Automatic Generation of Data-Oriented Exploits

Zhenkai Liang

NUS
National University of Singapore
School of Computing
Arms Race in Memory Exploit

• CPU functionality
  – Decode and execute instruction
  – Code vs. Data?

• Trend of memory exploits
  – Buffer overflow
  – Return-to-libc
  – Return-oriented programming (ROP)
Control Flow Attacks Are Getting Harder

- **State-of-the-art exploits**
  - Code injection
    - Buffer overflow/heap spray
  - Code reuse
    - Ret2libc, ROP
- **Defenses**
  - Data Execution Prevention
  - Control Flow Integrity

- **Control-flow bending**
CONTROL PLANE

DATA PLANE

- Stat-of-the-art attacks
- Context
- Call traces

- 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

// 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-ftpd setuid operation

IE SafeMode Bypass

+ Yang Yu. Write Once, Pwn Anywhere. In Black Hat USA 2014
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 $|\text{SrcFlow}| \times |\text{TgtFlow}|$
  – 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;
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 ...
```
- 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 corrupts pointer $vp$
  – $*(vp)$ --- target / source vertex
Pointer Stitch

• Pointer Stitch corrupts pointer $vp$
  – $*vp$ --- target / source vertex
More Ways of Stitches

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

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

- Multi-flow stitching
  - Intermediate data flows
  - Source flow $\rightarrow$ flow 1 $\rightarrow$ flow 2 $\rightarrow$ ... $\rightarrow$ Target flow
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

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

```

```
FlowStitch

- error-exhibiting
- benign

- error-exhibiting trace
- benign trace

- constraints, influence
- imp. data, data flows

- Data-Flow Stitching

- candidate exploits

- SMT Solver

- DOE
### 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>$L_0$: 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>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>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>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>CVE-2000-0573 *</td>
<td>wu-ftp</td>
<td>Format string</td>
<td>$L_0$: env variables</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M_0$: 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

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Assembly</th>
</tr>
</thead>
</table>
| //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

• **SSHDI** hashed key info leak

• `getspnam()` in `glibc` gets hashed key (heap copy)
  – `endspent()` in `glibc` releases memory, not clears it!
  – Still alive for stitching

• **SSHDI** 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