Precise and Accurate Patch Presence Test for Binaries

Usenix Security’18

Hang Zhang, Zhiyun Qian
University of California, Riverside
What’s the problem?
What’s the problem?

Short Answer: Given an Android image (or other binary), how do we decide whether a CVE has been patched?
A real-world example
A real-world example
A real-world example
A real-world example

Open Source
A real-world example

**Open Source**
A real-world example

Open Source

Few source “snapshots” w/o commit history.
A real-world example

Are the mainstream Linux/AOSP patches propagated?

Few source “snapshots” w/o commit history.
Open vs. Closed

- Open-source is the trend.
- Code reuse in closed-source software.
Open vs. Closed

- Open-source is the trend.
- Code reuse in closed-source software.
- Is the open-source security patch applied in the binary?
Why challenging?
#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.
#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.
  - if (a > 0)
  + if (a >= 0)
#1: Needle in the (changing) haystack

- if (a > 0)
+ if (a >= 0)
  
- a = 0;
+ a = 0;

Security patch as a needle: small, subtle.
#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

  - if (a > 0)
  + if (a >= 0)

- Patched function as a changing haystack.
#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.
  
  ```java
  - if (a > 0) ...
  + if (a >= 0) a = 0;
  + ....
  
  - + the line
  + the line
  
  - + the line
  + the line
  + the line
  + the line
  + the line
  + the line
  + the line
  ```

- Patched function as a changing haystack.

```
Func():
.....
AAAAAA
+ the line
AAAAAA
.....
```
#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

  - if (a > 0) ...
  + if (a >= 0) ...
  + a = 0;

- Patched function as a changing haystack.

```c
Func():
......
BBBBBB
+ the line
BBBBBB
......
```

```c
Func():
......
AAAAAA
+ the line
AAAAAA
......
```
#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.

  ```
  - if (a > 0)
  + if (a >= 0)
  ```

- Patched function as a changing haystack.

  ```
  Func():
  ......       
  BBBBBBB
  + the line
  BBBBBBB    
  ......       
  ```

  ```
  Func():
  ......       
  AAAAAAA
  + the line
  AAAAAAA    
  ......       
  ```

  ```
  Func():
  ......       
  CCCCCCC
  + the line
  CCCCCCC    
  ......       
  ```
#1: Needle in the (changing) haystack

- Security patch as a needle: small, subtle.
  
  ```
  - if (a > 0)
  + if (a >= 0)
  ... 
  a = 0;
  ...
  ```

- Patched function as a changing haystack.

```
Func():
.......
BBBBBBB
+ the line
BBBBBBB
.......
```

```
 Func():
 .......
AAAAAAA
+ the line
AAAAAAA
.......
```
#2: Haystack is a binary...

- Find the needle in a binary.
Related work
Related work

Category 1: Source-source matching.
Related work

Category 1: Source-source matching.

Cannot deal with binary haystack.
Related work

Category 1: Source-source matching.

Cannot deal with binary haystack.

Category 2: Binary-binary matching.
Related work

Category 1: Source-source matching.
Cannot deal with binary haystack.

Category 2: Binary-binary matching.
Lack of knowledge about the needle (i.e. the patch).
How does FIBER work?

How does a human expert work?
Patch change site analysis. (Source level)

Binary signature translation.

Match in binary.
Given an open-source security patch, you need to locate it in a binary.

What will you do at first?
Given an open-source security patch, you need to locate it in a binary.

What will you do at first?

Pick those most **obvious**, **unique** and **representative** change sites.
Unique – Exists only in the patched version.
Unique – Exists only in the patched version.

Solution: token-based string search to test uniqueness, add contexts if not unique.
Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.
- **Stable** – Not affected by other irrelevant changes.
Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.
- **Stable** – Not affected by other irrelevant changes.
Stable – Not affected by other irrelevant changes.

Func():
......
BBBBBBB
+ the line
BBBBB
......

Func():
......
AAAAAAA
+ the line
AAAAAA
......

Func():
......
CCCCCCC
+ the line
CCCCCC
......

Func():
......
DDDDDDD
+ the line
DDDDDD
......
Change Site Analysis

- **Stable** – Not affected by other irrelevant changes.

**Solution**: keep the change site as small as possible (always start from a single line), add contexts only when necessary.
Easy-to-recognize – Imagine what a human prefers.
Change Site Analysis

- Easy-to-recognize – Imagine what a human prefers.
  
  + func_noInline()
Easy-to-recognize – Imagine what a human prefers.

Perfect: easily located by call instruction and function name (Android images have symbol table).
Easy-to-recognize – Imagine what a human prefers.

Perfect: easily located by call instruction and function name (Android images have symbol table).
Easy-to-recognize – Imagine what a human prefers.

**Perfect**: easily located by call instruction and function name (Android images have symbol table).

**Good**: both syntax structure and semantic change.
Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

  ```
  + func_noinline()
  + if (cond)
  + a = b * c
  ```

  *Perfect*: easily located by call instruction and function name (Android images have symbol table).

  *Good*: both syntax structure and semantic change.
Change Site Analysis

- **Easy-to-recognize** – Imagine what a human prefers.

  ```
  + func_noinline()
  + if (cond)
  + a = b * c
  ```

  **Perfect**: easily located by call instruction and function name (Android images have symbol table).

  **Good**: both syntax structure and semantic change.

  **Meh**: only semantic change without syntax change.
Easy-to-recognize – Imagine what a human prefers.

+ `func_noinline()`

 Perfect: easily located by call instruction and function name (Android images have symbol table).

+ `if (cond)`

 Good: both syntax structure and semantic change.

+ `a = b * c`

 Meh: only semantic change without syntax change.

Solution: we rank the change sites based on statement types involved, according to our domain knowledge.
Patch change site analysis. (Source level)

Binary signature translation.

Match in binary.
What if we do it manually?

- How to connect the source change with binary code?
What if we do it manually?

- How to connect the source change with binary code?

```c
if (a > 1)
    do A;
else
    do B;
```
What if we do it manually?

- How to connect the source change with binary code?

```plaintext
if (a > 1)  
do A;
else  
do B;
```

Syntax

... cond jmp

... ...

... ...
What if we do it manually?

- How to connect the source change with binary code?

```
if (a > 1)
    do A;
else
    do B;
```

**Syntax**

```
... cond jmp

...  ...
```

**Semantics**

```
... a > 1?

A  B
```
What if we do it manually?

- How to connect the source change with binary code?

```c
if (a > 1)
  do A;
else
  do B;
```

Correlate both its syntax and semantics to the binary code.
Binary Signature Translation

- Identify and organize correlated instructions in the reference binary.
Binary Signature Translation

- Identify and organize correlated instructions in the reference binary.

```c
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
}
```
Binary Signature Translation

- Identify and organize correlated instructions in the reference binary.

```c
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
}
```

**AArch64 calling convention:**
- `a` -> `X0`
- `b` -> `X1`
- `c` -> `X2`

**DWARF Debug Information**

- **foo:**
  - ...
  - MOV X3,X0
  - MOV X4,X1
  - ...

- **ADD X0,X3,X4**
- **CMP X0,X2**
- **BGT label1**

- **label1:**
  - **BL bar**

- **...**
Identify and organize correlated instructions in the reference binary.

```plaintext
foo(a,b,c){
  ...
  if (a+b > c)
  bar(a+b);
  ...
}
```

**AArch64 calling convention:**
- a -> X0
- b -> X1
- c -> X2

```
foo:
  ...
  MOV X3,X0
  MOV X4,X1
  ...
  ADD X0,X3,X4
  CMP X0,X2
  BGT label1
  ...
label1:
  BL bar
```
Find the “root” instructions.

ADD X0,X3,X4
CMP X0,X2
BGT label1

label1:
BL bar
Find the “root” instructions.

\[ X_0 = X_3 + X_4 \]

ADD X0,X3,X4
CMP X0,X2
BGT label1

label1:
BL bar
Binary Signature Translation

- Find the “root” instructions.

\[ X_0 = X_3 + X_4 \]

\[(X_3 + X_4) > X_2\]

```
ADD X0,X3,X4
CMP X0,X2
BGT label1
```

**label1:**

```
BL bar
```
Find the “root” instructions.

\[ X_0 = X_3 + X_4 \]

\[(X_3 + X_4) > X_2\]

ADD X0,X3,X4
CMP X0,X2
BGT label1

label1:
BL bar
Find the “root” instructions.

\[
X_0 = X_3 + X_4
\]

\[(X_3 + X_4) > X_2\]

ADD X0,X3,X4
CMP X0,X2
BGT label1

label1:
BL bar
Find the “root” instructions.

- \( X_0 = X_3 + X_4 \)
- \( (X_3 + X_4) > X_2 \)
- \( \text{ADD} \ X_0, X_3, X_4 \)
- \( \text{CMP} \ X_0, X_2 \)
- \( \text{BGT} \ label1 \)

**Root instructions:** whose outputs will no longer be consumed by other instructions.
Find the “root” instructions.

\[ X_0 = X_3 + X_4 \]
\[ (X_3 + X_4) > X_2 \]

**Root instructions**: whose outputs will no longer be consumed by other instructions.

**Solution**: we perform a basic-block level data-flow analysis to identify root instructions.
Extract semantic formulas for root instructions.

```plaintext
foo:
...
MOV X3,X0
MOV X4,X1
...
ADD X0,X3,X4
CMP X0,X2
BGT label1
...
```

**AArch64 calling convention:**
- a -> X0
- b -> X1
- c -> X2

**label1:**
BL bar
Extract semantic formulas for root instructions.

```asm
foo:
  ...
  MOV X3,X0
  MOV X4,X1
  ...
  ADD X0,X3,X4
  CMP X0,X2
  BGT label1

label1:
  BL bar

AArch64 calling convention:
- a -> X0
- b -> X1
- c -> X2

X0 > X2
```
Extract semantic formulas for root instructions.

```
foo:
...
MOV X3,X0
MOV X4,X1
...
ADD X0,X3,X4
CMP X0,X2
BGT label1
label1:
BL  bar
...
```

AArch64 calling convention:
- a -> X0
- b -> X1
- c -> X2

\[(X3 + X4) > X2\]

\[X0 > X2\]
Extract semantic formulas for root instructions.

```
foo:
  ...
  MOV X3, X0
  MOV X4, X1
  ...
  ADD X0, X3, X4
  CMP X0, X2
  BGT label1

label1:
  BL bar
```

AArch64 calling convention:
- `a` -> `X0`
- `b` -> `X1`
- `c` -> `X2`

\[(X0 + X1) > X2\]
\[(X3 + X4) > X2\]
\[X0 > X2\]
Extract semantic formulas for root instructions.

foo(a,b,c){
  ...
  if (a+b > c)  
    bar(a+b);
  ...
}

(a + b) > c
(X0 + X1) > X2
(X3 + X4) > X2
X0 > X2

foo:
  ...
  MOV X3,X0
  MOV X4,X1
  ...
  ADD X0,X3,X4
  CMP X0,X2
  BGT label1
  label1:
  BL bar

AArch64 calling convention:
a -> X0
b -> X1
c -> X2
Extract semantic formulas for root instructions.

Solution: we use function-level, intra-procedure and under-constrained symbolic execution to obtain formulas.
Patch change site analysis. (Source level)

Binary signature translation.

Match in binary.
Matching

- Quick Pass.
Quick Pass.

```
foo(a,b,c){
    ...
    if (a+b > c)
        bar(a+b);
    ...
Cond. jmp
Call bar
```
Matching

- Quick Pass.

```plaintext
foo(a,b,c){
  ...
  if (a+b > c)
    bar(a+b);
  ...
```
foo(a,b,c) {
  ...
  if (a+b > c) {
    bar(a+b);
  }
  ...
}
Quick Pass.

```c
foo(a, b, c) {
    ...
    if (a+b > c) {
        bar(a+b);
    }
    ...
}
```
Quick Pass.

\[
\text{foo}(a,b,c)\{
\text{... if } (a+b > c) \text{ bar}(a+b); \text{ ...}
\]

Solution: look at easy-to-match attributes, e.g. topology, root instruction type, etc.
Matching

- Slow Pass.

```c
foo(a, b, c) {
  ... if (a+b > c) bar(a+b); ...
}
```
Matching

- Slow Pass.

```plaintext
foo(a,b,c) {
  ...
  if (a+b > c) bar(a+b);
  ...
}
```

Diagram:
- **Cond. jmp**
- **Call bar**
- **Cond. jmp**
- **Call bar**
- **Store**
- **Call bar**
- **Call sth**

Formulas:
- \((a + b) > c\)
- \(a \neq b\)
Matching

- Slow Pass.

```cpp
foo(a,b,c){
... if (a+b > c) bar(a+b);
...}
```

```
if (a+b > c)
bar(a+b);
```

```
(a + b) > c
Cond. jmp
Call bar
Store
Call bar
Cond. jmp
Call sth
```

```
Cond. jmp
```
Solution: basically we strictly compare two formulas simplified by Z3 solver, with necessary relaxations. (e.g. commutative operators)
Special (and Interesting) Cases
Func():

......

......

+ uni q_f unc_noi nl i ne()
Simply test the function call presence, no semantic formulas needed.
Func():
......
......
- f(a, b)
+ f(a, c)
......
......
That line matters? No, that parameter matters!
In evaluation:

\textcolor{red}{107} security patches crawled from Android Bulletin (Jun 2016 – May 2017)

\textcolor{red}{8} Android kernel images from \textcolor{red}{3} mainstream vendors.
### Accuracy

<table>
<thead>
<tr>
<th>Device</th>
<th>No.</th>
<th>Patch Cnt*</th>
<th>Build Date (mm/dd/yy)</th>
<th>Kernel Version</th>
<th>Accuracy</th>
<th>Online Matching Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>Samsung S7</td>
<td>0</td>
<td>102</td>
<td>06/24/16</td>
<td>3.18.20</td>
<td>42</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>102</td>
<td>09/09/16</td>
<td>3.18.20</td>
<td>43</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>102</td>
<td>01/03/17</td>
<td>3.18.31</td>
<td>85</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>102</td>
<td>05/18/17</td>
<td>3.18.31</td>
<td>92</td>
<td>4</td>
</tr>
<tr>
<td>LG G5</td>
<td>4</td>
<td>103</td>
<td>05/27/16</td>
<td>3.18.20</td>
<td>32</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>103</td>
<td>10/26/17</td>
<td>3.18.31</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>Huawei P9</td>
<td>6</td>
<td>31</td>
<td>02/22/16</td>
<td>3.10.90</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>30</td>
<td>05/22/17</td>
<td>4.11.18</td>
<td>25</td>
<td>2</td>
</tr>
</tbody>
</table>

* Some patches we collected are not applicable for certain test subject kernels.
### Accuracy

<table>
<thead>
<tr>
<th>Device</th>
<th>No.</th>
<th>Patch Cnt*</th>
<th>Build Date (mm/dd/yy)</th>
<th>Kernel Version</th>
<th>Accuracy</th>
<th>Online Matching Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total</td>
<td>Avg</td>
</tr>
<tr>
<td>Samsung S7</td>
<td>0</td>
<td>102</td>
<td>06/24/16</td>
<td>3.18.20</td>
<td>42 56 0 4(3.92%)</td>
<td>1690.43</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>102</td>
<td>09/09/16</td>
<td>3.18.20</td>
<td>43 55 0 4(3.92%)</td>
<td>1888.06</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>102</td>
<td>01/03/17</td>
<td>3.18.31</td>
<td>85 11 0 6(5.88%)</td>
<td>2421.44</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>102</td>
<td>05/18/17</td>
<td>3.18.31</td>
<td>92 4 0 6(5.88%)</td>
<td>1770.66</td>
</tr>
<tr>
<td>LG G5</td>
<td>4</td>
<td>103</td>
<td>05/27/16</td>
<td>3.18.20</td>
<td>32 65 0 6(5.88%)</td>
<td>2122.37</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>103</td>
<td>10/26/17</td>
<td>3.18.31</td>
<td>95 0 0 8(7.77%)</td>
<td>1384.47</td>
</tr>
<tr>
<td>Huawei P9</td>
<td>6</td>
<td>31</td>
<td>02/22/16</td>
<td>3.10.90</td>
<td>10 20 0 1(3.23%)</td>
<td>390.35</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>30</td>
<td>05/22/17</td>
<td>4.1.18</td>
<td>25 2 0 3(10.00%)</td>
<td>515.64</td>
</tr>
</tbody>
</table>

* Some patches we collected are not applicable for certain test subject kernels.

**Accuracy:** excellent, on average 94% accuracy w/o FP.
## Accuracy

<table>
<thead>
<tr>
<th>Device</th>
<th>No.</th>
<th>Patch Cnt*</th>
<th>Build Date (mm/dd/yy)</th>
<th>Kernel Version</th>
<th>Accuracy</th>
<th>Online Matching Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>Samsung S7</td>
<td>0</td>
<td>102</td>
<td>06/24/16</td>
<td>3.18.20</td>
<td>42</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>102</td>
<td>09/09/16</td>
<td>3.18.20</td>
<td>43</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>102</td>
<td>01/03/17</td>
<td>3.18.31</td>
<td>85</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>102</td>
<td>05/18/17</td>
<td>3.18.31</td>
<td>92</td>
<td>4</td>
</tr>
<tr>
<td>LG G5</td>
<td>4</td>
<td>103</td>
<td>05/27/16</td>
<td>3.18.20</td>
<td>32</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>103</td>
<td>10/26/17</td>
<td>3.18.31</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>Huawei P9</td>
<td>6</td>
<td>31</td>
<td>02/22/16</td>
<td>3.10.90</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>30</td>
<td>05/22/17</td>
<td>4.1.18</td>
<td>25</td>
<td>2</td>
</tr>
</tbody>
</table>

* Some patches we collected are not applicable for certain test subject kernels.

**FP:** we wrongly believe the patch is present. *Dangerous!*

**FN:** we wrongly believe the patch is not there. *Extra time to confirm.*

**Accuracy:** excellent, on average 94% accuracy w/o FP.
Why FN?

- Function inline.
- Function prototype change.
- Code customization.
- Patch adaptation.
- Other engineering issues.

Refer to section 6.2 in the paper for more details.
Why FN?

*Function inline:*

Added new callee function in the change site is inlined in different ways across reference and target binaries.
Why FN?

*Patch adaptation:*

The change site itself has been customized during patch porting.
## Performance

<table>
<thead>
<tr>
<th>Device</th>
<th>No.</th>
<th>Patch Cnt*</th>
<th>Build Date (mm/dd/yy)</th>
<th>Kernel Version</th>
<th>Accuracy</th>
<th>Online Matching Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>Samsung S7</td>
<td>0</td>
<td>102</td>
<td>06/24/16</td>
<td>3.18.20</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>102</td>
<td>09/09/16</td>
<td>3.18.20</td>
<td></td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>102</td>
<td>01/03/17</td>
<td>3.18.31</td>
<td></td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>102</td>
<td>05/18/17</td>
<td>3.18.31</td>
<td></td>
<td>92</td>
</tr>
<tr>
<td>LG G5</td>
<td>4</td>
<td>103</td>
<td>05/27/16</td>
<td>3.18.20</td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>103</td>
<td>10/26/17</td>
<td>3.18.31</td>
<td></td>
<td>95</td>
</tr>
<tr>
<td>Huawei P9</td>
<td>6</td>
<td>31</td>
<td>02/22/16</td>
<td>3.10.90</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>30</td>
<td>05/22/17</td>
<td>4.1.18</td>
<td></td>
<td>25</td>
</tr>
</tbody>
</table>

* Some patches we collected are not applicable for certain test subject kernels.
## Performance

Performance: acceptable, some cases may take long time to match, overall still much more efficient than manual work. Parallelization is also easily possible.

<table>
<thead>
<tr>
<th>Device</th>
<th>No.</th>
<th>Patch Cnt*</th>
<th>Build Date (mm/dd/yy)</th>
<th>Kernel Version</th>
<th>Accuracy</th>
<th>Online Matching Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>Samsung S7</td>
<td>0</td>
<td>102</td>
<td>06/24/16</td>
<td>3.18.20</td>
<td>42</td>
<td>56</td>
</tr>
<tr>
<td>Samsung S7</td>
<td>1</td>
<td>102</td>
<td>09/09/16</td>
<td>3.18.20</td>
<td>43</td>
<td>55</td>
</tr>
<tr>
<td>Samsung S7</td>
<td>2</td>
<td>102</td>
<td>01/03/17</td>
<td>3.18.31</td>
<td>85</td>
<td>11</td>
</tr>
<tr>
<td>Samsung S7</td>
<td>3</td>
<td>102</td>
<td>05/18/17</td>
<td>3.18.31</td>
<td>92</td>
<td>4</td>
</tr>
<tr>
<td>LG G5</td>
<td>4</td>
<td>103</td>
<td>05/27/16</td>
<td>3.18.20</td>
<td>32</td>
<td>65</td>
</tr>
<tr>
<td>LG G5</td>
<td>5</td>
<td>103</td>
<td>10/26/17</td>
<td>3.18.31</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>Huawei P9</td>
<td>6</td>
<td>31</td>
<td>02/22/16</td>
<td>3.10.90</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Huawei P9</td>
<td>7</td>
<td>30</td>
<td>05/22/17</td>
<td>4.1.18</td>
<td>25</td>
<td>2</td>
</tr>
</tbody>
</table>

* Some patches we collected are not applicable for certain test subject kernels.
# Un-ported patches

<table>
<thead>
<tr>
<th>CVE</th>
<th>Type**</th>
<th>Severity*</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2014-9781</td>
<td>P</td>
<td>High</td>
</tr>
<tr>
<td>CVE-2016-2502</td>
<td>P</td>
<td>High</td>
</tr>
<tr>
<td>CVE-2016-3813</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-4578</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-2184</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-7910</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-8413</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-10200</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-10229</td>
<td>E</td>
<td>Critical</td>
</tr>
</tbody>
</table>

* Obtained from Android security bulletin.

** P: Privilege Elevation  E: Remote Code Execution  I: Information Disclosure
Un-ported patches

<table>
<thead>
<tr>
<th>CVE</th>
<th>Type**</th>
<th>Severity*</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2014-9781</td>
<td>P</td>
<td>High</td>
</tr>
<tr>
<td>CVE-2016-2502</td>
<td>P</td>
<td>High</td>
</tr>
<tr>
<td>CVE-2016-3813</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-4578</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-2184</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-7910</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-8413</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-10200</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-10229</td>
<td>E</td>
<td>Critical</td>
</tr>
</tbody>
</table>

* Obtained from Android security bulletin.

** P: Privilege Elevation  E: Remote Code Execution  I: Information Disclosure

<table>
<thead>
<tr>
<th>Lag (month)</th>
<th>Cnt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>
Un-ported patches

Some critical patches were not propagated even after 6 months (confirmed)!

<table>
<thead>
<tr>
<th>CVE</th>
<th>Type**</th>
<th>Severity*</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2014-9781</td>
<td>P</td>
<td>High</td>
</tr>
<tr>
<td>CVE-2016-2502</td>
<td>P</td>
<td>High</td>
</tr>
<tr>
<td>CVE-2016-3813</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-4578</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-2184</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-7910</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-8413</td>
<td>I</td>
<td>Moderate</td>
</tr>
<tr>
<td>CVE-2016-10200</td>
<td>P</td>
<td>Critical</td>
</tr>
<tr>
<td>CVE-2016-10229</td>
<td>E</td>
<td>Critical</td>
</tr>
</tbody>
</table>

* Obtained from Android security bulletin.
** P: Privilege Elevation ** E: Remote Code Execution
** I: Information Disclosure

<table>
<thead>
<tr>
<th>Lag (month)</th>
<th>Cnt.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>
CVE-2016-7910

diff --git a/block/genhd.c b/block/genhd.c
index 3c9dede..0ad8796 100644
--- a/block/genhd.c
+++ b/block/genhd.c
@@ -856,6 +856,7 @@ static void disk_seqf_stop(
       , void *
       struct seq_file *seqf
     if (iter) {
-       class_dev_iter_exit(iter);
+       seqf->private = NULL;
       kfree(iter);
     }
   }
}
CVE-2016-7910

diff --git a/block/genhd.c b/block/genhd.c
index 3c9dede..0ad8796 100644
--- a/block/genhd.c
+++ b/block/genhd.c
@@ -856,6 +856,7 @@ static void disk_seqf_stop(
               , void
               *v)
               struct seq_file *seqf)
+  if (iter) {
+    class_dev_iter_exit(iter);
+    kfree(iter);
+    seqf->private = NULL;
+  }
+}

0x0  →  [X0] + offset
How much code do you write?
We use Angr as our symbolic execution engine. (w/ modifications)
We use Angr as our symbolic execution engine. (w/ modifications)

#code of Fiber: 5,097 LOC Python.
We use Angr as our symbolic execution engine. (w/ modifications)

#code of Fiber: 5,097 LOC Python.

Still under improvement.
We use Angr as our symbolic execution engine. (w/ modifications)

#code of Fiber: 5,097 LOC Python.

Still under improvement.

Now fully open-sourced on Github!

https://fiberx.github.io
Thanks!
Q & A

https://fiberx.github.io