
t0 = load r0
t1 = load r1
t2 = add t0, t1
t3 = add t1, 3
store t3, r1
```

```assembly
add  rbx, 2
add  rax, 1
add  rax, rbx
```

Canonical

```assembly
t0 = load r0
t1 = add 2, t0
store t1, r0
```
Canonical Form: The Bad

```
add rax, 1
add rbx, 2
add rbx, rax
```

```
t0 = load r0
t1 = load r1
t2 = add t0, t1
t3 = add t1, 3
store t3, r1
```

```
t0 = load r0
t1 = load r1
t2 = add t0, t1
t3 = add t1, 3
store t3, r0
```

≠
### Previous Work

<table>
<thead>
<tr>
<th></th>
<th>Gitz-1500: Cross-{Comp, Arch, Opt}</th>
<th>Esh-1500: Cross-Comp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#FPs</td>
<td>CROC</td>
</tr>
<tr>
<td>Heartbleed</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Shellshock</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Venom</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Clobberin’ Time</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Shellshock #2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>WS-snmp</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>wget</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>WS-statx</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## BinDiff

<table>
<thead>
<tr>
<th>Alias</th>
<th>Matched?</th>
<th>Similarity</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heartbleed</td>
<td>✗</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Shellshock</td>
<td>✗</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Venom</td>
<td>✗</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Clobberin’ Time</td>
<td>✗</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Shellshock #2</td>
<td>✗</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ws-snmp</td>
<td>✓</td>
<td>0.89</td>
<td>0.91</td>
</tr>
<tr>
<td>wget</td>
<td>✗</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>✓</td>
<td>0.72</td>
<td>0.79</td>
</tr>
</tbody>
</table>
The Global Context

• Sampled over **canonical fragments**, from procedures of all archs\compilers\optimizations
• A new arch\compiler\optimization?
  • We are somewhat future proof due to optimization
    • Even if a compiler decides to do things a bit different, it should arrive at the same canonical form
  • Entirely new behaviors will require a (partial) resampling
    • For instance: using the stack in offsets of 13 O_0
All v. All
Limitations

```c
uerr_t
ftp_syst (int csock, ...){
    ...  
    /* Send SYST request. */
    request = ftp_request ("SYST", NULL);
    nwritten = fd_write (csock, request,
                         strlen (request), -1);

    if (nwritten < 0) {
        free (request);
        return WRITEFAILED;
    }
    free (request);
    ...}
```

```c
mov r14, rsi
mov r15, rdi
mov edi, esi
call esi, esi
mov rbp, rdi
mov edi, 0F93h
mov r12, rsi
mov esi, esi
call [ebp+0]
test edx, eax
mov [ebp+0]
```

```c
(b) CLang 3.5 -02
(c) CLang 3.4 -02
(d) gcc 4.6 -02
```
Evaluating Binary Classifiers

- The Receiver Operating Characteristic (ROC) is a widespread method for evaluating a binary classifier, by plotting the ratio of TPs to FPs, for all the different thresholds.
- The Area Under Curve is then summed and value between 0-1 is produced. Our results were > .96.
- ROC means “how well did we cover all true positives, before we encounter false positives”
- We used Concentrated ROC, an adaptation for huge corpora, which further “punishes” highly ranked FPs.
Jaccard Index?

• Major difference: does not take statistical significance into account, at all.

\[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} \]
Is \( \text{Pr}(s) \) a probability even?

\[
\text{Pr}(s) = \frac{|\{ p \in W | s \in R(p) \}|}{|W|}
\]

- \( \text{Pr}(s) \) is a probability over the sample space of \( W \)
- \( W \) is a “multiset” of all canonical fragments in existence
- \(|\{ p \in W | s \in R(p) \}|\) counts the occurrences of \( s \) in \( W \)
- \( \text{Pr}(s_1) + \text{Pr}(s_2) + \text{Pr}(s_3) + \text{Pr}(s_4) = \frac{3}{7} + \frac{2}{7} + \frac{1}{7} + \frac{1}{7} = 1 \)

- \( \text{Pr}(s) \) is an estimation of \( W \), which betters as \( P \) grows
- As we evaluated
You’re over-thinking this

• Why not just run the binary and get a version string??
  • Sometimes a lib is embedded
  • You can’t always easily run (different arch, dependencies, etc.)
  • Running can put you in unnecessary risk
  • Purely static
• We’ve seen cases where the version string is not maintained correctly!
Out-of-Context?

• In context: The slice is surrounded with:
  • Other instructions from the block
  • Other blocks
  • Other procedures

The optimizer here must account for the surroundings, and cannot easily “cut-through” unrelated operations

• Out-of-context: Just the slice. The optimizer can easily extract a concise canonical fragment, that can be matched with semantically equivalent fragments from other procedures.
More Experiments!

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Queries</th>
<th>Targets</th>
<th>FP Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-Optimization ARM-64</td>
<td>aarch64-gcc 4.8 -0*</td>
<td>aarch64-gcc 4.8 -0*</td>
<td>0</td>
</tr>
<tr>
<td>Cross-(Optimization V Version) x86_64</td>
<td>gcc 4.{6,8,9} -0*</td>
<td>gcc 4.{6,8,9} -0*</td>
<td>0.001</td>
</tr>
<tr>
<td>Cross-Compiler x86_64</td>
<td>Compilers\text{\textsubscript{x86}} -01</td>
<td>Compilers\text{\textsubscript{x86}} -01</td>
<td>0.002</td>
</tr>
</tbody>
</table>
## GitZ Accuracy:
Cross-Compiler/Optimization/Architecture

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Queries</th>
<th>Targets</th>
<th>FP Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-Architecture Low Optimization</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(Compilers\_x86 ∨ Compilers\_ARM) -01 | (Compilers\_x86 ∨ Compilers\_ARM) -01 | 0.006   |
| Cross-Architecture Standard Optimization | 
(Compilers\_x86 ∨ Compilers\_ARM) -02 | (Compilers\_x86 ∨ Compilers\_ARM) -02 | 0.005   |
| Cross-Architecture Heavy Optimization | 
(Compilers\_x86 ∨ Compilers\_ARM) -03 | (Compilers\_x86 ∨ Compilers\_ARM) -03 | 0.004   |