Automatic Generation of Data-Oriented Exploits

Hong Hu, Zheng Leong Chua, Sendroiu Adrian, Prateek Saxena, Zhenkai Liang

National University of Singapore

USENIX Security Symposium 2015, Washington, D.C., USA
Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
  - heap spray / JIT spray
Control Flow Attacks Are Getting Harder

• State-of-the-art exploits
  – Code injection
    • heap spray / JIT spray
  – Code reuse
    • ret2libc, ROP
Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
    - heap spray / JIT spray
  - Code reuse
    - ret2libc, ROP

- Defenses
  - Data Execution Prevention
Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
    - heap spray / JIT spray
  - Code reuse
    - ret2libc, ROP

- Defenses
  - Data Execution Prevention
  - Control Flow Integrity
Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
    - heap spray / JIT spray
  - Code reuse
    - ret2libc, ROP
- Defenses
  - Data Execution Prevention
  - Control Flow Integrity
- control-flow bending
- Stat-of-the-art exploits from memory errors
  - Code injection (e.g., heap spray / JIT spray)
  - Code reuse (e.g., rel2libc, ROP)
- Defenses
  - DEP, CFI, ASLR
  - Block control flow hijacking, in principle
DATA PLANE

- Stat-of-the-Art
- Code

CONTROL PLANE

- Defenses
  - DEP, CFI, ASLR
  - Block control flow hijacking in principle
Data-Oriented Exploits

• State-of-the-art: Corrupt security-critical data
  – leave control flow as the same
  – Exhibit “significant” damage
Data-Oriented Exploits

• State-of-the-art: Corrupt security-critical data
  – leave control flow as the same
  – Exhibit “significant” damage

```c
// set root privilege
seteuid(0);
......
// set normal user privilege
seteuid(pw->pw_uid);
// execute user's command
```

Wu-ftp`d setuid` operation*

Data-Oriented Exploits

• State-of-the-art: Corrupt security-critical data
  – leave control flow as the same
  – Exhibit “significant” damage

// set root privilege
seteuid(0);
......
// set normal user privilege
seteuid(pw->pw_uid);
// execute user's command

//0x1D4, 0x1E4 or 0x1F4 in JScript 9,
//0x188 or 0x184 in JScript 5.8,
safemode = *(DWORD *)(jsobj + 0x188);
if(safemode & 0xB == 0 ) {
  Turn_on_God_Mode();
}

Wu-ftp setuid operation*

IE SafeMode Bypass+

+ Yang Yu. Write Once, Pwn Anywhere. In Black Hat USA 2014
Data-Oriented Exploits

- State-of-the-art: Corrupt security-critical data
  - leave control flow as the same
  - Exhibit “significant” damage
 Contributions

• New class of Data-Oriented Exploits
  – Reuses existing *data flows* in normal execution
  – Agnostic to CFI, DEP and often ASLR
Contributions

• New class of Data-Oriented Exploits
  – Reuses existing *data flows* in normal execution
  – Agnostic to CFI, DEP and often ASLR
Contributions

• New class of Data-Oriented Exploits
  – Reuses existing *data flows* in normal execution
  – Agnostic to CFI, DEP and often ASLR
Contributions

• New class of Data-Oriented Exploits
  – Reuses existing *data flows* in normal execution
  – Agnostic to CFI, DEP and often ASLR

• Data Flow Stitching
  – Systematic search for data-oriented exploits
  – Works on binary directly
Contributions

• New class of Data-Oriented Exploits
  – Reuses existing *data flows* in normal execution
  – Agnostic to CFI, DEP and often ASLR

• Data Flow Stitching
  – Systematic search for data-oriented exploits
  – Works on binary directly

• Results
  – Concrete exploits on real web/file servers
  – 19 exploits (16 new) from 8 vulnerabilities
Motivating Example

- SSL-enabled web server

```c
int server() {
    char *userInput, *fileName;
    char *privKey, *result, output[BUFSIZE];
    char fullPath[BUFSIZE]="/path/to/root/";

    privKey=loadPrivKey("/path/to/privKey");
    GetConnection(privKey, ...);
    userInput = read_socket();
    if (checkInput(userInput)) {
        fileName = getFileName(userInput);
        strcat(fullPath, fileName);
        result = retrieve(fullPath);
        sprintf(output, "%s:%s", fileName, result);
        sendOut(output);
    }
}
```
Motivating Example

• SSL-enabled web server

```c
int server() {
    char *userInput, *fileName;
    char *privKey, *result, output[BUFSIZE];
    char fullPath[BUFSIZE] = "/path/to/root/";

    privKey = loadPrivKey("/path/to/privKey");
    GetConnection(privKey, ...);
    userInput = read_socket();

    if (checkInput(userInput)) {
        fileName = getFileName(userInput);
        strcat(fullPath, fileName);
        result = retrieve(fullPath);
        sprintf(output, "%s:%s", fileName, result);
        sendOut(output);
    }
}
```
Motivating Example

- SSL-enabled web server

```c
int server() {
    char *userInput, *fileName;
    char *privKey, *result, output[BUFSIZE];
    char fullPath[BUFSIZE]="/path/to/root/";

    privKey=loadPrivKey("/path/to/privKey");
    GetConnection(privKey, ...);

    userInput = read_socket();
    if (checkInput(userInput)) {
        fileName = getFileName(userInput);
        strcat(fullPath, fileName);
        result = retrieve(fullPath);
        sprintf(output, "%s:%s", fileName, result);
        sendOut(output);
    }
}
```
Motivating Example

• SSL-enabled web server

```c
int server() {
    char *userInput, *fileName;
    char *privKey, *result, output[BUFSIZE];
    char fullPath[BUFSIZE]="/path/to/root/";

    privKey=loadPrivKey("/path/to/privKey");
    GetConnection(privKey, ...);
    userInput = read_socket();
    if (checkInput(userInput)) {
        fileName = getFileName(userInput);
        strcat(fullPath, fileName);
        result = retrieve(fullPath);
        sprintf(output, "%s:%s", fileName, result);
        sendOut(output);
    }

    return 0;
}
```

Motivating Example

- SSL-enabled web server

```c
int server() {
    char *userInput, *fileName;
    char *privKey, *result, output[BUFSIZE];
    char fullPath[BUFSIZE] = "/path/to/root/";

    privKey = loadPrivKey("/path/to/privKey");
    GetConnection(privKey, ...);
    userInput = read_socket();

    if (checkInput(userInput)) {
        fileName = getFileName(userInput);
        strcat(fullPath, fileName);
        result = retrieve(fullPath);
        sprintf(output, "%s:%s", fileName, result);
        sendOut(output);
    }
}
```
Data-Flow Stitching

• Manipulate data flows for exploits
• Enables systematic way to search for exploits
  – Input: binary & error-exhibiting input
  – Output: data-oriented exploits

• Goal:
  – Information Leakage (e.g., password, keys)
  – Privilege Escalation (e.g., setuid, access priv. files)

• Constraints:
  – Keep the control-flow same
  – Prevent abrupt termination
  – No knowledge of randomized values (CFI tags, ASLR addresses)
Challenges

• Time-consuming search
  – The search-space: Cartesian product $\mid$SrcFlow$\mid \times \mid$TgtFlow$\mid$
  – Heavy analysis for each candidate

• Our solution:
  – Filter out candidates with memory error influence
  – Use an SMT solver to verify candidates
Single-Edge Stitch

- Corrupt data vertex

```c
struct passwd {uid_t pw_uid; ... } pw;
...
int uid = getuid();
pw->pw_uid = uid;
printf(...); //format string error
...
seteuid(0);  //set root uid
...
seteuid(pw->pw_uid); //set normal uid
```
Single-Edge Stitch

• Corrupt data vertex

1 struct passwd {uid_t pw_uid; ... } pw;
2 ...
3 int uid = getuid();
4 pw->pw_uid = uid;
5 printf(...); //format string error
6 ...
7 seteuid(0); //set root uid
8 ...
9 seteuid(pw->pw_uid); //set normal uid
10 ...

• 2D-DFG
Single-Edge Stitch

- Corrupt data vertex

```c
1 struct passwd {uid_t pw_uid; ... } pw;
2 ...
3 int uid = getuid();
4 pw->pw_uid = uid;
5 printf(...); //format string error
6 ...
7 seteuid(0); //set root uid
8 ...
9 seteuid(pw->pw_uid); //set normal uid
10 ...
```
Single-Edge Stitch

- Corrupt data vertex

```
1 struct passwd {uid_t pw_uid; ... } pw;
2 ...
3 int uid = getuid();
4 pw->pw_uid = uid;
5 printf(...);  // format string error
6 ...
7 seteuid(0);  // set root uid
8 ...
9 seteuid(pw->pw_uid);  // set normal uid
10 ...
```
Single-Edge Stitch

- Corrupt data vertex

```c
1 struct passwd {uid_t pw_uid; ... } pw;
2 ...
3 int uid = getuid();
4 pw->pw_uid = uid;
5 printf(...); //format string error
6 ...
7 seteuid(0); //set root uid
8 ...
9 seteuid(pw->pw_uid); //set normal uid
10 ...
```
Single-Edge Stitch

- Corrupt data vertex

1 struct passwd {uid_t pw_uid; ... } pw;
2 ...
3 int uid = getuid();
4 pw->pw_uid = uid;
5 printf(...); //format string error
6 ...
7 seteuid(0); //set root uid
8 ...
9 seteuid(pw->pw_uid); //set normal uid
10 ...
Pointer Stitch

• Corrupt pointers to connect data flows
  – Pointers decide data movement direction
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction
**Pointer Stitch**

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction

![Diagram of Pointer Stitch](image)
**Pointer Stitch**

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction
**Pointer Stitch**

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction

Diagram:
- Addresses: &arg, &pw
- Source flow: 0 to 0
- Target flow: 0 to b2
- Time: 0 to 9
- Data: 100
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction
Pointer Stitch

• Corrupt pointers to connect data flows
  – Pointers decide data movement direction
• Corrupt pointers to connect data flows
  – Pointers decide data movement direction
Pointer Stitch

- Pointer Stitch corrupts pointer $vp$
  - $*(vp) \rightarrow$ target / source vertex

![Diagram of Pointer Stitch](image-url)
Pointer Stitch

• Pointer Stitch corrupts pointer $vp$
  - $*(vp)$ --- target / source vertex

![Diagram of Pointer Stitch](image-url)
Pointer Stitch

- Pointer Stitch corrupts pointer $vp$
  - $*(vp)$ $\rightarrow$ target / source vertex
**Pointer Stitch**

- Pointer Stitch corrupts pointer \( vp \)
  - \( *vp \) --> target / source vertex

![Diagram showing pointer stitch with nodes and arrows]

- Address
- &pw
- a1
- &arg
- time
- Target flow

100

b2

a1

100
• Pointer Stitch corrupts pointer $vp$
  
  – $*(vp)$ --> target / source vertex
• Pointer Stitch corrupts pointer \( vp \)
  
  – \( *(vp) \) ---> target / source vertex
• Pointer Stitch corrupts pointer $vp$
  - $*(vp) \rightarrow$ target / source vertex

Diagram:
- Vertices labeled with addresses: a1, b2, 0
- Edges show changes over time:
  - From a1 to b2
  - From b2 to 0
  - From 0 to target flow
- Time axis from 0 to 100
More Ways of Stitches

• 2-level stitch corrupts pointer $vp_2$
  
  – $*(*(vp_2))$ --- $*(vp)$ --- target / source vertex
More Ways of Stitches

• 2-level stitch corrupts pointer $vp_2$
  – $(*(*(vp_2))) \rightarrow *(vp) \rightarrow$ target / source vertex

• N-level stitch corrupts pointer $vp_N$
  – $(*(*(...(*(vp_N)...))) \rightarrow$ target / source vertex
  – Recursively invoke pointer stitch N times
  – Stitch Alignment
    • $vp_N \rightarrow vp'_N$ so that $(*(*(...(*(vp'_N)...)))$ is the source / target vertex
More Ways of Stitches

• 2-level stitch corrupts pointer $vp_2$
  – $*(*(vp_2)) \rightarrow *(vp) \rightarrow$ target / source vertex

• N-level stitch corrupts pointer $vp_N$
  – $*(...(*(vp_N)...)) \rightarrow$ target / source vertex
  – Recursively invoke pointer stitch N times
  – Stitch Alignment
    • $vp_N \rightarrow vp_N'$ so that $*(...(*(vp_N')...))$ is the source / target vertex

• Multi-flow stitching
  – Intermediate data flows
  – Source flow -> flow 1 -> flow 2 -> ... -> Target flow
Defeat ASLR --- Address Reuse

- Partial reuse: offset is fixed

```
//attackers control %eax
mov (%esi,%eax,4), %ebx
mov %ecx, (%edi,%eax,4)
```
Defeat ASLR --- Address Reuse

• Partial reuse: offset is fixed

  //attackers control %eax
  mov (%esi, %eax, 4), %ebx
  mov %ecx, (%edi, %eax, 4)

• Complete reuse:
  – randomized address in memory

  //attacker controls %eax
  mov (%esi, %eax, 4), %ebx
  mov %ecx, (%ebx)
  mov (%ebx), %ecx
Defeat ASLR --- Address Reuse

- Partial reuse: offset is fixed

  ```
  //attackers control %eax
  mov (%esi, %eax, 4), %ebx
  mov %ecx, (%edi, %eax, 4)
  ```

- Complete reuse:
  - randomized address in memory

  ```
  //attacker controls %eax
  mov (%esi, %eax, 4), %ebx
  mov %ecx, (%ebx)
  mov (%ebx), %ecx
  ```
Defeat ASLR --- Address Reuse

- Partial reuse: offset is fixed
  
  ```
  //attackers control %eax
  mov (%esi,%eax,4), %ebx
  mov %ecx, (%edi,%eax,4)
  ```

- Complete reuse:
  - randomized address in memory

  ```
  //attacker controls %eax
  mov (%esi, %eax, 4), %ebx
  mov %ecx, (%ebx)
  mov (%ebx), %ecx
  ```
Defeat ASLR --- Address Reuse

- Partial reuse: offset is fixed

```c
//attackers control %eax
mov (%esi, %eax, 4), %ebx
mov %ecx, (%edi, %eax, 4)
```

- Complete reuse:
  - randomized address in memory

```c
//attackers control %eax
mov (%esi, %eax, 4), %ebx
mov %ecx, (%edi, %eax, 4)
```

```c
mov %ecx, (%ebx)
```
Stitch with ASLR

• Target deterministic addresses
  – non-PIE binaries on Linux
Stitch with ASLR

• Target deterministic addresses
  – non-PIE binaries on Linux
  – msvcr71.dll, hxds.dll on Windows
FlowStitch

error-exhibiting

benign
FlowStitch

error-exhibiting

benign

error-exhibiting trace

benign trace

1 1 1 1 0 0 0 0

0 0 0 0 1 1 1 1
FlowStitch

- Error-exhibiting
- Benign

Constraints, influence
Imp. data, data flows
FlowStitch

error-exhibiting

benign

error-exhibiting trace

benign trace

constraints, influence

imp. data, data flows

Data-Flow Stitching

candidate exploits
FlowStitch
FlowStitch

error-exhibiting

constraints, influence

benign

imp. data, data flows

SMT Solver

candidate exploits

DOE

error-exhibiting trace

benign trace

Data-Flow Stitching
## Evaluation --- Generated Exploits

<table>
<thead>
<tr>
<th>ID</th>
<th>Vul. bin</th>
<th>Vulnerability</th>
<th>Data-Oriented Exploits</th>
<th>ASLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2013-2028</td>
<td>nginx</td>
<td>Stack bof</td>
<td>$L_0$: private key</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: http root dir</td>
<td></td>
</tr>
<tr>
<td>CVE-2012-0809</td>
<td>sudo</td>
<td>Format string</td>
<td>$M_0$: user id</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2009-4769</td>
<td>httpdx</td>
<td>Format string</td>
<td>$L_0$: admin’s passwd</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: admin’s passwd</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: anon.’s permission</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_2$: anon.’s root dir</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_3$: CGI root dir</td>
<td>✓</td>
</tr>
<tr>
<td>bugtraq ID: 41956</td>
<td>orzhttpd</td>
<td>Format string</td>
<td>$L_0$: randomized addr</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: http root dir</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2002-1496</td>
<td>nullhttpd</td>
<td>Heap overflow</td>
<td>$M_0$: http root dir</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: CGI root dir</td>
<td></td>
</tr>
<tr>
<td>CVE-2001-0820</td>
<td>ghttpd</td>
<td>Stack bof</td>
<td>$M_0$: CGI root dir</td>
<td></td>
</tr>
<tr>
<td>CVE-2001-0144</td>
<td>SSHD</td>
<td>integer overflow</td>
<td>$L_0$: root passwd hash</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: user id</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: authenticated flag</td>
<td></td>
</tr>
<tr>
<td>CVE-2000-0573</td>
<td>wu-ftpd</td>
<td>Format string</td>
<td>$M_0$: user id (single-edge)</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: user id (pointer stitch)</td>
<td>✓</td>
</tr>
</tbody>
</table>

* CVEs discussed in Shuo Chen’s work [1]
## Evaluation --- Generated Exploits

<table>
<thead>
<tr>
<th>ID</th>
<th>Vul. bin</th>
<th>Vulnerability</th>
<th>Data-Oriented Exploits</th>
<th>ASLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2013-2028</td>
<td>nginx</td>
<td>Stack bof</td>
<td>( L_0: \text{private key} ) ( M_0: \text{http root dir} )</td>
<td></td>
</tr>
<tr>
<td>CVE-2012-0809</td>
<td>sudo</td>
<td>Format string</td>
<td>( M_0: \text{user id} ) ( \checkmark )</td>
<td></td>
</tr>
<tr>
<td>CVE-2009-4769</td>
<td>httpdx</td>
<td>Format string</td>
<td>( L_0: \text{admin’s passwd} ) ( M_0: \text{admin’s passwd} ) ( M_1: \text{anon.’s permission} ) ( M_2: \text{anon.’s root dir} ) ( M_3: \text{CGI root dir} )</td>
<td></td>
</tr>
<tr>
<td>bugtraq ID: 41956</td>
<td>orzhttpd</td>
<td>Format string</td>
<td>( L_0: \text{randomized addr} ) ( M_0: \text{http root dir} ) ( \checkmark )</td>
<td></td>
</tr>
<tr>
<td>CVE-2002-1496 *</td>
<td>nullhttpd</td>
<td>Heap overflow</td>
<td>( M_0: \text{http root dir} ) ( M_1: \text{CGI root dir} )</td>
<td></td>
</tr>
<tr>
<td>CVE-2001-0820 *</td>
<td>ghttpd</td>
<td>Stack bof</td>
<td>( M_0: \text{CGI root dir} )</td>
<td></td>
</tr>
<tr>
<td>CVE-2001-0144 *</td>
<td>SSHD</td>
<td>integer overflow</td>
<td>( L_0: \text{root passwd hash} ) ( M_0: \text{user id} ) ( M_1: \text{authenticated flag} )</td>
<td></td>
</tr>
<tr>
<td>CVE-2000-0573 *</td>
<td>wu-ftp</td>
<td>Format string</td>
<td>( L_0: \text{env variables} ) ( M_0: \text{user id (single-edge)} ) ( \checkmark ) ( M_1: \text{user id (pointer stitch)} ) ( \checkmark )</td>
<td></td>
</tr>
</tbody>
</table>

* CVEs discussed in Shuo Chen’s work [1]
### Evaluation --- Generated Exploits

<table>
<thead>
<tr>
<th>ID</th>
<th>Vul. bin</th>
<th>Vulnerability</th>
<th>Data-Oriented Exploits</th>
<th>ASLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2013-2028</td>
<td>nginx</td>
<td>Stack bof</td>
<td>$L_0$: private key</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: http root dir</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: private key</td>
<td></td>
</tr>
<tr>
<td>CVE-2012-0809</td>
<td>sudo</td>
<td>Format string</td>
<td>$L_0$: admin’s passwd</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: user id</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: anon.’s permission</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_2$: anon.’s root dir</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_3$: CGI root dir</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2009-4769</td>
<td>httpdx</td>
<td>Format string</td>
<td>$L_0$: randomized addr</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: http root dir</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: anon.’s permission</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_2$: anon.’s root dir</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_3$: CGI root dir</td>
<td>✓</td>
</tr>
<tr>
<td>bugtraq ID: 41956</td>
<td>orzhttpd</td>
<td>Format string</td>
<td>$L_0$: randomized addr</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: http root dir</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2002-1496 *</td>
<td>nullhttpd</td>
<td>Heap overflow</td>
<td>$M_0$: http root dir</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: CGI root dir</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2001-0820 *</td>
<td>ghttpd</td>
<td>Stack bof</td>
<td>$M_0$: CGI root dir</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2001-0144 *</td>
<td>SSHD</td>
<td>integer overflow</td>
<td>$M_0$: user id</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: authenticated flag</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2000-0573 *</td>
<td>wu-ftpd</td>
<td>Format string</td>
<td>$M_0$: user id (single-edge)</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1$: user id (pointer stitch)</td>
<td>✓</td>
</tr>
</tbody>
</table>

* CVEs discussed in Shuo Chen’s work [1]

- 19 exploits
### Evaluation --- Generated Exploits

<table>
<thead>
<tr>
<th>ID</th>
<th>Vul. bin</th>
<th>Vulnerability</th>
<th>Data-Oriented Exploits</th>
<th>ASLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2013-2028</td>
<td>nginx</td>
<td>Stack bof</td>
<td>L₀: private key</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₀: http root dir</td>
<td></td>
</tr>
<tr>
<td>CVE-2012-0809</td>
<td>sudo</td>
<td>Format string</td>
<td>L₀: admin’s passwd</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₀: http root dir</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₀: user id</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2009-4769</td>
<td>httpdx</td>
<td>Format string</td>
<td>L₀: randomized addr</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₀: http root dir</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₀: admin;s passwd</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₁: anon.’s permission</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₂: anon.’s root dir</td>
<td>✔</td>
</tr>
<tr>
<td>bugtraq ID: 41956</td>
<td>orzhttpd</td>
<td>Format string</td>
<td>L₀: randomized addr</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₀: http root dir</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2002-1496 *</td>
<td>nullhttpd</td>
<td>Heap overflow</td>
<td>M₀: http root dir</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2001-0820 *</td>
<td>ghttpd</td>
<td>Stack bof</td>
<td>M₀: CGI root dir</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2001-0144 *</td>
<td>SSHD</td>
<td>integer overflow</td>
<td>M₀: root passwd hash</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₀: user id</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₁: authenticated flag</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2000-0573 *</td>
<td>wu-ftpd</td>
<td>Format string</td>
<td>M₀: user id (single-edge)</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₁: user id (pointer stitch)</td>
<td>✔</td>
</tr>
</tbody>
</table>

* CVEs discussed in Shuo Chen’s work [1]

- 19 exploits
- 16 prev. unknown
### Evaluation --- Generated Exploits

<table>
<thead>
<tr>
<th>ID</th>
<th>Vul. bin</th>
<th>Vulnerability</th>
<th>Data-Oriented Exploits</th>
<th>ASLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2013-2028</td>
<td>nginx</td>
<td>Stack bof</td>
<td>$L_0: \text{private key}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0: \text{http root dir}$</td>
<td></td>
</tr>
<tr>
<td>CVE-2012-0809</td>
<td>sudo</td>
<td>Format string</td>
<td>$L_0: \text{admin's passwd}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0: \text{admin};\text{passwd}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1: \text{anon.'s permission}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_2: \text{anon.'s root dir}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_3: \text{CGI root dir}$</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2009-4769</td>
<td>httpdx</td>
<td>Format string</td>
<td>$L_0: \text{admin's passwd}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0: \text{admin};\text{passwd}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1: \text{anon.'s permission}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_2: \text{anon.'s root dir}$</td>
<td>✓</td>
</tr>
<tr>
<td>bugtraq ID: 41956</td>
<td>orzhttpd</td>
<td>Format string</td>
<td>$L_0: \text{randomized addr}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0: \text{http root dir}$</td>
<td>✓</td>
</tr>
<tr>
<td>CVE-2002-1496 *</td>
<td>nullhttpd</td>
<td>Heap overflow</td>
<td>$M_0: \text{http root dir}$</td>
<td></td>
</tr>
<tr>
<td>CVE-2001-0820 *</td>
<td>ghttpd</td>
<td>Stack bof</td>
<td>$M_0: \text{CGI root dir}$</td>
<td></td>
</tr>
<tr>
<td>CVE-2001-0144 *</td>
<td>SSHD</td>
<td>integer overflow</td>
<td>$L_0: \text{root passwd hash}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0: \text{user id}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1: \text{authenticated flag}$</td>
<td></td>
</tr>
<tr>
<td>CVE-2000-0573 *</td>
<td>wu-ftpd</td>
<td>Format string</td>
<td>$L_0: \text{env variables}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0: \text{user id (single-edge)}$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_1: \text{user id (pointer stitch)}$</td>
<td>✓</td>
</tr>
</tbody>
</table>

- 19 exploits
- 16 prev. unknown
- 7 advanced stitch
  - 2-level stitch

* CVEs discussed in Shuo Chen’s work [1]
## Evaluation --- Generated Exploits

<table>
<thead>
<tr>
<th>ID</th>
<th>Vul. bin</th>
<th>Vulnerability</th>
<th>Data-Oriented Exploits</th>
<th>ASLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2013-2028</td>
<td>nginx</td>
<td>Stack bof</td>
<td>L₀: private key</td>
<td></td>
</tr>
<tr>
<td>CVE-2012-0809</td>
<td>sudo</td>
<td>Format string</td>
<td>M₀: http root dir</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₁: user id</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2009-4769</td>
<td>httpdx</td>
<td>Format string</td>
<td>M₀: admin’s passwd</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₁: anon.’s permission</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₂: anon.’s root dir</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₃: CGI root dir</td>
<td>✔</td>
</tr>
<tr>
<td>bugtraq ID: 41956</td>
<td>orzhttpd</td>
<td>Format string</td>
<td>L₀: randomized addr</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₀: http root dir</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2002-1496 *</td>
<td>nullhttpd</td>
<td>Heap overflow</td>
<td>M₀: http root dir</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₁: CGI root dir</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2001-0820 *</td>
<td>ghtpd</td>
<td>Stack bof</td>
<td>M₀: CGI root dir</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2001-0144 *</td>
<td>SSHD</td>
<td>integer overflow</td>
<td>M₀: user id (single-edge)</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₁: authenticated flag</td>
<td>✔</td>
</tr>
<tr>
<td>CVE-2000-0573 *</td>
<td>wu-ftp</td>
<td>Format string</td>
<td>M₀: env variables</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>M₁: user id (single-edge)</td>
<td>✔</td>
</tr>
</tbody>
</table>

* CVEs discussed in Shuo Chen’s work [1]

- 19 exploits
- 16 prev. unknown
- 7 advanced stitch
  - 2-level stitch
- 10 bypass ASLR
  - 8 fixed addresses
  - 2 address reuse
Evaluation --- Performance

- 6.5 min/exploit
- Slice takes long
  - faster version is available (binary version)
Case Study – 2-Level Stitch

- **ghttpd** web server: stack buffer overflow

```c
//serveconnection():
    char *ptr; //URL pointer
    //esi is allocated for it
1: if(strstr(ptr,"/.."))
     reject the request;
2: log(...);
3: exec(ptr);
```

<table>
<thead>
<tr>
<th>Assembly of log(...):</th>
<th>Assembly of line 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>push %ebp</td>
<td>push %esi</td>
</tr>
<tr>
<td>push %esi</td>
<td>push %esi</td>
</tr>
<tr>
<td>// stack overflow</td>
<td>// stack overflow</td>
</tr>
<tr>
<td>pop %esi</td>
<td>pop %ebp</td>
</tr>
<tr>
<td></td>
<td>ret</td>
</tr>
</tbody>
</table>

- Previous exploit\(^1\)
  - Corrupt pointer ptr: *(ptr) -> url
Case Study – 2-Level Stitch

• **httpd** web server: stack buffer overflow

<table>
<thead>
<tr>
<th><code>//serveconnection()</code>:</th>
<th>Assembly of log(...)</th>
<th>Assembly of line 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *ptr; //URL pointer</td>
<td>push %ebp</td>
<td>mov -0xc(%ebp), %esi</td>
</tr>
<tr>
<td>//esi is allocated for it</td>
<td>push %esi</td>
<td>push %esi</td>
</tr>
<tr>
<td>1: if strstr(ptr,&quot;../&quot;)</td>
<td>// stack overflow</td>
<td>...</td>
</tr>
<tr>
<td>reject the request;</td>
<td>pop %esi</td>
<td>call <a href="mailto:exec@plt">exec@plt</a></td>
</tr>
<tr>
<td>2: log(...)</td>
<td>pop %ebp</td>
<td></td>
</tr>
<tr>
<td>3: exec(ptr);</td>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

• Previous exploit[1] does not work any more
  – Corrupt pointer ptr: *(ptr) -> url
# Case Study – 2-Level Stitch

- **`ghhttpd`** web server: stack buffer overflow

```c
//serveconnection():
char *ptr; //URL pointer
//esi is allocated for it
1: if(strstr(ptr,"/.."))
   reject the request;
2: log(...);
3: exec(ptr);
```

**Assembly of log(...):**

- push %ebp
- push %esi
- // stack overflow
- pop %esi
- pop %ebp
- ret

**Assembly of line 3:**

- mov -0xc(%ebp), %esi
- push %esi
- ...
- call <exec@plt>

- Previous exploit\(^1\) does not work any more
  - Corrupt pointer ptr: *(ptr) -> url

- We build a 2-level stitch
  - Corrupt pointer saved ebp: *(*(saved ebp)) -> *ptr -> url
Case Study – Sensitive Data Lifespan

- **SSHD** hashed key info leak
- `getspnam()` in *glibc* gets hashed key (heap copy)
- **SSHD** copies hashed key to local stack (stack copy)
Case Study – Sensitive Data Lifespan

- SSHD hashed key info leak
- `getspnam()` in `glibc` gets hashed key (heap copy)
- SSHD copies hashed key to local stack (stack copy)
  - Overwritten by later usage
Case Study – Sensitive Data Lifespan

• SSHD hashed key info leak
• `getspnam()` in `glibc` gets hashed key (heap copy)
  – `endspent()` in `glibc` releases memory, not clears it!
  – Still alive for stitching
• SSHD copies hashed key to local stack (stack copy)
  – Overwritten by later usage
Case Study – Sensitive Data Lifespan

- **SSHDA** hashed key info leak
- `getspnam()` in `glibc` gets hashed key (heap copy)
  - `endspent()` in `glibc` releases memory, not clears it!
  - Still alive for stitching
- **SSHDA** copies hashed key to local stack (stack copy)
  - Overwritten by later usage
- **Challenging** to make lifespan correct!
Conclusion

• Rich Category: Data-Oriented Exploits
  – Single-edge stitch, Pointer stitch
  – N-level stitch, Multi-flow stitch

• Data Flow Stitching
  – Systematic way to generate data-oriented exploits
  – Agnostic to CFI, DEP and often ASLR

• Automatic construction is feasible
Thanks!

Hong Hu

huhong@comp.nus.edu.sg

http://www.comp.nus.edu.sg/~huhong/