Tracelet-Based Code Search in Executables

Yaniv David & Eran Yahav
Technion, Israel
Finding vulnerable apps

We can find identical or patched code

```c
int foo() {
    ...
    // buffer
    // overflow
    ...
    printf(...)
    ...
}
```

```c
int alsoFoo() {
    ...
    // buffer
    // overflow
    ...
    printf(...)
    ...
}
```

```c
int patchedFoo() {
    ...
    // buffer
    // overflow
    ...
    if (...) {}
    printf(...)
    ...
}
```

Where else does this vulnerable function exist?
Finding vulnerable apps

We can find identical or patched code

int foo() {
  …
  // buffer overflow
  …
  printf(…)
  …
}

int alsoFoo() {
  …
  // buffer overflow
  …
  printf(…)
  …
}

int patchedFoo() {
  …
  if (…)
  {}
  printf(…)
  …
}

int alsoFoo() {
  …
  // buffer overflow
  …
  printf(…)
  …
}

What if we don’t have the source code?

Where else does this vulnerable function exist?
mov [esp+18h+var_18], offset aD1
mov ecx, 1
mov [esp+18h+var_14], ecx
call _printf
...

Search in Binaries

Function 1 - wc
Coreutils 6.12

Function 2 – diff
Coreutils 7.15
Search engine core

- Fast & Scalable
- Accurate (low false positives)
Challenge 1: similarity at the binary level

printf(...)@foo():

```c
int foo() {
    ...
    // buffer
    // overflow
    ...
    printf(...)
    ...
}
```

printf(...)@patchedFoo():

```c
int patchedFoo() {
    ...
    // buffer
    // overflow
    ...
    if (...) {}
    printf(...)
    ...
}
```
Challenge 1: similarity at the binary level

- Offsets in memory
- Register allocation
- New Instruction
Challenge2: similarity between different structures

foo’s CFG:

```assembly
loc_401358:
  mov [esp+18h+var_18], offset aD1
  mov ecx, 1
  mov [esp+18h+var_14], ecx
  call _printf
```

patchedFoo’s CFG:

```assembly
loc_401370:
  mov [esp+28h+var_28], offset aD1
  mov ebx, 1
  mov esi, 4
  mov [esp+28h+var_24], ebx
  call _printf
```
In this talk

• A system for searching code in executables
  – Based on tracelet decomposition of each function
  – Works by solving a set of alignment and dataflow constraints with minimal violations on tracelets

• An evaluation methodology based on tools from Information Retrieval
  – How do we know that our search engine is good?
Our Approach

Extract tracelets
Deal with structural changes

Pair tracelets using alignment and rewrite
Deal with the code changes

Similarity score
Using tracelets to deal with CFG structural changes

A tracelet is a fixed length sub-trace

For length=3,

In this example we get:

(A1,A2,A5)
(A1,A3,A5)
(A3,A4,A5)
Using tracelets calculate similarity between different structures

foo's CFG:

patchedFoo's CFG:

We need to find the corresponding tracelet
Comparing tracelets

foo’s tracelet

patchedFoo’s tracelet:

Graph -> linear code

Align & RW

Edit distance
Dealing with code changes: Align

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>mov</td>
</tr>
<tr>
<td>[esp+18h+var_18], offset aD1</td>
<td>[esp+28h+var_28], offset aD1</td>
</tr>
<tr>
<td>mov</td>
<td>mov</td>
</tr>
<tr>
<td>ecx, 1</td>
<td>ebx, 1</td>
</tr>
<tr>
<td>mov</td>
<td>mov</td>
</tr>
<tr>
<td>[esp+18h+var_14], ecx</td>
<td>[esp+28h+var_24], ebx</td>
</tr>
<tr>
<td>call</td>
<td>call</td>
</tr>
<tr>
<td>_printf</td>
<td>_printf</td>
</tr>
</tbody>
</table>

Align tracelets using specialized edit-distance

A1

1. mov [esp+18h+var_18], offset aD1
2. mov ecx, 1
3. mov [esp+18h+var_14], ecx
4. call _printf

B1

1. mov [esp+28h+var_28], offset aD1
2. mov ebx, 1
3. mov esi, 4
4. mov [esp+28h+var_24], ebx
5. call _printf

A5

B7

A2

B2
Dealing with code changes: DFA

Analyze data flow

Record live registers

A1

B1

A2

B2

A5

B7
# Dealing with code changes: Symbolize

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) mov</td>
<td>(1) mov</td>
</tr>
<tr>
<td>[esp+18h+var_18], offset ad1</td>
<td>[esp+28h+var_28], offset ad1</td>
</tr>
<tr>
<td>(2) mov</td>
<td>(2) mov</td>
</tr>
<tr>
<td>ecx, 1</td>
<td>ebx, 1</td>
</tr>
<tr>
<td>(3) mov</td>
<td>(3) mov</td>
</tr>
<tr>
<td>[esp+18h+var_14], ecx</td>
<td>[esp+28h+var_24], ebx</td>
</tr>
<tr>
<td>(4) call _printf</td>
<td>(4) call _printf</td>
</tr>
</tbody>
</table>

A5  | B7  |

move to symbolic names

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) mov</td>
<td>[r11+28h+m12], OF13</td>
</tr>
<tr>
<td>(2) mov</td>
<td>r21, 1</td>
</tr>
<tr>
<td>ecx, 1</td>
<td>esi, 4</td>
</tr>
<tr>
<td>(3) mov</td>
<td>[r31+28h+m31], r33</td>
</tr>
<tr>
<td>[esp+18h+var_14], ecx</td>
<td>FC41</td>
</tr>
<tr>
<td>(4) call _printf</td>
<td>(4) call FC41</td>
</tr>
</tbody>
</table>

A5  | B7  |
Dealing with code changes: Solve & Rewrite

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) mov [esp+18h+var_18], offset aD1</td>
<td></td>
</tr>
<tr>
<td>(2) mov ecx, [esp+18h+var_14]</td>
<td></td>
</tr>
<tr>
<td>(3) mov [esp+18h+var_14], ecx</td>
<td></td>
</tr>
<tr>
<td>(4) call _printf</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A5</th>
<th>B7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) mov [r11+28h+m12], OF13</td>
<td></td>
</tr>
<tr>
<td>(2) mov r21, 1</td>
<td></td>
</tr>
<tr>
<td>(X) mov esi, 4</td>
<td></td>
</tr>
<tr>
<td>(3) mov [r31+28h+m31], r33</td>
<td></td>
</tr>
<tr>
<td>(4) call _printf</td>
<td></td>
</tr>
</tbody>
</table>

Use alignment & DFA to create constraints

Data Flow constraints:
- r21=r33;
- r11=r31;

Alignment constraints:
- r11=esp; F13=...; m12=var_18;
- r21=ecx; e31=esp;
- m32=var_14; r33=ecx;
- FC41=_printf;
Dealing with code changes: Solve & Rewrite

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) mov [esp+18h+var_18], offset ad1</td>
<td>(1) mov [r11+28h+m12], OF13</td>
</tr>
<tr>
<td>(2) mov ecx, 1</td>
<td>(2) mov r21, 1</td>
</tr>
<tr>
<td>(3) mov [esp+18h+var_14], ecx</td>
<td>(x) mov esi, 4</td>
</tr>
<tr>
<td>(4) call _printf</td>
<td>(3) mov [esp+28h+var_14], ecx</td>
</tr>
<tr>
<td></td>
<td>(4) call _printf</td>
</tr>
</tbody>
</table>

Distance after rewrite = 1 instruction delete + 2 value changes
Our Approach

Extract tracelets
Deal with structural changes

Pair tracelets using alignment and rewrite
Deal with the code changes
From paired tracelets to function similarity score

\[ \frac{2 \times \#PairedTracelets(f_1, f_2)}{\#Tracelets(f_1) + \#Tracelets(f_2)} \]

\[ \frac{\#PairedTracelets(f_1, f_2)}{\text{Min}(\#Tracelets(f_1), \#Tracelets(f_2))} \]
Using tracelets calculate similarity between different structures

foo’s CFG:

```assembly
loc_401358:
mov [esp+18h+var_18], offset ad1
mov ecx, 1
mov [esp+18h+var_14], ecx
call _printf
```

patchedFoo’s CFG:

```assembly
mov [esp+28h+var_28], offset ad1
mov ebx, 1
mov esi, 4
mov [esp+28h+var_24], ebx
call _printf
```

(A1,A2,A5)~(B1,B2,B7),(A1,A3,A4)~(B1,B3,B4),
(A3,A4,A5)~(B3,B4,B7),(A1,A3,A5) -> “lost”
Using tracelets calculate similarity between different structures

(A1, A2, A5)~(B1, B2, B7), (A1, A3, A4)~(B1, B3, B4),
(A3, A4, A5)~(B3, B4, B7), (A1, A3, A5) \rightarrow \text{“lost”}

\[
\frac{2 \times 3}{4 + 6} = \frac{6}{10} = 60\% \text{ similarity (ratio)}
\]
Our system

Repository crawler

Index into DB

Functions DB (Mongodb)

Web front

Search engine core & CLI interface @ github

Crawling server

Similarity search results

Similarity search engine

<table>
<thead>
<tr>
<th>Function info</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:0x041...@tar_1_22.rpm">0x041...@tar_1_22.rpm</a></td>
<td>98%</td>
</tr>
<tr>
<td><a href="mailto:0x043...@tar_1_21.rpm">0x043...@tar_1_21.rpm</a></td>
<td>92%</td>
</tr>
<tr>
<td><a href="mailto:0x042...@cpio_2_10.rpm">0x042...@cpio_2_10.rpm</a></td>
<td>89%</td>
</tr>
<tr>
<td>Other functions</td>
<td>70%</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
Our system

<table>
<thead>
<tr>
<th>Function info</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:0x041...@tar_1_22.rpm">0x041...@tar_1_22.rpm</a></td>
<td>98%</td>
</tr>
<tr>
<td><a href="mailto:0x043...@tar_1_21.rpm">0x043...@tar_1_21.rpm</a></td>
<td>92%</td>
</tr>
<tr>
<td><a href="mailto:0x042...@cpio_2_10.rpm">0x042...@cpio_2_10.rpm</a></td>
<td>89%</td>
</tr>
<tr>
<td>Other functions</td>
<td>70%</td>
</tr>
</tbody>
</table>

Similarity search results

Repository crawler

Google crawler

Crawling server

Index into DB

Functions DB (Mongodb)

Web front

Web

Similarity search engine
One experiment – find my Heartbleed (CVE-2014-0160)

Tracelet-based Search engine

Mixed & stripped* Executables (1 Million functions)

Function info

<table>
<thead>
<tr>
<th>Function info</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>tls1_heartbeat @openssl_1_0_1f.rpm</td>
<td>98%</td>
</tr>
<tr>
<td>dtls1_process_heartbeat @openssl_1_0_1f.rpm</td>
<td>96%</td>
</tr>
<tr>
<td><a href="mailto:...@openssl_1_0_1e.rpm">...@openssl_1_0_1e.rpm</a></td>
<td>89%</td>
</tr>
<tr>
<td>more vulnerable functions ....</td>
<td>....</td>
</tr>
</tbody>
</table>

TLS implementation does not properly handle Heartbeat Extension packets causes information disclosure
## Using a single threshold

90% similarity score is... good?
Can we really choose one threshold?

<table>
<thead>
<tr>
<th>Function info</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>tls1_heartbeat @openssl_1_0_1f.rpm</td>
<td>98%</td>
</tr>
<tr>
<td>dtls1_process_heartbeat</td>
<td>96%</td>
</tr>
<tr>
<td>@openssl_1_0_1f.rpm</td>
<td>89%</td>
</tr>
<tr>
<td>other functions</td>
<td>....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function info</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:0x041...@tar_1_22.rpm">0x041...@tar_1_22.rpm</a></td>
<td>88%</td>
</tr>
<tr>
<td><a href="mailto:0x043...@tar_1_21.rpm">0x043...@tar_1_21.rpm</a></td>
<td>83%</td>
</tr>
<tr>
<td><a href="mailto:0x042...@cpio_2_10.rpm">0x042...@cpio_2_10.rpm</a></td>
<td>89%</td>
</tr>
<tr>
<td>Other functions</td>
<td>70%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function info</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:0x042...@wget_1_12.rpm">0x042...@wget_1_12.rpm</a></td>
<td>94%</td>
</tr>
<tr>
<td><a href="mailto:0x045...@wget_1_14.rpm">0x045...@wget_1_14.rpm</a></td>
<td>91%</td>
</tr>
<tr>
<td>Other functions</td>
<td>60%</td>
</tr>
</tbody>
</table>

90% similarity score is... good?
Can we really choose one threshold?
Using a single threshold

90% similarity score is...good?
Can we really choose one threshold?

<table>
<thead>
<tr>
<th>Function info</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>tls1_heartbeat @openssl_1_0_1f.rpm</td>
<td>98%</td>
</tr>
<tr>
<td>dtls1_process_heartbeat @openssl_1_0_1f.rpm</td>
<td>96%</td>
</tr>
<tr>
<td><a href="mailto:0x041...@tar_1_22.rpm">0x041...@tar_1_22.rpm</a></td>
<td>88%</td>
</tr>
<tr>
<td><a href="mailto:0x043...@tar_1_21.rpm">0x043...@tar_1_21.rpm</a></td>
<td>83%</td>
</tr>
<tr>
<td><a href="mailto:0x042...@wget_1_12.rpm">0x042...@wget_1_12.rpm</a></td>
<td>94%</td>
</tr>
<tr>
<td><a href="mailto:0x045...@wget_1_14.rpm">0x045...@wget_1_14.rpm</a></td>
<td>91%</td>
</tr>
<tr>
<td>other functions</td>
<td>....</td>
</tr>
<tr>
<td>Other functions</td>
<td>70%</td>
</tr>
</tbody>
</table>

There should be a more accurate way
ROC – trying all thresholds

• Receiver operating characteristic
• Try every threshold (=>binary classifier)
• Get a number representing the method’s accuracy
The function we are searching for

Threshold: XX%

Tracelet-based Search engine

Remove any functions below Threshold

Calculate Accuracy

Accuracy = (TP + TN) / (P + N)
ROC – trying all thresholds

• Method’s accuracy is Area Under Curve (AUC) determines precision
CROC is better than ROC

- The matches we expect are very sparse
- We need to “punish” false positives – they have a high cost
- CROC does exactly that
Experiment Structure

Manually Compiled (GNU ftp sources)

Random (Google crawler)

Linux Repositories (RpmFind.com crawler)

Context Group

Code Change group

Vulnerable Code
Experiment goal

Context group representative

Tracelet-based Search engine

Mixed & stripped Executables (1 Million functions)

Similar Functions

\(? \quad =\)

Context Group
Experiment Setup & Results

Tracelet-based Search engine

Mixed & stripped Executables (1 Million functions)

<table>
<thead>
<tr>
<th></th>
<th>N-grams Size 5, Delta 1</th>
<th>Graphlets K=5</th>
<th>Tracelets K=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUC[ROC]</td>
<td>72%</td>
<td>60%</td>
<td>99%</td>
</tr>
<tr>
<td>AUC[CROC]</td>
<td>25%</td>
<td>12%</td>
<td>99%</td>
</tr>
</tbody>
</table>
Conclusions

• Tracelets based code search system
  – Effective in finding exact and near matches
  – Provides a quantitative similarity score

• Evaluated using Information Retrieval tools
  – Achieves good precision and recall
  – Tested against other leading methods