

NCTU Version

Everyday is Zero Day Today is Flash Player

By Lucas Leong

Original version

- 2014 HITCON Enterprise
 - <http://www.slideshare.net/wmliang/everyday-is-zero-day-today-is-flash-player>

Who is the speaker

- NCTU Software Quality Lab
- Trend Micro
- Focus on...
 - Document exploit
 - Antivirus engine
 - CTF

Document exploit

- Used in APT(Advanced Persistent Threat) attack
 - Targeted attack
 - C&C server
 - Social engineering
- Document type
 - PDF
 - **Flash**
 - HTML
 - Office
 - Java
 - Font
 - ...

Today I will talk about ...

- A journey for a complete zero-day exploit nowadays
- Crash it
 - How is the birth of a Flash Player zero-day exploit
- **Debug it**
 - **How to debug Flash Player**
- Exploit it
 - How to take control of it
- Bypass it
 - How many and how to overcome exploit mitigations nowadays

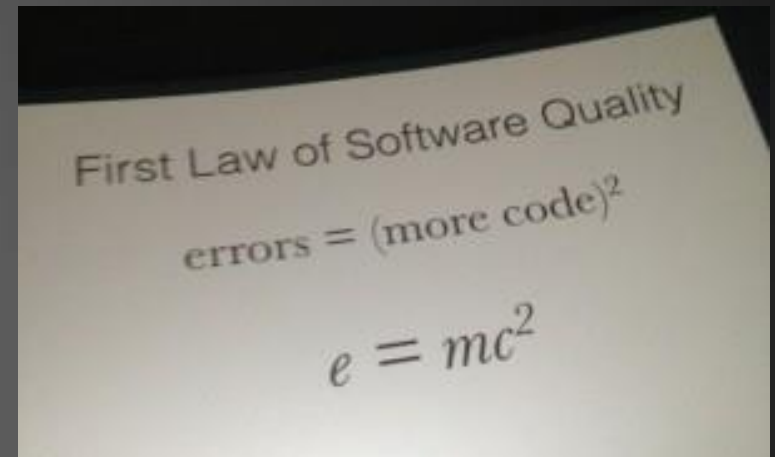
Crash it

Vulnerability review

- Part of my daily work is to review new document exploit CVE and enhance antivirus engine
- Most of Flash Player CVE relate to ActionScript
- Flash Player's regular expression CVE in 2014 H1
 - CVE-2014-0498
 - CVE-2014-0499
 - CVE-2014-0502

Why RE

- Easy to implement RE fuzzer
 - RE syntax is less diverse
- Hard to implement RE engine
 - RE semantic is complicated
 - RE engine == RE compiler
- Flash player maintains a modified-PCRE



What is fuzz testing

Random
Inputs



核四運轉首日出現輕微事故
政府表示為正常能量釋放 台北市市民不必驚慌

連核爆

馬英九總統 宣布今日將出訪美國
桃園國際機場人多擁塞 一度停擺

北市府昨決議 屬輕微運轉事故 將不會進行疏散

行政院決議將懲處 爆料核四事故官員

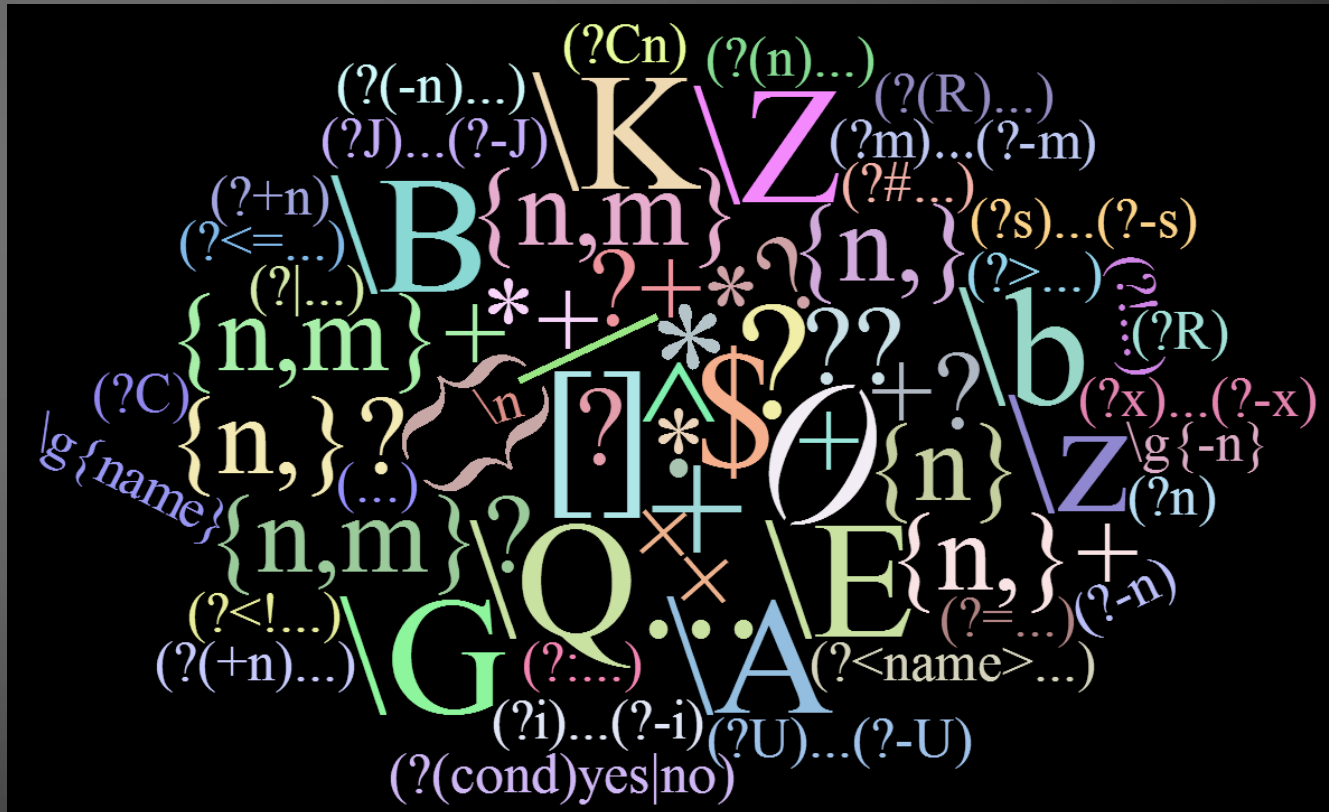
衛生署駁媒體報導 指輻射劑量未超標

力星郵輪 救命星號 目前停靠基隆港 最後搭客乘機起飛 逾時不候

帝飽百坪豪宅 一元起標無底價 請洽 連先生(成交再選擇)

PCRE fuzzing

- List all PCRE syntax

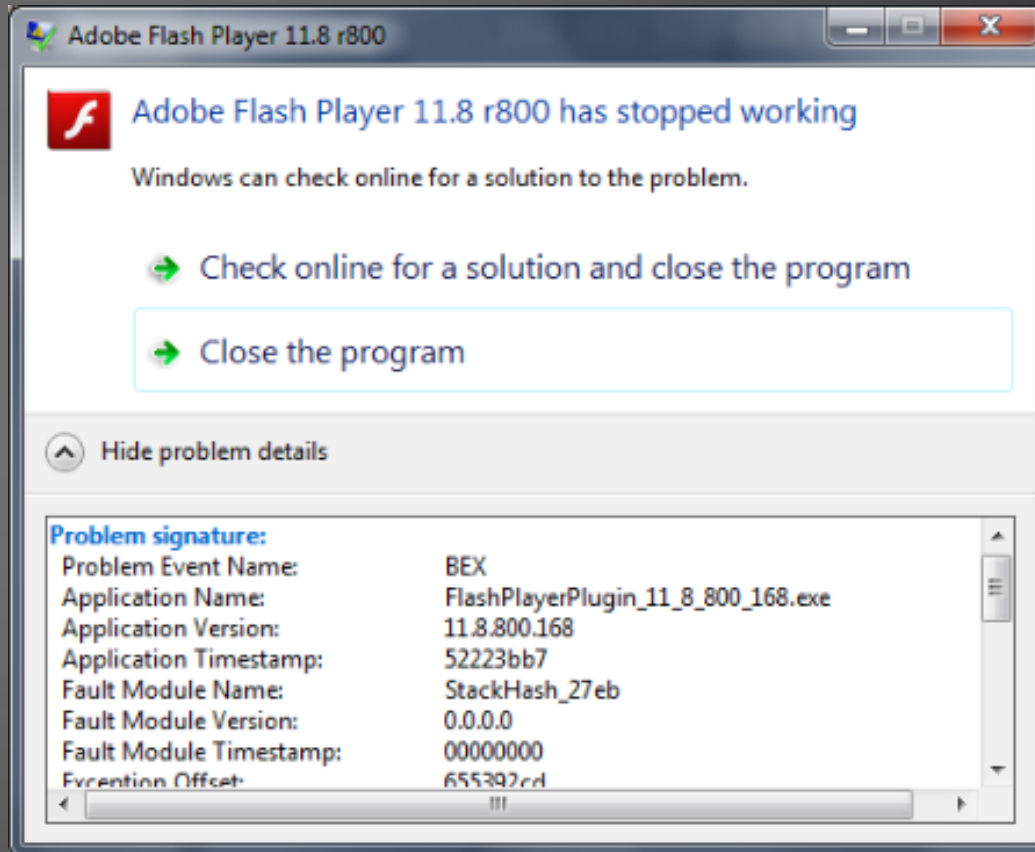


PCRE fuzzing

- Fuzzer in Actionscript

```
while (true) {  
    valid_syntax_re = random_generate();  
    re = new RegExp(valid_syntax_re, "");  
    "WMLIANG".match(re);  
}
```

Just wait for crash



Debug it

Now...

- Generated a set of crashes ideally
- Figure out the root cause manually
 - !exploitable
 - <http://msecdbg.codeplex.com/>

Debug Flash Player

```
71AB4009 B8FF          MOV     EDI,EDI
71AB400A 55           PUSH  ESP
71AB400B 8BEC        MOV     EBP,ESP
71AB400C 83EC 18     SUB     ESP,18
71AB400D 57           PUSH  EDI
71AB400E 8D45 E8     LEA    EAX,DWORD PTR SS:[EBP-16]
71AB400F 50           PUSH  EAX
71AB4010 8D45 EC     LEA    EAX,DWORD PTR SS:[EBP-14]
71AB4011 50           PUSH  EAX
71AB4012 FF15 80400C71 CALL   DWORD PTR DS:[71AC4060]      WS2_32.71AB2C29
71AB4013 58         POP     EAX
71AB4014 8BC7        MOV     EDI,EDI
71AB4015 8BC7        MOV     EAX,EDI
71AB4016 8D45 FC     LEA    EAX,DWORD PTR SS:[EBP-4],EAX
71AB4017 8F25 09010000 JMP     WORD PTR SS:[71AB4014]
71AB4018 FF75 08     PUSH  DWORD PTR SS:[EBP+8]
71AB4019 53         PUSH  ESP
71AB401A 8B45 2E     MOV     EAX,DWORD PTR SS:[71AB2E2E]
71AB401B 8BC7        MOV     EAX,EDI
71AB401C 8D45 F0     LEA    EAX,DWORD PTR SS:[EBP-10],EAX
71AB401D 8F25 0C010000 JMP     WORD PTR SS:[71AB401C]
71AB401E 53         PUSH  ESP
71AB401F 8B45 0C     MOV     EAX,DWORD PTR DS:[EAX*4]
71AB4020 8B70 0C     MOV     ESI,DWORD PTR DS:[EAX*4]
71AB4021 8B70 08     MOV     EDI,DWORD PTR SS:[EBP+3],EDI
71AB4022 8D45 FC     LEA    EAX,DWORD PTR SS:[EBP+4]
71AB4023 50           PUSH  EAX
71AB4024 57           PUSH  EDI
71AB4025 57           PUSH  EDI
71AB4026 57           PUSH  EDI
71AB4027 57           PUSH  EDI
71AB4028 FF75 10     PUSH  DWORD PTR SS:[EBP+10]
71AB4029 FF75 0C     PUSH  DWORD PTR SS:[EBP+8]
71AB402A FF75 08     PUSH  DWORD PTR SS:[EBP+8]
71AB402B FF25 24     CALL   DWORD PTR DS:[ESI+24]
71AB402C 8BD8        MOV     EBX,EBX
71AB402D 8BFB        MOV     EBX,-1
71AB402E 8F25 0C010000 JMP     WORD PTR SS:[EBP-10]
71AB402F 8B4D F0     MOV     ECX,DWORD PTR SS:[EBP-10]
71AB4030 8B4D F0     MOV     ECX,DWORD PTR SS:[EBP-10]
71AB4031 53         PUSH  ESP
71AB4032 5E         POP     ESI
71AB4033 8BC7        MOV     EAX,EDI
71AB4034 5B         POP     EBX
71AB4035 5B         POP     EBX
71AB4036 8F25 0C010000 JMP     WORD PTR SS:[71AB4036]
71AB4037 53         PUSH  ESP
71AB4038 5F         POP     EDI
71AB4039 5F         POP     EDI
71AB403A C2 0C00    RETN   0C
```



Debug Flash Player

- What the ...



Debug Flash Player



Debug Flash Player

- Figure out the mapping of ActionScript and x86 assembly



Debug Flash Player



Open source project

- Tamarin Project
 - Old VM open source project @ 2008
 - Developed by Mozilla and Adobe
 - <http://www-archive.mozilla.org/projects/tamarin/>
- Part of Flash Player
 - No render
 - No memory management
 - Out-of-date

Debug Flash Player

- X86 assembly in debugger

```
• debug074:02488235 db 0
• debug074:02488236 db 0
EAX debug074:02488237 ; -----
EIP • debug074:02488237 push ebp
• debug074:02488238 mov ebp, esp
• debug074:0248823A sub esp, 1B8h
• debug074:02488240 mov [ebp-124h], ebx
• debug074:02488246 mov [ebp-16Ch], esi
• debug074:0248824C mov [ebp-170h], edi
• debug074:02488252 mov esi, [ebp+8]
• debug074:02488255 mov [ebp-184h], esi
• debug074:0248825B mov edx, [ebp+10h]
• debug074:0248825E lea ebx, [ebp-130h]
• debug074:02488264 mov eax, offset unk_17DF810
• debug074:02488269 mov [ebp-180h], eax
• debug074:0248826F mov edi, off_17DF844
• debug074:02488275 mov [ebp-12Ch], esi
• debug074:0248827B mov [ebp-130h], edi
• debug074:02488281 mov off_17DF844, ebx
• debug074:02488287 lea ecx, [ebp-100h]
• debug074:0248828D lea edi, [ebp-120h]
• debug074:02488293 mov eax, dword_17DF834
• debug074:02488298 cmp ebx, eax
```

Debug Flash Player



Open source project

- PCRE
 - <http://www.pcre.org/>

Debug PCRE

- Demo
 - Adobe Flash Player
 - Standalone version
 - PCRE project
 - Adobe Flex SDK
 - mxmhc [.as]

CVE-2014-0559

Root cause

- In PCRE compiler

```
function pcre_compile() {  
    int length;  
    // first-phase for estimating the length of compiled RE  
    compile_regex(NULL, &size);  
    re = malloc(size);  
    // second-phase for compiling RE actually  
    compile_regex(re, NULL);  
}
```

Root cause

- In PCRE compiler

```
function pcre_compile() {  
    int size;  
    compile_regex(NULL, &size); // return size = 0xA0 bytes  
    re = malloc(size);  
    compile_regex(re, NULL); // but overwrite > 0xA0 bytes  
}
```

It's heap overflow !



Exploit it

Exploit in school

- Heap exploit
- ASLR
- DEP

SNOWMEN



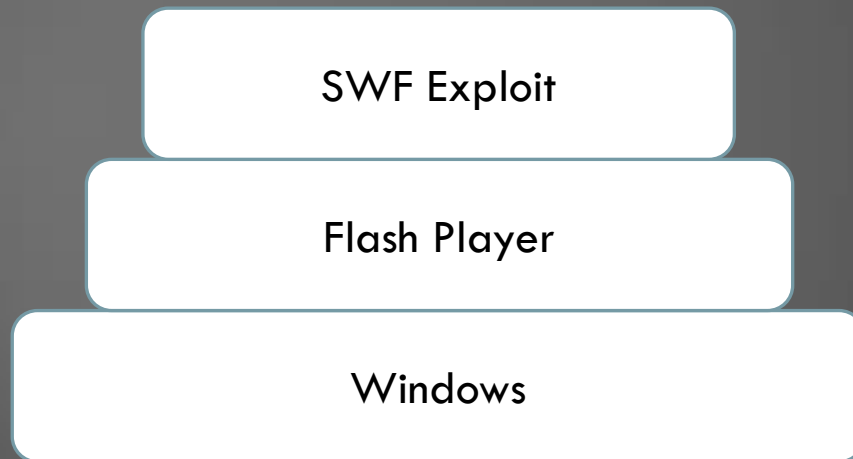
Exploit in real

- Heap exploit in Windows
 - Coalesce unlink overwrite
 - LAL link overwrite
 - Freelist[] attack
 - Heap cache attack
 - LFH FreeEntryOffset overwrite
 - LFH bucket overwrite
 - ...



Exploit in real

- Custom heap management for large application



```
var i:* = new Vector();
```

```
player_malloc()
```

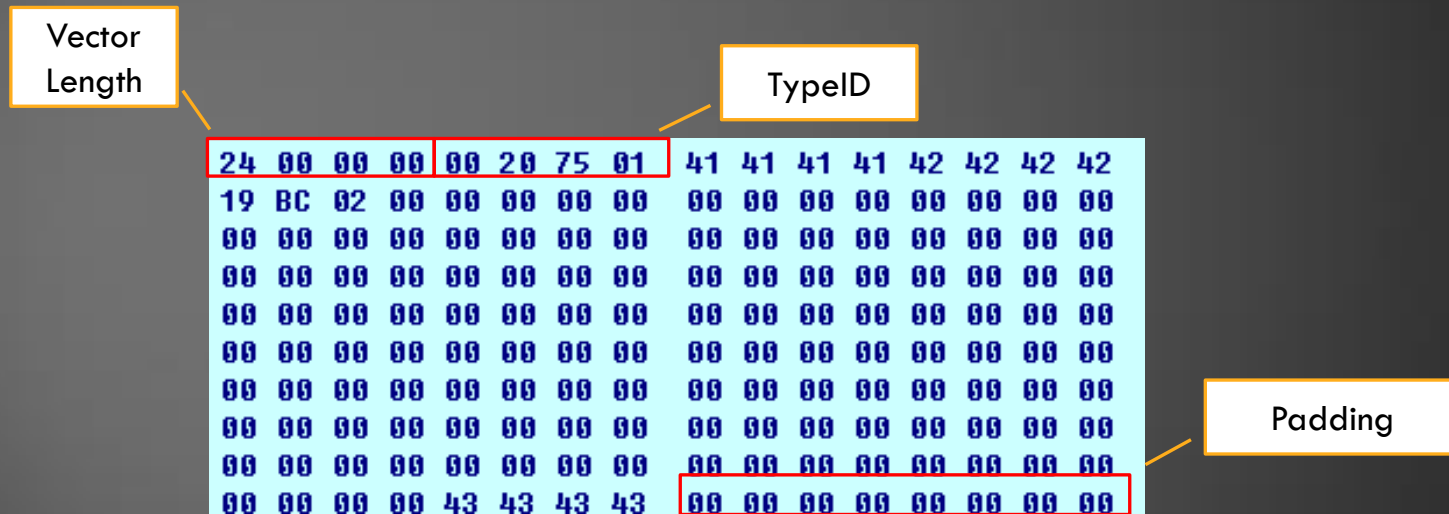
```
RtlAllocateheap()
```

Exploit in real

- Heap exploit
 - Overwrite application/object data
 - Less mitigation
 - Object data in Flash Player, IE, ...
 - Size
 - new Vector(1024)
 - Function pointer (vtable)
 - new Sound().open()
 - Pointers
 - Flags
 - Type ID
 - ...

Application data

- `var i:* = new Vector.<uint>(36);`

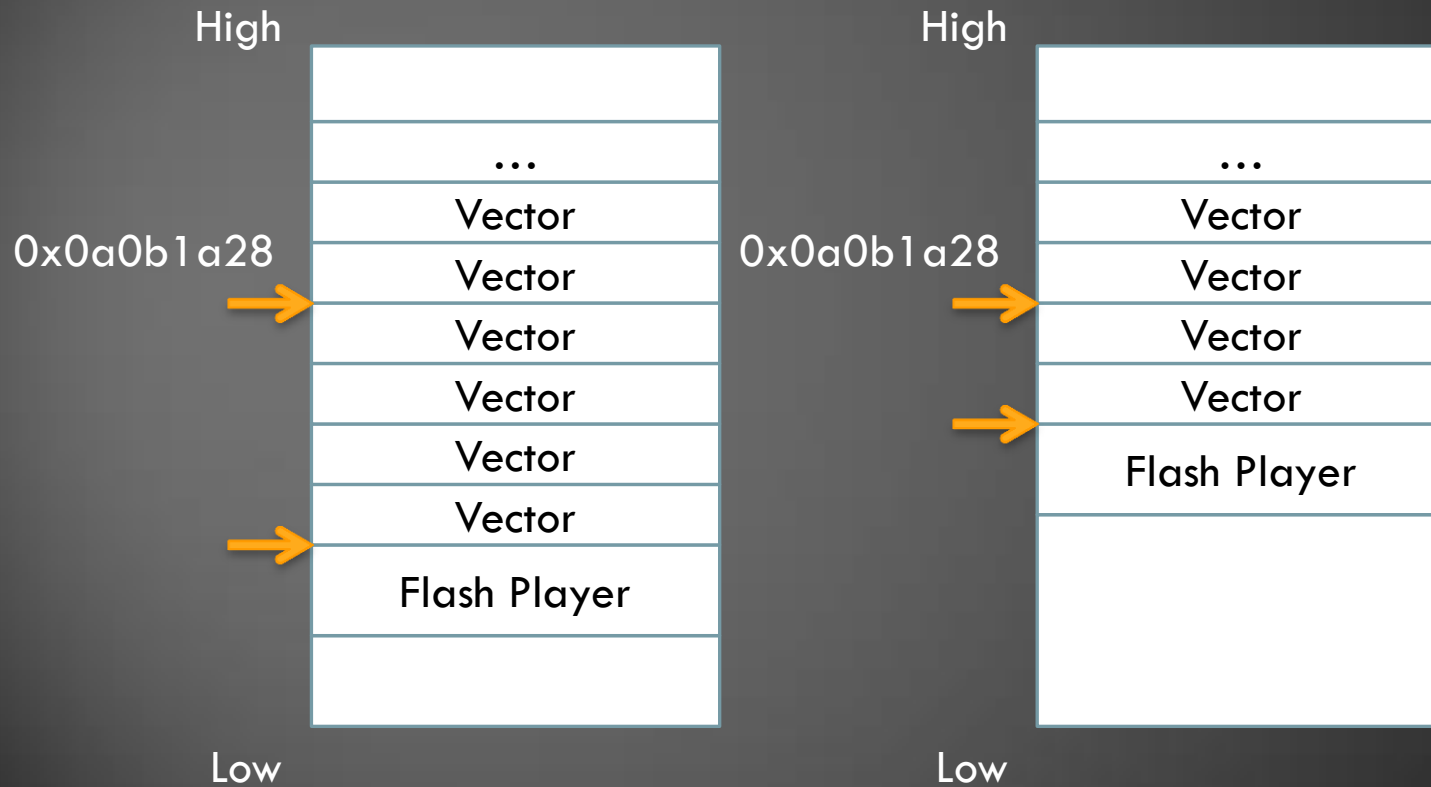


Application data

- How to place the RegExp object before vector object ?



Heap spray



Heap spray

- The objective of heap spray is a workaround to bypass ASLR “ugly”
 - Detectable
 - Unreliable
- Demo
 - Write a ASCII “WMLIANG” at 0x0a0b1a28

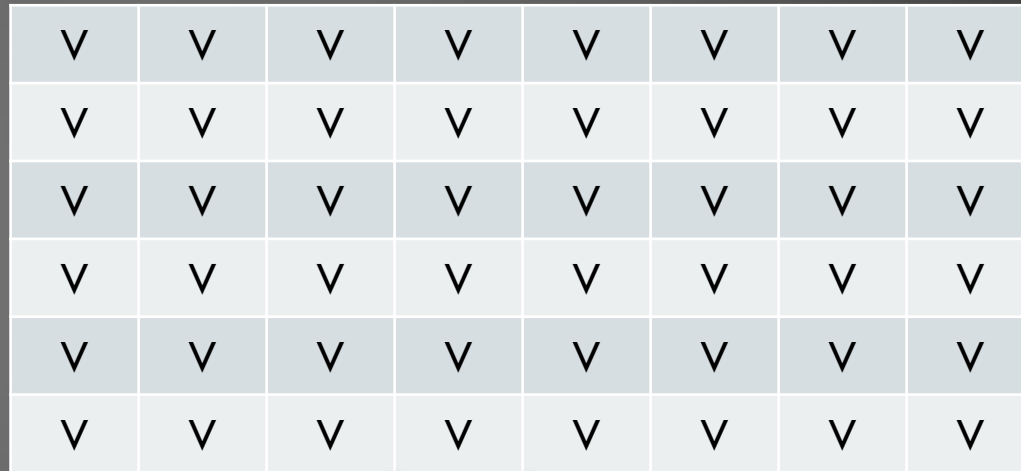
Heap massage/grooming

- The objective of heap massage is to control the memory layout
- Memory allocation is fewer than heap spray

Memory layout

```
while (i < VECTOR_NUM) {  
    s[i] = new Vector.<uint>(36);  
    s[i++][0] = 0x41414141;  
    ...  
}
```

- Allocate Vector objects in memory



Vector length

24	00	00	00	00	20	75	01	41	41	41	41	42	42	42	42
19	BC	02	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	43	43	43	43	00	00	00	00	00	00	00	00

a Vector object with 0xA0 bytes

Memory Layout

- Free some of them

```
while(i < VECTOR_NUM) {  
    if(i % 4 == 2)  
        s[i++] = null;  
}
```

V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V

Vector length

24	00	00	00	00	00	20	75	01	41	41	41	41	42	42	42	42
19	BC	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	43	43	43	43	00	00	00	00	00	00	00	00	00

a Vector object with 0xA0 bytes

```
new RegExp(CVE_2014_0559, "");
```

Memory layout

- Allocate malicious RE object and trigger the overflow

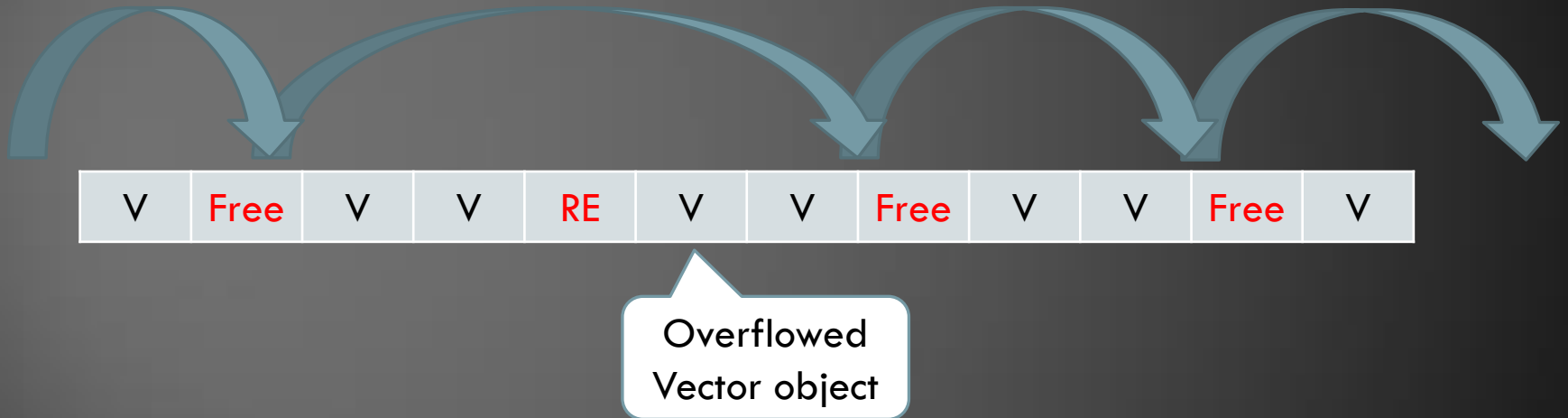
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	RE	V	V	V	Free	V

Vector length

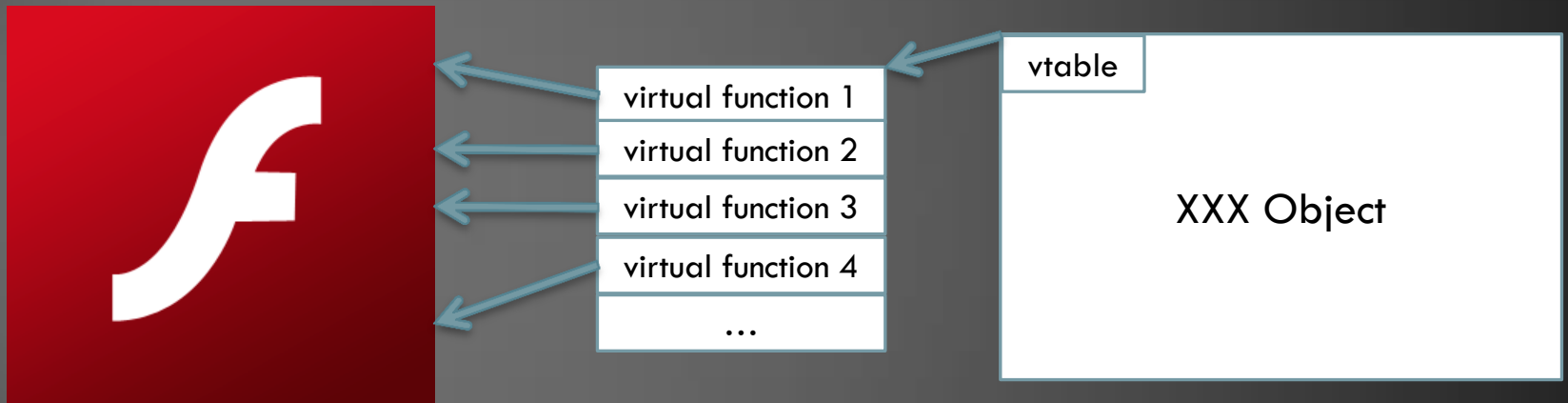
```
00 53 00 05 18 00 53 00 05 18 00 53 00 05 18 00
53 00 05 18 00 54 00 83 18 01 18 00 54 00 91 54
00 97 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 43 43 43 43 00 00 00 00 00 00 00 00
```

a Vector object with 0xA0 bytes

Locate the address



Bypass ASLR



Bypass DEP

- Ret-2-library
 - VirtualAlloc
 - HeapCreate
 - SetProcessDEPPolicy
 - NtSetInformationProcess
 - VirtualProtect
 - WriteProcessMemory
 - ZwProtectVirtualMemory
- Ret-2-shellcode
 - Pop up a calc.exe



Bypass it

Exploit mitigations



DEP

Exploit mitigations



DEP



ASLR

Exploit mitigations



DEP



ASLR



EMET

Exploit mitigations



DEP



ASLR



EMET



**Browser
Sandbox**

Traditional exploit

- If you ignore EMET and Browser sandbox...
 1. Locate the base address of DLLs
 2. Build ROP
 3. Execute shellcode

4.



Disadvantage of ROP

- Platform-dependent
- Easily detected (eg. EMET)
- Cost of time
- Continuous execution

How to exploit without ROP ?

Function1 in Flash Player

```
bool __cdecl sub_10212D7E(int a1)
{
    BOOL v1; // eax@1
    struct _STARTUPINFOA StartupInfo; // [sp+8h] [bp-64h]@1
    struct _PROCESS_INFORMATION ProcessInformation; // [sp+4Ch] [bp-20h]@1
    LPSTR lpCommandLine; // [sp+5Ch] [bp-10h]@1
    bool v6; // [sp+6Bh] [bp-1h]@1

    strcpy((int)&lpCommandLine, (int)"iexplore.exe ");
    strcat(&lpCommandLine, (const char *)a1);
    ProcessInformation.hProcess = 0;
    ProcessInformation.hThread = 0;
    ProcessInformation.dwProcessId = 0;
    ProcessInformation.dwThreadId = 0;
    memset(&StartupInfo.lpReserved, 0, 0x40u);
    StartupInfo.wShowWindow = 5;
    StartupInfo.cb = 68;
    StartupInfo.dwFlags = 1;
    v1 = CreateProcessA(0, lpCommandLine, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
    v6 = v1 != 0;
    if ( v1 )
    {
        CloseHandle(ProcessInformation.hProcess);
        CloseHandle(ProcessInformation.hThread);
    }
    free_str(&lpCommandLine);
    return v6;
}
```

Function1 in Flash Player

```
bool __cdecl sub_10212D7E(int a1)
{
    BOOL v1; // eax@1
    struct _STARTUPINFOA StartupInfo; // [sp+8h] [bp-64h]@1
    struct _PROCESS_INFORMATION ProcessInformation; // [sp+4Ch] [bp-20h]@1
    LPSTR lpCommandLine; // [sp+5Ch] [bp-10h]@1
    bool v6; // [sp+6Bh] [bp-1h]@1

    strcpy((int)&lpCommandLine, (int)"iexplore.exe ");
    strcat(&lpCommandLine, (const char *)a1);
    ProcessInformation.hProcess = 0;
    ProcessInformation.hThread = 0;
    ProcessInformation.dwProcessId = 0;
    ProcessInformation.dwThreadId = 0;
    memset(&StartupInfo.lpReserved, 0, 0x40u);
    StartupInfo.wShowWindow = 5;
    StartupInfo.cb = 68;
    StartupInfo.dwFlags = 1;
    v1 = CreateProcessA(0, lpCommandLine, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
    v6 = v1 != 0;
    if ( v1 )
    {
        CloseHandle(ProcessInformation.hProcess);
        CloseHandle(ProcessInformation.hThread);
    }
    free_str(&lpCommandLine);
    return v6;
}
```

Let's change to calc.exe,
but read-only ...

Function2 in Flash Player

```
BOOL __cdecl sub_10600A30(unsigned int lpAddress, unsigned int len, char a3)
{
    LPCVOID v3; // edi@1
    int v4; // ecx@1
    SIZE_T v5; // ebx@1
    SIZE_T v6; // esi@5
    BOOL result; // eax@7
    DWORD f10ldProtect; // [sp+10h] [bp-20h]@4
    struct _MEMORY_BASIC_INFORMATION Buffer; // [sp+14h] [bp-1Ch]@5

    v3 = (LPCVOID)lpAddress;
    v4 = sub_10669EF0();
    v5 = len;
    if ( lpAddress % v4 || len % v4 )
        abort();
    f10ldProtect = 0;
    do
    {
        VirtualQuery(v3, &Buffer, 0x1Cu);
        v6 = Buffer.RegionSize;
        if ( v5 <= Buffer.RegionSize )
            v6 = v5;
        result = VirtualProtect((LPCVOID)v3, v6, a3 != 0 ? 32 : 4, &f10ldProtect);
        v3 = (char *)v3 + v6;
        v5 -= v6;
    }
    while ( v5 && result );
    return result;
}
```

Let's change to read-write

New exploit

1. Prepare the arguments
2. Call Function2 -> VirtualProtect()
3. Change to “calc.exe”
4. Call Function1 -> CreateProcess()
- 5.



Function-Oriented Programming

Disadvantage of FOP

- ~~Platform dependent~~
- ~~Easily detected (eg. EMET)~~
- ~~Cost of time~~
- ~~Continuous execution~~

Exploit mitigations



~~DEP~~



~~ASLR~~



~~EMET~~



Browser
Sandbox

Browser sandbox

- Demo

Demo

<http://youtu.be/pyC5NII1BRE>

Conclusion

- Flash Player is more hacker-friendly than others
- SWF is a good carrier in target attack

Reference

- [1] Haifei Li, "Smashing the Heap with Vector: Advanced Exploitation Technique in Recent Flash Zero-day Attack"

Q & A

Email: wmliang.tw@gmail.com

Twitter: [@_wmliang_](https://twitter.com/_wmliang_)