VUDDY: A Scalable Approach for Vulnerable Code Clone Detection

Seulbae Kim, Seunghoon Woo, Heejo Lee, and Hakjoo Oh

May 23, 2017
Question

- Number of unpatched vulnerabilities in smartphone firmware’s source code?

200+ unpatched vulnerable code clones detected!
Motivation

• Number of open source software is increasing
Motivation

• Code clones – reused code fragments
  • Major cause of vulnerability propagation

CVE-2016-5195
Problem: Scalable & Accurate Vulnerable Code Clone Discovery
Scalable & Accurate Vulnerable Code Clone discovery

• Scalability

Software systems are getting bigger
Linux kernel – 25.4 MLoC
“L” Smart TV – 35 MLoC
Scalable & Accurate Vulnerable Code Clone discovery

• Accuracy

FP == increased time and efforts
Scalable & Accurate Vulnerable Code Clone discovery

- Previous approaches

<table>
<thead>
<tr>
<th>Matching Type</th>
<th>Authors</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token-level</td>
<td>Kamiya et al., CCFinder</td>
<td>TSE’02</td>
</tr>
<tr>
<td>Graph/tree matching</td>
<td>Jiang et al.,</td>
<td>ICSE’07</td>
</tr>
<tr>
<td>Bag-of-tokens</td>
<td>Sajnani et al., SourcererCC</td>
<td>ICSE’16</td>
</tr>
<tr>
<td>Line-level</td>
<td>Jang et al., ReDeBug</td>
<td>S&amp;P’12</td>
</tr>
<tr>
<td>File-level</td>
<td>Sasaki et al., FCFinder</td>
<td>MSR’10</td>
</tr>
</tbody>
</table>
Scalable & Accurate Vulnerable Code Clone discovery

• Goal

accuracy

scalability

Token-level matching
Kamiya et al., CCFinder (TSE’02)

Graph/tree matching
Jiang et al., (ICSE’07)

Bag-of-tokens matching
Sajnani et al., SourcererCC (ICSE’16)

Line-level matching
Jang et al., ReDeBug (S&P’12)

File-level matching
Sasaki et al., FCFinder (MSR’10)
Proposed Method: VUDDY
Demonstration of VUDDY
Proposed method: VUDDY

- VUDDY: VUlnerable coDe clone DiscoverY
Proposed method: VUDDY

• VUDDY: VUlnerable coDe clone DiscoverY
  • Searches for vulnerable code clones
Proposed method: VUDDY

• VUDDY: VUlnerable coDe clone DiscoverY
  • Searches for vulnerable code clones
  • Scales beyond 1 BLoC target
Proposed method: VUDDY

• VUDDY: VUlnerable coDe clone DiscoverY
  • Searches for vulnerable code clones
  • Scales beyond 1 BLoC target
  • Detects both known & unknown vulnerability
Proposed method: VUDDY

• VUDDY: VUlnerable coDe clone DiscoverY
  • Searches for vulnerable code clones
  • Scales beyond 1 BLoC target
  • Detects both known & unknown vulnerability
  • Low false positive rate
Proposed method: VUDDY

• Overview

vulnerable functions

fingerprinting

fingerprint dictionary of vulnerable functions

dictionary comparison

vulnerable code clones

A Program

a target program

fingerprinting

fingerprint dictionary of target functions
Collecting vulnerable code

• Vulnerability patching
Collecting vulnerable code

• Reconstructing vulnerability from security patch

Software repository → CVE patch → Old code (vulnerable)
Fingerprinting a program

A Program
Fingerprinting a program

1. Retrieve all functions from a program

```c
int sum (int a, int b) {
    return a + b;
}

void increment() {
    int num = 80;
    num++; // no return
}

void printer (char* src) {
    printf("%s", src);
}
```
Fingerprinting a program

2. Apply abstraction and normalization to functions

```c
int sum (int a, int b) {
    return a + b;
}

void increment() {
    int num = 80;
    num++; // no return
}

void printer (char* src) {
    printf("%s", src);
}
```

- `return fparam + fparam;`
- `dtype lvar = 80; lvar++;`
- `funcall("%s", fparam);`
Fingerprinting a program

3. Compute length and hash value

A Program

```
int sum (int a, int b)
{
    return a + b;
}
```

```
void increment()
{
    int num = 80;
    num++; // no return
}
```

```
void printer (char* src)
{
    printf("%s", src);
}
```

- `int sum (int a, int b)`
  - Length: 20
  - Hash Value: C94D9910...

- `void increment()`
  - Length: 20
  - Hash Value: D6E77882...

- `void printer (char* src)`
  - Length: 23
  - Hash Value: 9A45E4A1...
Fingerprinting a program

4. Store in a dictionary

- Program length: 20
  Hash val: C94D9910...

- Program length: 20
  Hash val: D6E77882...

- Program length: 23
  Hash val: 9A45E4A1...

"Fingerprint dictionary"
Abstraction

• Transform function by replacing
  • Formal parameters
  • Data types
  • Local variables
  • Function names

Level 0: No abstraction
1 void avg (float arr[], int len) {
2   static float sum = 0;
3   unsigned int i;
4
5   for (i = 0; i < len; i++) {
6     sum += arr[i];
7   }
8
9   printf("%f %d\n", sum/len, validate(sum));
10 }
Abstraction

- Transform function by replacing
  - Formal parameters
  - Data types
  - Local variables
  - Function names

Level 1: Formal parameter abstraction

```
void avg (float FPARAM[], int FPARAM) {
    static float sum = 0;
    unsigned int i;
    for (i = 0; i < FPARAM; i++) {
        sum += FPARAM[i];
    }
    printf("%f %d\n", sum/FPARAM, validate(sum));
}
```
Abstraction

• Transform function by replacing
  • Formal parameters
  • Data types
  • Local variables
  • Function names

Level 2: Local variable name abstraction
1 void avg (float FPARAM[], int FPARAM) {
2    static float LVAR = 0;
3    unsigned int LVAR;
4
5    for (LVAR = 0; LVAR < FPARAM; LVAR++) {
6        LVAR += FPARAM[LVAR];
7    }
8
9    printf("%f %d
", LVAR/FPARAM, validate(LVAR));
10 }
Abstraction

• Transform function by replacing
  • Formal parameters
  • Data types
  • Local variables
  • Function names

Level 3: Data type abstraction
1 `DTYPE` avg (`DTYPE` FPARAM[], `DTYPE` FPARAM) {
2     `DTYPE` LVAR = 0;
3     unsigned `DTYPE` LVAR;
4
5     for (LVAR = 0; LVAR < FPARAM; LVAR++) {
6         LVAR += FPARAM[LVAR];
7     }
8
9     printf("%f %d\n", LVAR/FPARAM, validate(LVAR));
10 }
Abstraction

• Transform function by replacing
  • Formal parameters
  • Data types
  • Local variables
  • Function names

```c
Level 4: Function call abstraction
1 DTYPE avg (DTYPE FPARAM[], DTYPE FPARAM) {
2   DTYPE LVAR = 0;
3   unsigned DTYPE LVAR;
4
5   for (LVAR = 0; LVAR < FPARAM; LVAR ++) {
6     LVAR += FPARAM[LVAR];
7   }
8
9   FUNCCALL("%f %d\n", LVAR/FPARAM, FUNCCALL(LVAR));
10 }
```
Normalization

- Remove
  - comments
  - tabs
  - white spaces
  - CRLF
- Convert into lowercase

```c
DTYPE avg (DTYPE FPARAM[], DTYPE FPARAM) {
    DTYPE LVAR = 0;
    unsigned DTYPE LVAR;
    for (LVAR = 0; LVAR < FPARAM; LVAR ++) {
        LVAR += FPARAM[LVAR];
    }
    FUNCCALL("%f %d\n", LVAR/FPARAM, FUNCCALL(LVAR));
}
dtype lvar=0;unsigned dtype lvar;for(lvar=0;lvar<fparam;lvar++){lvar+=fparam[lvar];}funccall("%f %d\n",lvar/fparam,funccall(lvar));
```
Vulnerable code clone detection

• By comparing two fingerprint dictionaries

repository → fingerprint dictionary of vulnerable functions
Vulnerable code clone detection

• By comparing two fingerprint dictionaries

repository → fingerprint dictionary of vulnerable functions

target program → fingerprint dictionary of target functions
Vulnerable code clone detection

• By comparing two fingerprint dictionaries

![Diagram]

Repository fingerprint dictionary of vulnerable functions

20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]

Target program fingerprint dictionary of target functions

20: [C94D9910, D6E77882]
23: [9A45E4A1]
Vulnerable code clone detection

• By comparing two fingerprint dictionaries

20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]

key\_lookup(20) hit

20: [C94D9910, D6E77882]
23: [9A45E4A1]
Vulnerable code clone detection

• By comparing two fingerprint dictionaries

Repository fingerprint dictionary of vulnerable functions

20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]

Target program fingerprint dictionary of target functions

20: [C94D9910, D6E77882]
23: [9A45E4A1]

key_lookup(20) hit → have **C94D9910** in common (CLONE!)
Vulnerable code clone detection

• By comparing two fingerprint dictionaries

20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]

key_lookup(21) fail

20: [C94D9910, D6E77882]
23: [9A45E4A1]
Vulnerable code clone detection

- By comparing two fingerprint dictionaries

```
repository                                    fingerprint dictionary of vulnerable functions
```

```
20: [ABCDEF01, C94D9910]
21: [D155F630]
22: [C67F45FD, DDBF3838]
```

key_lookup(22) fail

```
target program                                fingerprint dictionary of target functions
```

```
20: [C94D9910, D6E77882]
23: [9A45E4A1]
```
Vulnerable code clone detection

• By comparing two fingerprint dictionaries

<table>
<thead>
<tr>
<th>Repository</th>
<th>Fingerprint Dictionary of Vulnerable Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>20:</td>
<td>[ABCDEF01, C94D9910]</td>
</tr>
<tr>
<td>21:</td>
<td>[D155F630]</td>
</tr>
<tr>
<td>22:</td>
<td>[C67F45FD, DDBF3838]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target Program</th>
<th>Fingerprint Dictionary of Target Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>20:</td>
<td>[C94D9910, D6E77882]</td>
</tr>
<tr>
<td>23:</td>
<td>[9A45E4A1]</td>
</tr>
</tbody>
</table>

```c
int sum (int a, int b) {
    return a + b;
}
```
Performance Evaluation
&
Case Study
Performance

• Scalability evaluation
  • Dataset: 25 K GitHub projects (>1 push, >1 star during Jan 1~July 28, 2016)
  • Execution time when varying size of target programs are given to VUDDY, CCFinderX, DECKARD, ReDeBug, and SourcererCC
Performance

• Accuracy evaluation
  • Vulnerability database VS Apache HTTPD 2.4.23 (350 KLoC)
    • TP: CCFinderX > VUDDY > DECKARD > SourcererCC (the greater, the better)
    • FP: VUDDY < SourcererCC < CCFinderX < DECKARD (the lower, the better)

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>VUDDY</td>
<td>22 s</td>
<td>9</td>
<td>0</td>
<td>3</td>
<td>1.000</td>
</tr>
<tr>
<td>SourcererCC</td>
<td>125 s</td>
<td>2</td>
<td>54</td>
<td>10</td>
<td>0.036</td>
</tr>
<tr>
<td>DECKARD</td>
<td>234 s</td>
<td>4</td>
<td>458</td>
<td>8</td>
<td>0.009</td>
</tr>
<tr>
<td>CCFinderX</td>
<td>1201 s</td>
<td>11</td>
<td>63</td>
<td>1</td>
<td>0.147</td>
</tr>
</tbody>
</table>

TABLE I: Accuracy of VUDDY, SourcererCC, DECKARD, and CCFinderX when detecting clones between the vulnerability database and Apache HTTPD 2.4.23
Performance

• VUDDY vs ReDeBug (CMU, S&P’12)
  • Detecting vulnerable code clones in an Android smartphone’s firmware (15 MLoC)

<table>
<thead>
<tr>
<th></th>
<th>VUDDY</th>
<th>ReDeBug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing time</td>
<td>17 m 3 s</td>
<td>11 m 16 s</td>
</tr>
<tr>
<td>Clone detection time</td>
<td>1.09 s</td>
<td>16 m 59 s</td>
</tr>
<tr>
<td># initial reports</td>
<td>206</td>
<td>2,090</td>
</tr>
<tr>
<td># true positives</td>
<td>206</td>
<td>202</td>
</tr>
<tr>
<td># false positives</td>
<td>0</td>
<td>1,888</td>
</tr>
</tbody>
</table>

TABLE II: Comparison of VUDDY and ReDeBug, targeting Android firmware
Performance

- VUDDY vs ReDeBug (CMU, S&P’12)
  - Detecting vulnerable code clones in an Android smartphone’s firmware (15 MLoC)

![Chart showing performance comparison between VUDDY and ReDeBug]

- Generated fingerprints can be reused
- Actual detection in practice: 1000x faster
Case study

• Unknown vulnerability detected in Linux kernel (even in 4.11.1)

Original patch for CVE-2008-3528 targeting ext2 file system

1 struct ext2_dir_entry_2 * ext2_dotdot (struct inode *dir, struct page **p)
2 {
3    struct page *page = ext2_get_page(dir, 0);
4    struct page *page = ext2_get_page(dir, 0, 0);
5    ext2_dirent *de = NULL;
6
7    if (!IS_ERR(page)) {

Patched function in ext2 file system

1 struct ext2_dir_entry *ext2_dotdot (struct inode * dir, struct page **p)
2 {
3    struct page *page = ext2_get_page(dir, 0, 0);
4    struct ext2_dir_entry *de = NULL;
5
6    if (!IS_ERR(page)) {

Vulnerable function in nilfs2 file system

1 struct nilfs_dir_entry *nilfs_dotdot (struct inode * dir, struct page **p)
2 {
3    struct page *page = nilfs_get_page(dir, 0);
4    struct nilfs_dir_entry *de = NULL;
5
6    if (!IS_ERR(page)) {

Could trigger “printk flood” & DoS in CentOS 7, and Ubuntu14.04
Case study

• Zero-day in Apache HTTPD 2.4.23 (2.4.20 through 2.4.25)
  • HTTPD uses unpatched Expat library for parsing XML
    • vulnerable to CVE-2012-0876
  • Hash DoS attack triggered by sending a crafted packet!

```c
// Vulnerable function in httpd/src/lib/apr-util/xml/expat/lib/xmlparse.c, lines 5429-5433.
for (i = 0; i < table->size; i++) {
  if (table->v[i]) {
    unsigned long newHash = hash(table->v[i]->name);
    size_t j = newHash & newMask;
    step = 0;
```

```c
100.1
```
Summary
Summary

• VUDDY is an approach capable of detecting software vulnerability using a database of previously security-patched functions
Summary

• VUDDY is an approach capable of detecting **software vulnerability** using a database of previously security-patched functions.

• Applying **abstraction** to the functions enable identifying **unknown vulnerable functions** while still maintaining a low margin of errors.
Summary

• VUDDY is an approach capable of detecting software vulnerability using a database of previously security-patched functions

• Applying abstraction to the functions enable identifying unknown vulnerable functions while still maintaining a low margin of errors

• Function-level granularity and length-based filtering reduces the number of signature comparisons, guaranteeing high scalability
Summary

• VUDDY is an approach capable of detecting software vulnerability using a database of previously security-patched functions

• Applying abstraction to the functions enable identifying unknown vulnerable functions while still maintaining a low margin of errors

• Function-level granularity and length-based filtering reduces the number of signature comparisons, guaranteeing high scalability

• Open web service
  • Implementation and testing available at https://iotcube.net