Pilot Study on Semi-Automated Patch Differing by Applying Machine-Learning Techniques

Asuka Nakajima (@AsuNa_jp)
NTT Secure Platform Laboratories
#whoami

- **Asuka Nakajima (@AsuNa_jp)**
  Security Researcher at NTT Secure Platform Laboratories
  Vulnerability Discovery / Reverse Engineering / IoT Security

- **Founder of “CTF for GIRLS”**
  First female infosec community in Japan (est.2014)

- **Black Hat Asia Review Board**
  From 2018-Present

- **Veteran Conference/Event Speaker**
  BlackHatUSA 2019, AsiaCCS 2019, AIS3 2018/2016, PHDays IV, SECCON, etc
## Agenda

**Background**

**PART 1**

Extracting Security Fix Patterns Using Unsupervised Machine Learning Algorithm*

**PART 2**

Classifying Security Fixes and Other Fixes*

**Conclusion**

*Original Paper*

Asuka Nakajima, Ren Kimura, Yuhei Kawakoya, Makoto Iwamura, Takeo Hariu, “An Investigation of Method to Assist Identification of Patched Part of the Vulnerable Software Based on Patch Diffing” Multimedia, Distributed, Cooperative, and Mobile Symposium, June 2017, Japan
What is Patch Diffing?

Identify vulnerable part & Create 1-day exploit
Example: CVE-2006-4691 (MS06-70)

Stacked-Based Buffer overflow in NetpManageIPCConnect Function

Before Patched (netapi32.dll)

After Patched (netapi32.dll)

```assembly
if ( !v5 )
{
    _wcsncpy(&Dest, L"\\\\\\");
    v6 = (wchar_t *)&v24;
}
```

Pseudo Code

```
if ( _wcslen(Str) > 0x101 ){
    NetpLogPrintHelper("NetpManageIPCConnect:
    server name %ws too long
- error out\n", (char)Str);
    return 87;
}
```

Security Check

Pseudo Code

```
if ( *Str != 92 ){
    _wcsncpy(&Dest, L"\\\\\\");
    v4 = (wchar_t *)&v24;
}
```
Tools for Patch Diffing

- **Bindiff** (Zynamics)
  - [https://www.zynamics.com/bindiff.html](https://www.zynamics.com/bindiff.html)
  - Acquired by Google

- **Turbodiff** (Core SECURITY)

- **Diaphora** (Joxean Koret)
  - [http://diaphora.re/](http://diaphora.re/)

However, patch diffing is still a difficult task because it requires deep knowledge and experience.

Semi-automated patch diffing
Previous Work

**DarunGrim**
- Shows the candidate functions that security fixes might have been applied

**Approach**
- Use heuristics pattern-matching rules to identify the candidate functions
  - **These patterns are manually defined by the developer**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Type</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmp</td>
<td>Opcode</td>
<td>+1</td>
</tr>
<tr>
<td>test</td>
<td>Opcode</td>
<td>+1</td>
</tr>
<tr>
<td>0xFFFFFFF</td>
<td>Immediate Value</td>
<td>+3</td>
</tr>
<tr>
<td>wcslen</td>
<td>Function Name</td>
<td>+2</td>
</tr>
<tr>
<td>strlen</td>
<td>Function Name</td>
<td>+2</td>
</tr>
<tr>
<td>StringCchCopyW</td>
<td>Function Name</td>
<td>+2</td>
</tr>
<tr>
<td>ULongLongToUlong</td>
<td>Function Name</td>
<td>+2</td>
</tr>
</tbody>
</table>

**Machine learning techniques could be applied**
Extracting Security Fix Patterns Using Unsupervised Machine Learning Algorithm

PART 1
Null Pointer Dereference

✓ Occurs when a program attempts to read or write to memory with a NULL pointer
✓ Check weather the pointer is NULL or not

Hypothesis

Similar Types of Vulnerabilities will be Fixed in a Similar Manner

CVE-2014-0198
@@ -672,10 +675,6 @@
static int do_ssl3_write(SSL *s,
+ if (wb->buf == NULL)
+ if (!ssl3_setup_write_buffer(s))
+ return -1;
+ if (len == 0 && !create_empty_fragment)
  return 0;

CVE-2015-0288
@@ -92,8 +92,6 @@
X509_REQ *X509_to_X509_REQ(X509 *x,
  pktmp = X509_get_pubkey(x);
+ if (pktmp == NULL)
+ goto err;
  i = X509_REQ_set_pubkey(ret, pktmp);
  EVP_PKEY_free(pktmp);

Extract Fix Patterns Using Unsupervised Machine Learning Algorithm (Cluster Analysis)
Challenges

Challenge 1 : Optimization

1. Basic Block Reordering
2. Instruction Reordering
3. Operand Changes
4. Inline Expansion / Loop Unrolling

Challenge 2 : Other Fixes May Have Been Applied
# Challenge 1: Basic Block Reordering

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Program1 (Before Patched)</th>
<th>Program2 (After Patched)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2015-1788</td>
<td>call 0x81031d0 &lt;BN_copy&gt; &lt;br&gt; test eax,eax &lt;br&gt; setnebl &lt;br&gt; jmp 0x820337a &lt;BN_GF2m_mod_inv+970&gt;</td>
<td>call 0x81031d0 &lt;BN_copy&gt; &lt;br&gt; test eax,eax &lt;br&gt; setnebl &lt;br&gt; jmp 0x820338a &lt;BN_GF2m_mod_inv+986&gt;</td>
</tr>
<tr>
<td></td>
<td>mov eax,DWORD PTR [esp+0x38] &lt;br&gt; mov DWORD PTR [esp+0x4],edi &lt;br&gt; mov DWORD PTR [esp],eax &lt;br&gt; call 0x8102f90 &lt;bn_expand2&gt; &lt;br&gt; jmp 0x8203149 &lt;BN_GF2m_mod_inv+409&gt; &lt;br&gt; lea eax,[esp+0x58] &lt;br&gt; mov edx,eax &lt;br&gt; jmp 0x820339a &lt;BN_GF2m_mod_inv+1002&gt; &lt;br&gt; mov eax,DWORD PTR [esp+0x3c] &lt;br&gt; mov DWORD PTR [esp],eax &lt;br&gt; call 0x8102f90 &lt;bn_expand2&gt; &lt;br&gt; jmp 0x8203149 &lt;BN_GF2m_mod_inv+409&gt; &lt;br&gt; mov eax,DWORD PTR [esp+0x38] &lt;br&gt; mov DWORD PTR [esp+0x4],edi &lt;br&gt; mov DWORD PTR [esp+0x3c] &lt;br&gt; mov DWORD PTR [esp],eax &lt;br&gt; call 0x8102f90 &lt;bn_expand2&gt; &lt;br&gt; jmp 0x8102f90 &lt;bn_expand2&gt; &lt;br&gt; jmp 0x820339a &lt;BN_GF2m_mod_inv+1002&gt; &lt;br&gt; mov eax,DWORD PTR [esp+0x3c] &lt;br&gt; mov DWORD PTR [esp],eax &lt;br&gt; call 0x8102f90 &lt;bn_expand2&gt; &lt;br&gt; jmp 0x8102f90 &lt;bn_expand2&gt; &lt;br&gt; jmp 0x820339a &lt;BN_GF2m_mod_inv+1002&gt;</td>
<td></td>
</tr>
</tbody>
</table>
# Challenge 1

## Instruction Reordering

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Program1</th>
<th>Program2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2015-1789</td>
<td>mov [esp+44h+var_4], eax</td>
<td>mov [esp+44h+var_4], eax</td>
</tr>
<tr>
<td></td>
<td>push esi</td>
<td>push esi</td>
</tr>
<tr>
<td></td>
<td>mov ebp, [esp+4Ch+arg_4]</td>
<td>mov ebp, [esp+4Ch+arg_4]</td>
</tr>
<tr>
<td></td>
<td>push esi</td>
<td>push esi</td>
</tr>
<tr>
<td></td>
<td>mov esi, [esp+4Ch+arg_0]</td>
<td>mov esi, [esp+4Ch+arg_0]</td>
</tr>
<tr>
<td></td>
<td>mov ecx, [esi]</td>
<td>mov ecx, [esi]</td>
</tr>
<tr>
<td></td>
<td>mov eax, [esi + 8]</td>
<td>mov eax, [esi + 8]</td>
</tr>
<tr>
<td></td>
<td>push edi</td>
<td>push edi</td>
</tr>
<tr>
<td></td>
<td>mov edi, [esi+4]</td>
<td>mov edi, [esi+4]</td>
</tr>
<tr>
<td></td>
<td>cmp edi, 17h</td>
<td>cmp edi, 17h</td>
</tr>
</tbody>
</table>

**IDA Pro**
## Challenge 1: Operand Changes

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Program1</th>
<th>Program2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-5023</td>
<td>xor ebx, ebx</td>
<td>xor r12d, r12d</td>
</tr>
<tr>
<td></td>
<td>add rsp, 38h</td>
<td>add rsp, 38h</td>
</tr>
<tr>
<td></td>
<td>mov eax, ebx</td>
<td>mov eax, r12d</td>
</tr>
<tr>
<td></td>
<td>pop rbx</td>
<td>pop rbx</td>
</tr>
<tr>
<td></td>
<td>pop rbp</td>
<td>pop rbp</td>
</tr>
<tr>
<td></td>
<td>pop r12</td>
<td>pop r12</td>
</tr>
<tr>
<td></td>
<td>pop r13</td>
<td>pop r13</td>
</tr>
<tr>
<td></td>
<td>retn</td>
<td>retn</td>
</tr>
</tbody>
</table>

Register is different (ebx -> r12d)
## Challenge 1: Inline Expansion/Loop Unrolling

<table>
<thead>
<tr>
<th>Inline Expansion</th>
<th>Source code</th>
<th>Program1 (Before)</th>
<th>Program2 (After)</th>
</tr>
</thead>
<tbody>
<tr>
<td>void my_print(int n){ printf(&quot;%d&quot;, n); } int main(){ int n = 1; my_print(n); return 0; }</td>
<td>&lt;main&gt;: push ebp mov ebp,esp sub esp,0x4 mov DWORD PTR [ebp-0x4],0x1 push DWORD PTR [ebp-0x4] call 804840b &lt;my_print&gt; add esp,0x4 mov eax,0x0 leave ret</td>
<td>&lt;main&gt;: push 0x1 push 0x80484e0 push 0x1 call 8048310 __printf_chk@plt add esp,0xc xor eax,eax ret</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loop Unrolling</th>
<th>Source code</th>
<th>Program1 (Before)</th>
<th>Program2 (After)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int main(){ int i; for(i = 0; i &lt; 3; i++){ printf(&quot;HelloWorld!&quot;); } return 0; }</td>
<td>&lt;main&gt;: push ebp mov ebp,esp sub esp,0x4 mov DWORD PTR [ebp-0x4],0x0 jmp 804842b &lt;main+0x20&gt; push 0x80484c0 call 8048310 <a href="mailto:printf@plt">printf@plt</a> add esp,0x4 add DWORD PTR [ebp-0x4],0x1 cmp DWORD PTR [ebp-0x4],0x2 jle 804841a &lt;main+0xf&gt;</td>
<td>&lt;main&gt;: push 0x80484d0 push 0x1 push 0x1 push 0x8048310 __printf_chk@plt push 0x80484d0 push 0x1 call 8048310 __printf_chk@plt push 0x80484d0 push 0x1 call 8048310 __printf_chk@plt push 0x80484d0 push 0x1 call 8048310 __printf_chk@plt</td>
<td></td>
</tr>
</tbody>
</table>
Challenge 2

- Other fixes may have been applied
  1. Bug Fixes
  2. Refactoring
  3. Feature Updates
Experiment Overview

- **Dataset**
  - **Target Software: OpenSSL 1.0.1**
    - Collected 62 Security Fixes
  
- **Cluster Analysis**
  - Hierarchical Clustering Algorithm

**GOAL**

Extract security fix patterns which could be used to support the semi-automated patch diffing
Data Collection Method [1/3]

- OpenSSL 1.0.1 (git / 4675a56 (openssl 1.0.1 stable))

**STEP 1**

Analyze Commit Log & Release note*

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Type</th>
<th>Hash value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2012-2110</td>
<td>Before</td>
<td>d36e0ee460f41d6b64015 455c4f5414a319865c3</td>
</tr>
<tr>
<td></td>
<td>Patched</td>
<td>8d5505d099973a06781b7 e0e5b65861859a7d994</td>
</tr>
<tr>
<td>CVE-2016-6304</td>
<td>Before</td>
<td>151ad2e5cc23284a059e0 f155505006a1c9fad9</td>
</tr>
<tr>
<td></td>
<td>Patched</td>
<td>2c0d295e26306e15a92eb 23a84a1802005c1c137</td>
</tr>
</tbody>
</table>

**STEP 2**

Diff the source code & Identify the patched part (Function)

```c
@@ -260,7 +265,11 @@ int BN_dec2bn(BIGNUM **bn, const char *a)
    a++;
 }
 - for (i = 0; isdigit((unsigned char)a[i]); i++) ;
+ for (i = 0; i <= (INT_MAX/4) && isdigit((unsigned char)a[i]); i++)
+     continue;
+     if (i > INT_MAX/4)
```

**STEP 3**

Compile/Disassemble before & after patched source code

*OpenSSL 1.0.1 Series Release Notes
Data Collection Method [2/3]

- Feature Extraction Method

**STEP 1**

**Before Patched**
- push
- mov
- sub
- jmp
- ...

**After Patched**
- push
- mov
- sub
- + mov
- + mov
- jmp
- ...

**STEP 2**

**Increased Instructions**
- mov
- mov
- cmp
- jne
- test
- je
- push
- jmp
- lea
- pop
- ...

**Before Normalization**
- mov
- mov
- cmp
- jne
- test
- je
- push
- jmp
- lea
- pop
- ...

**After Normalization**
- trans
- trans
- cmp
- jump
- ...

**STEP 3**

**Normalized Instructions**
- trans
- trans
- cmp
- jump
- ...

**# of occurrences (each instruction)**
- trans: 2
- cmp: 2
- jump: 3
- stack: 2
- lea: 1
- ...

Extract the increased instructions

Normalize the instructions

Count the number of occurrences of each normalized instruction
## Opcode (Instruction) Normalization

- Summarized and expressed the instructions that fall into similar categories by one (normalized) instruction
  - e.g.) Branch instruction such as `jns, jle, jne, jge` are normalized as “jump”
  - Normalized the instructions which appeared in the security fixes (Function)

<table>
<thead>
<tr>
<th>Normalized Instruction</th>
<th>Type of Instruction</th>
<th>Target Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jump</td>
<td>Branch</td>
<td>jns, jle, jne, jge, jae, jmp, js, jl, je, jg, ja, jb, jbe</td>
</tr>
<tr>
<td>trans</td>
<td>Data Transfer</td>
<td>movxz, mov, movsx, xchg, cdq</td>
</tr>
<tr>
<td>ctrans</td>
<td>Conditional Data Transfer</td>
<td>cmovge, cmovae, cmovs, cmovns, cmove, cmovne</td>
</tr>
<tr>
<td>stack</td>
<td>Stack Manipulation</td>
<td>push, pop</td>
</tr>
<tr>
<td>logical</td>
<td>Logical Operation</td>
<td>and, xor, or, not</td>
</tr>
<tr>
<td>arith</td>
<td>Arithmetic Operation</td>
<td>sub, add, imul, neg, adc</td>
</tr>
<tr>
<td>nop</td>
<td>No Operation</td>
<td>nop</td>
</tr>
<tr>
<td>bop</td>
<td>Bit/Byte Operation</td>
<td>bt, setne, sete</td>
</tr>
<tr>
<td>shift</td>
<td>Shift Operation</td>
<td>shr, shl, sar</td>
</tr>
<tr>
<td>func</td>
<td>Function Operation</td>
<td>call, ret</td>
</tr>
<tr>
<td>str</td>
<td>String Operation</td>
<td>repz *</td>
</tr>
<tr>
<td>cmp</td>
<td>Comparison</td>
<td>test, cmp</td>
</tr>
<tr>
<td>lea</td>
<td>Address Computation</td>
<td>lea</td>
</tr>
</tbody>
</table>
Cluster Analysis [1/2]

Divides data into groups that are meaningful/useful
Cluster Analysis [2/2]

Divides data into groups that are meaningful/useful

- **Hierarchical Clustering Algorithm**
  - Produce a classification in which small clusters of very similar data points are nested within larger clusters of less closely-related data points*
    - e.g.) Agglomerative Hierarchical Clustering

- **Non-Hierarchical Clustering Algorithm**
  - Generates a classification by partitioning dataset*
    - e.g.) K-means Clustering

*Hierarchical and non-Hierarchical Clustering
https://www.daylight.com/meetings/mug96/barnard/E-MUG95.html
Hierarchical Clustering Algorithm

- Produce a classification in which small clusters of very similar data points are nested within larger clusters of less closely-related data points*
  - e.g.) Agglomerative Hierarchical Clustering

Ward’s Method

\[ d(A,B) = E(A \cup B) - E(P) - E(Q) \]

Euclidean Distance

\[ \|a - b\|_2 = \sqrt{\sum_{i}(a_i - b_i)^2} \]

*Hierarchical and non-Hierarchical Clustering
https://www.daylight.com/meetings/mug96/barnard/E-MUG95.html
CWE [1/2]

- **CWE** (Common Weakness Enumeration)
  - **List of Software Weakness Types**
    - Gives a unique identifier (CWE-ID) to each type
    - e.g., CWE-120: Buffer Copy without Checking Size of Input
    - Latest Version: 3.4 / Total 808 Weaknesses.

https://cwe.mitre.org/data/definitions/120.html
CWE organizes a wide variety of weakness types in a hierarchical structure

- The weakness types at higher levels in the structure gives a more abstract and broader concept*

CWE Overview, IPA
CWE organizes a wide variety of weakness types in a hierarchical structure

- The weakness types at higher levels in the structure gives a more abstract and broader concept*

Structure Types
- Development Concepts
- Research Concepts
- Architectural Concepts

* Used these root CWE-IDs as labels

CWE Overview, IPA
## Details of the Cluster 1

Most of the labels are CWE-19 (Data Processing Error)

<table>
<thead>
<tr>
<th>CWE-ID</th>
<th>CVE-ID</th>
<th>Feature Vectors (Normalized Instruction : # of Occurrences)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-19</td>
<td>CVE-2016-6306</td>
<td>jump: 9, trans: 7, cmp: 6, lea: 4, arith: 3, func: 1</td>
</tr>
<tr>
<td>CWE-19</td>
<td>CVE-2016-0797</td>
<td>jump: 7, lea: 4, cmp: 4, trans: 2, arith: 2</td>
</tr>
<tr>
<td>CWE-19</td>
<td>CVE-2014-3508</td>
<td>jump: 9, trans: 5, cmp: 5, stack: 4, nop: 3, lea: 2, arith: 2, bop: 1, func: 1</td>
</tr>
<tr>
<td>CWE-398</td>
<td>CVE-2014-5139</td>
<td>jump: 7, stack: 4, cmp: 3, logical: 2, trans: 2, nop: 2, func': 1</td>
</tr>
</tbody>
</table>

### Summary

- Most of the vulnerabilities in this cluster are related to the memory or value manipulation error, which was not initially expected by developers
  - e.g.) Out-of-bounds read, info-leak, integer overflow)
- A certain number of Comparison/Branch/Arithmetic Operation instructions exist
### CVE-2016-0797

**Integer Overflow Vulnerability**

---

#### Patched Part of CVE-2016-0797

```c
@@ -190,11 +189,7 @@ int BN_hex2bn(BIGNUM **bn,
     }
+    for (i = 0; i <= (INT_MAX/4) &&
+          isxdigit((unsigned char)a[i]); i++)
+        continue;
+    
+    if (i > INT_MAX/4)
+        goto err;
-   for (i = 0; isxdigit((unsigned char)a[i]); i++) ;
```

---

**Added a check to confirm the integer value is under the expected upper limit**
### Details of the Cluster 2

Most of the labels are CWE-254 (Security Features)

<table>
<thead>
<tr>
<th>CWE-ID</th>
<th>CVE-ID</th>
<th>Feature Vectors: (Normalized Instruction : # of Occurrences)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-254</td>
<td>CVE-2015-1793</td>
<td>trans: 4, jump: 3, ctrans: 1, nop: 1, func: 1</td>
</tr>
<tr>
<td>CWE-254</td>
<td>CVE-2014-3567</td>
<td>trans: 4, jump: 2, func: 1</td>
</tr>
<tr>
<td>CWE-254</td>
<td>CVE-2014-3470</td>
<td>trans: 4, jump: 2, nop: 2, lea: 1, func: 1, cmp: 1</td>
</tr>
<tr>
<td>CWE-254</td>
<td>CVE-2015-0205</td>
<td>trans: 5, func: 1</td>
</tr>
<tr>
<td>CWE-254</td>
<td>CVE-2014-0224</td>
<td>trans: 5, jump: 3, logical: 2, cmp: 1</td>
</tr>
<tr>
<td>CWE-19</td>
<td>CVE-2014-0195</td>
<td>trans: 6, jump: 3, cmp: 1</td>
</tr>
</tbody>
</table>

**Summary**

- Most of the security fixes for the vulnerabilities in this cluster contain some sort of error handling function
- A certain number of Data Transfer/Branch/Function related instructions exist
CVE-2014-3470

Patched Part of CVE-2014-3470

@@ -2512,13 +2512,6 @@
     int ssl3_send_client_key_exchange
     int field_size = 0;

     if (s->session->sess_cert == NULL)
         ssl3_send_alert(s, SSL3_AL_FATAL,
                         SSL_AD_UNEXPECTED_MESSAGE);
+        SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
                  SSL_R_UNEXPECTED_MESSAGE);
     goto err;
   }

Added two error handling function + exit the function
Discussions

- Why Only Two Clusters?
  - Some vulnerabilities are found in multiple functions
    - Similar functions contain the same vulnerability

- How to Improve
  - Include other features such as function name?
  - Collect more security fixes
    - Use vulnerability corpus generation tools? (e.g., LAVA)
  - Use other machine learning techniques

- For Semi-Automated Patch-Diffing
  - Calculate the similarity between the extracted security fix patterns (instructions) and the difference (increased instructions) found by the patch diffing

*LAVA: Large-scale Automated Vulnerability Addition

PART 2

Classifying Security Fixes and Other Fixes
Classifying Security Fixes and Other Fixes

- **Dataset**
  - OpenSSL 1.0.1 (62 Security Fixes / 377 Other fixes)

- **Classification Method**
  - Supervised Linear Classifier
    - Soft Margin Support Vector Machine (SVM)
    - Kernel: RBF (C=10, γ = 0.001)

- **Experiment**
  - Used 62 Security Fixes and 62 Other fixes (Random sampling)
  - Conducted 10-fold Cross-Validation 3 times
    - Perform random sampling for each cross-validation

- **Environment**
  - OS: Ubuntu 14.04, Compiler: gcc 5.4.0
Support Vector Machine (SVM)

Method used for classification (+regression) tasks
### Result

<table>
<thead>
<tr>
<th></th>
<th>Type of Fix</th>
<th>Dataset 1</th>
<th>Dataset 2</th>
<th>Dataset 3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security Fix</td>
<td>0.62</td>
<td>0.54</td>
<td>0.54</td>
<td><strong>0.56</strong></td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>0.59</td>
<td>0.54</td>
<td>0.54</td>
<td><strong>0.55</strong></td>
</tr>
<tr>
<td><strong>Precision</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Fix</td>
<td>0.70</td>
<td>0.57</td>
<td>0.57</td>
<td>0.61</td>
<td><strong>0.61</strong></td>
</tr>
<tr>
<td>Other</td>
<td>0.59</td>
<td>0.54</td>
<td>0.54</td>
<td>0.55</td>
<td><strong>0.55</strong></td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Fix</td>
<td>0.42</td>
<td>0.49</td>
<td>0.42</td>
<td>0.41</td>
<td><strong>0.41</strong></td>
</tr>
<tr>
<td>Other</td>
<td>0.82</td>
<td>0.71</td>
<td>0.68</td>
<td>0.73</td>
<td><strong>0.73</strong></td>
</tr>
<tr>
<td><strong>F-Score</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Fix</td>
<td>0.53</td>
<td>0.46</td>
<td>0.44</td>
<td>0.47</td>
<td><strong>0.47</strong></td>
</tr>
<tr>
<td>Other</td>
<td>0.68</td>
<td>0.61</td>
<td>0.63</td>
<td>0.64</td>
<td><strong>0.64</strong></td>
</tr>
</tbody>
</table>
Summary of Result

- **Summary**
  - Overall Accuracy: 56% (average)
  - Security Fixes: Precision 61% / Recall 41% (average)
  - Other Fixes: Precision 55% / Recall 73% (average)

- **Discussions**
  - Use other metrics? (e.g., Cyclomatic complexity)

**Glossary**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Ratio of the number of correctly labeled fixes to the number of all fixes in the dataset</td>
</tr>
<tr>
<td>Precision</td>
<td>Ratio of the number of correctly labeled security fixes to the number of all fixes labeled as &quot;security fix&quot; by the program</td>
</tr>
<tr>
<td>Recall</td>
<td>Ratio of the number of correctly labeled security fixes to the number of all security fixes in the dataset</td>
</tr>
<tr>
<td>F-Score</td>
<td>Harmonic mean of the Precision and Recall</td>
</tr>
</tbody>
</table>
Summary & Conclusion

- Patch diffing is still a difficult task because it requires a deep knowledge and experience.
- Extracted security fix patterns which could be used to support the semi-automated patch diffing.
- Conducted an experiment to see if it is possible to distinguish between security fixes and other fixes.

Provided insights for future research related to the semi-automated patch diffing.
Appendix [1/3]

Original Paper

Asuka Nakajima, Ren Kimura, Yuhei Kawakoya, Makoto Iwamura, Takeo Hariu, “An Investigation of Method to Assist Identification of Patched Part of the Vulnerable Software Based on Patch Diffing” Multimedia, Distributed, Cooperative, and Mobile Symposium, June 2017, Japan

Download URL
https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_uri&item_id=190132&file_id=1&file_no=1
Appendix [2/3]

● Other Research (1)


● [PDF] https://www.cylab.cmu.edu/_files/pdfs/tech_reports/CMUCyLab19001.pdf

1-Day Risk: Unsynchronized Patch Release (Geographical Arbitrage)

Example: CVE-2017-7852

Patch Release Timeline

DCS-932L RevA 2015/Nov/18

DCS-932L RevA 2016/Jul/19

244 Days

Vendor: D-Link, Product: Network Camera

Revealed Significant 1-Day Risk Related to IoT
Appendix [3/3]

Other Activities (Female InfoSec Community)

- **CTF for GIRLS**: http://girls.seccon.jp (Twitter:@ctf4g)
- **Asuka Nakajima**, Suhee Kang, Hazel Yen, “Women in Security: Building a Female InfoSec Community in Korea, Japan, and Taiwan”, BlackHatUSA 2019

Women-Only CTF Workshop

Talk about Asian Female InfoSec Community
Questions?

Asuka Nakajima
E-mail: asuka.nakajima.db@hco.ntt.co.jp
Twitter: @AsuNa_jp