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

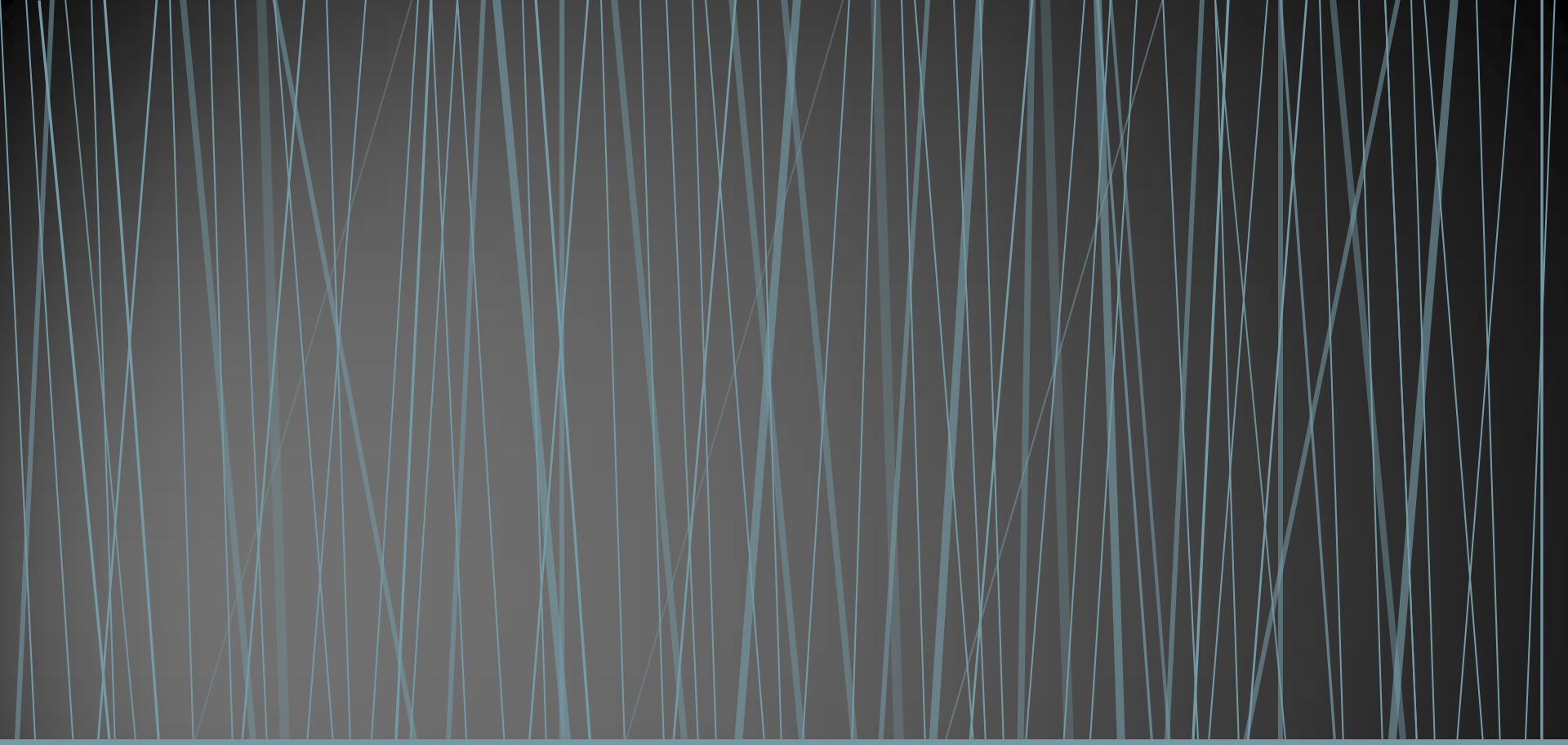
- NCTU Software Quality Lab
- Trend Micro
- Focus on…
  - Document exploit
  - Antivirus engine
  - CTF

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



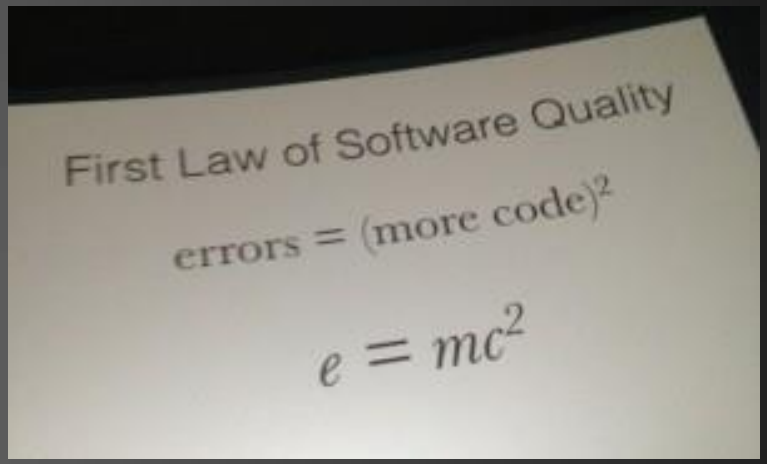
Crash it

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

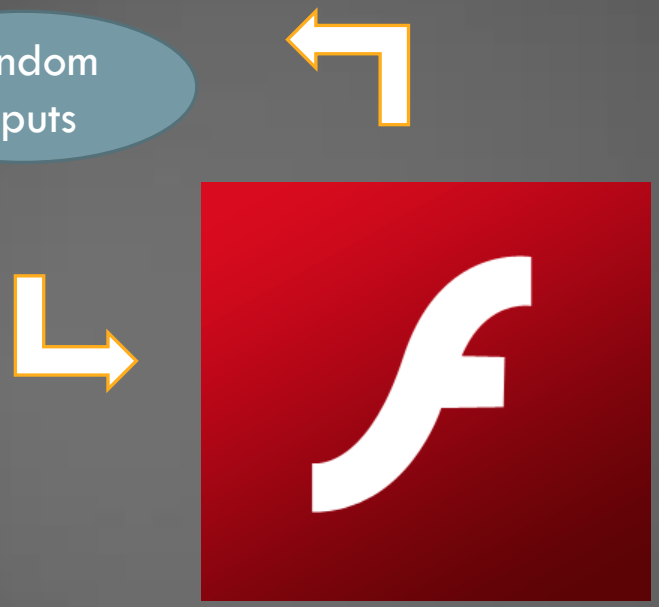
Why RE

- 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



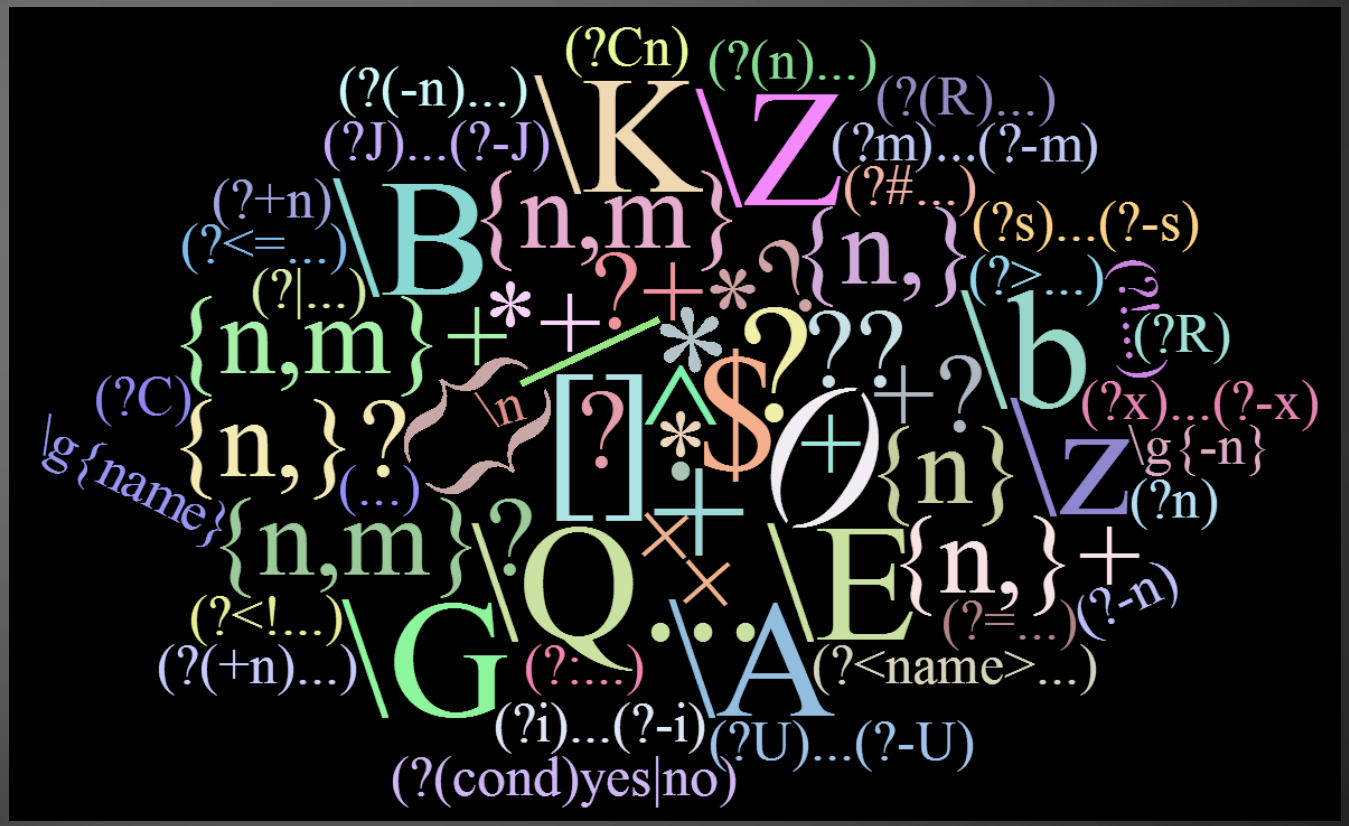
What is fuzz testing

Random Inputs



PCRE fuzzing

- List all PCRE syntax

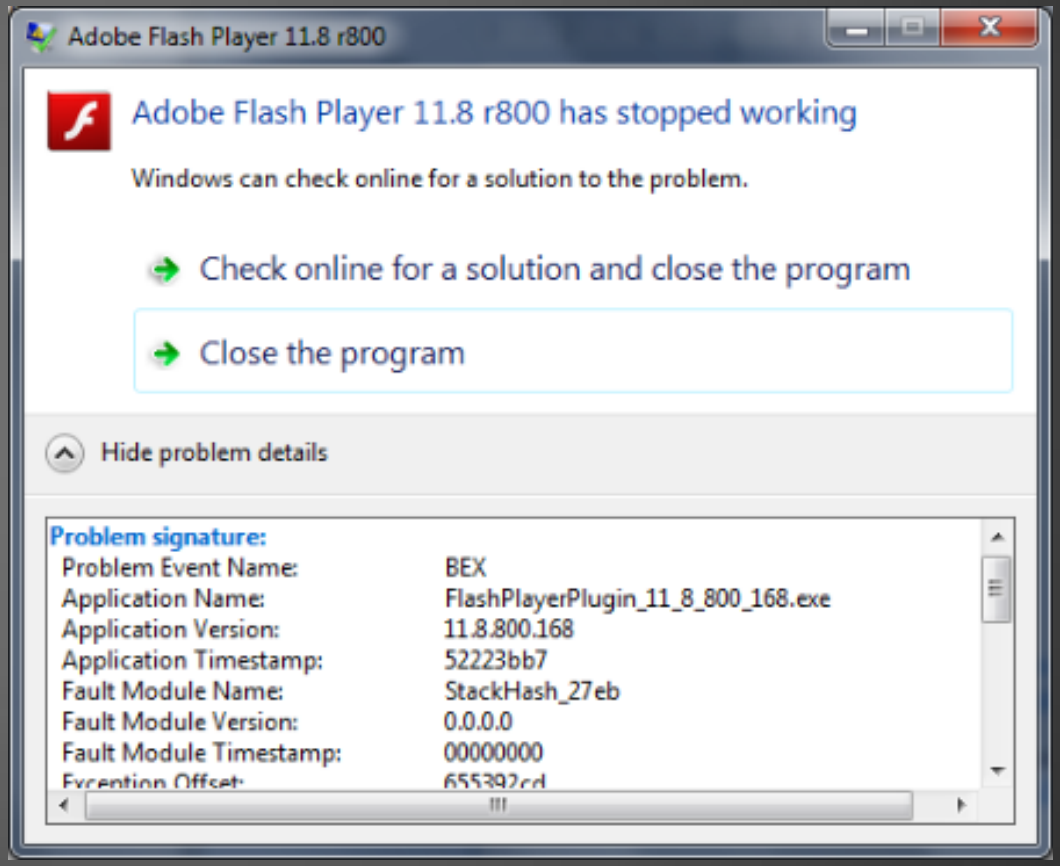


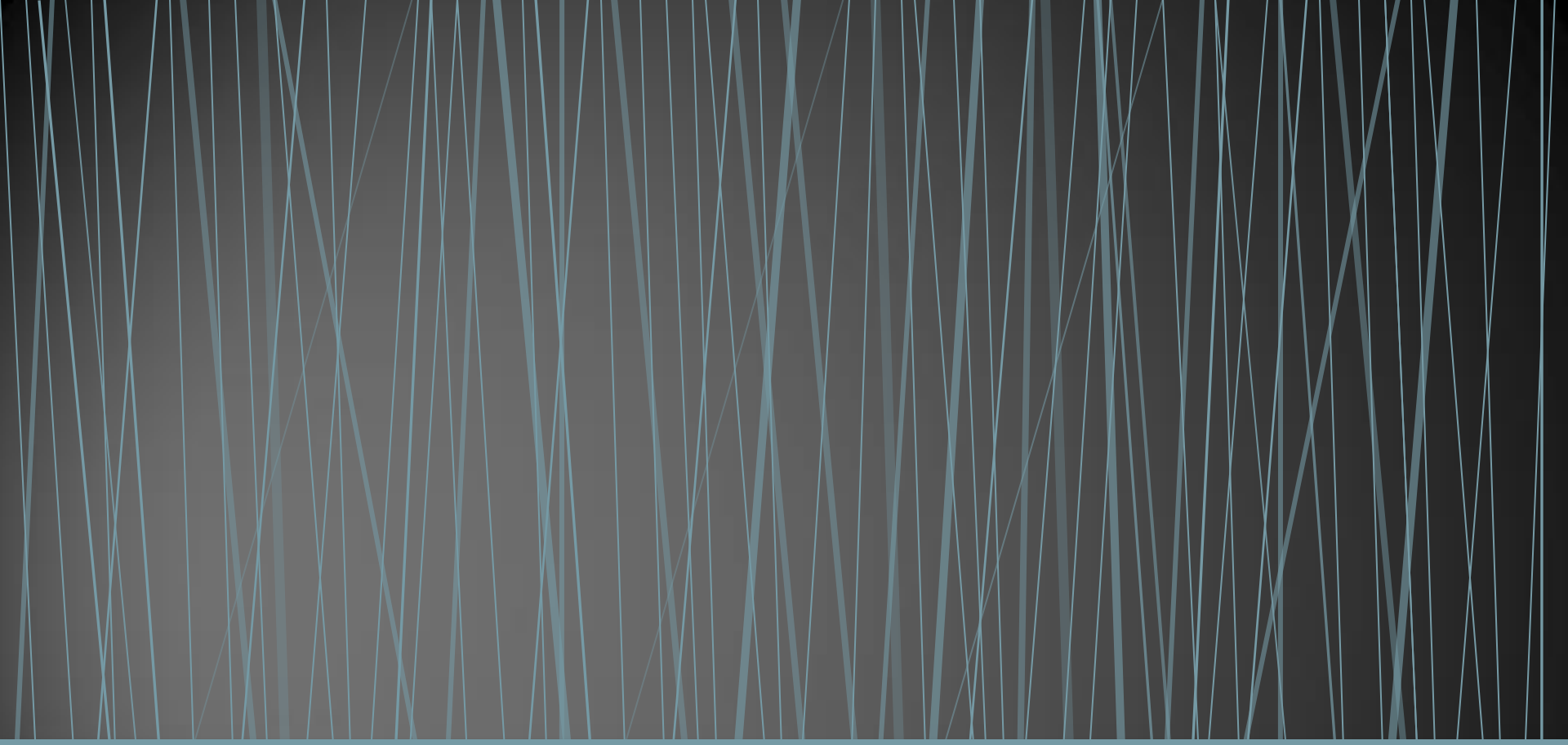
PCRE fuzzing

- Fuzzer in Actionscript

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

Just wait for crash



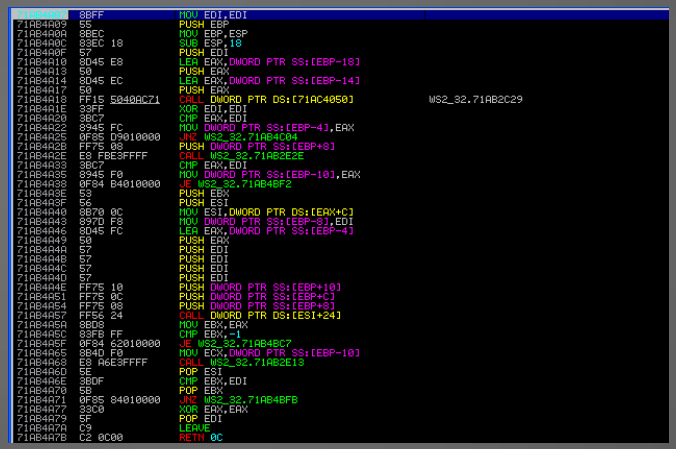


Debug it

Now…

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

Debug Flash Player



Debug Flash Player

- What the …

Debug Flash Player

Debug Flash Player

- Figure out the mapping of ActionScript and x86 assembly

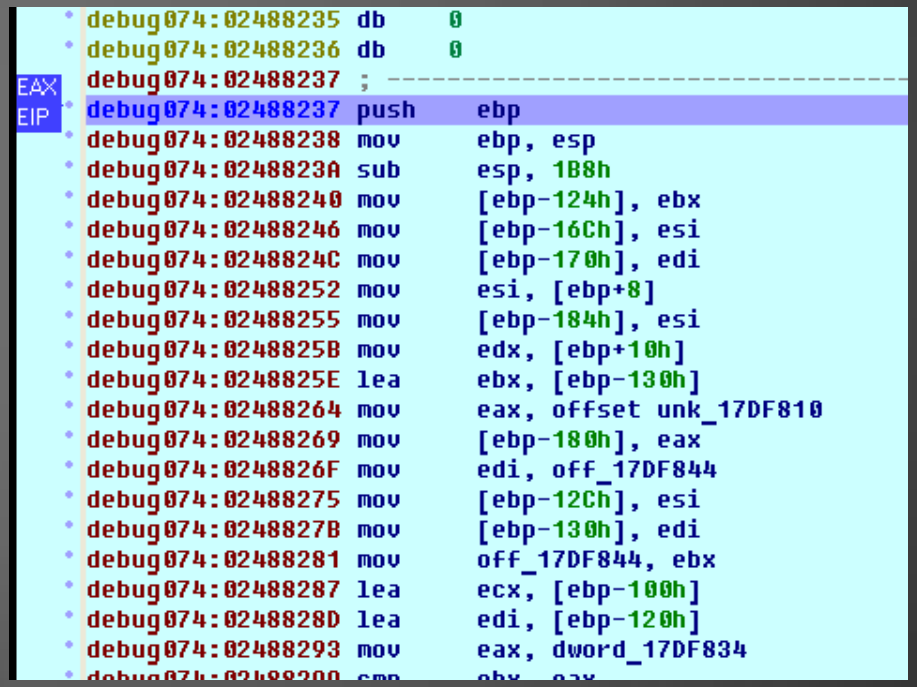
Debug Flash Player

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



Debug Flash Player

Open source project

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

Debug PCRE

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

CVE-2014-0559

Root cause

- In PCRE compiler

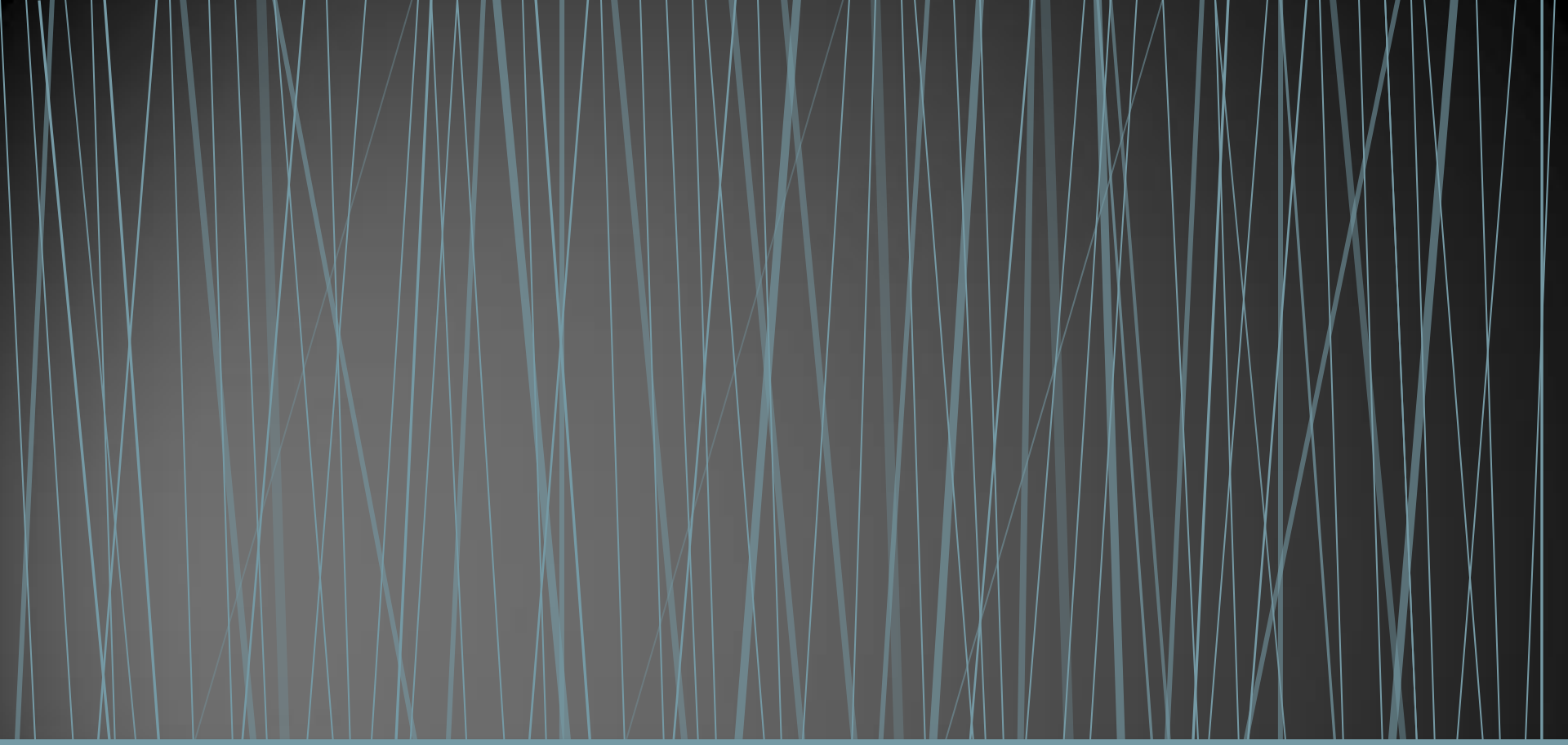
```
function pcre_compile() {
    int length;
    // first-phase for estimating the length of compiled RE
    compile_regex(NULL, &size);
    re = malloc(size);
    // second-phase for compiling RE actually
    compile_regex(re, NULL);
}
```

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

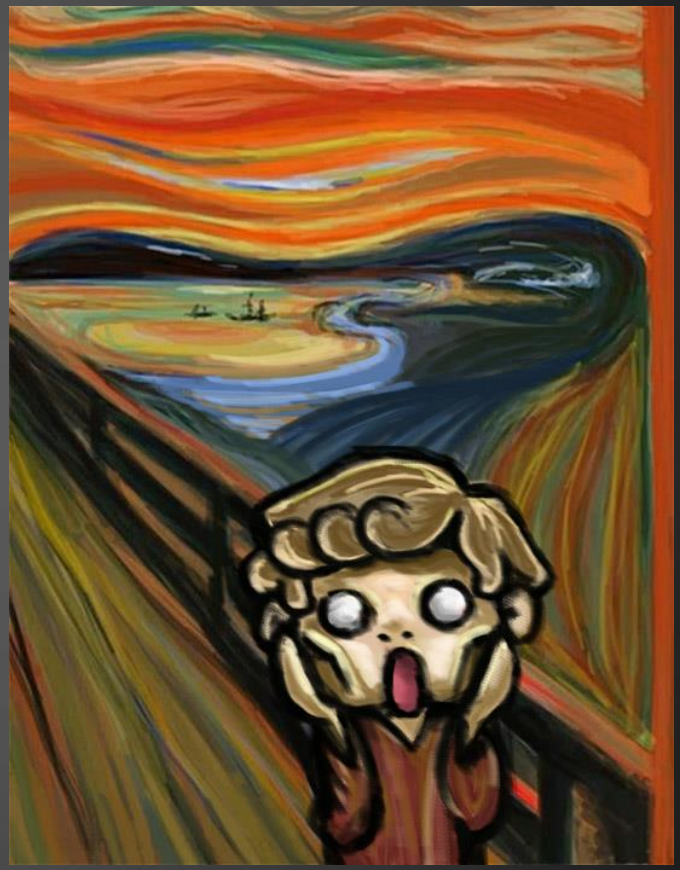
Exploit in school

- Heap exploit
- ASLR
- DEP

SNOWMEN

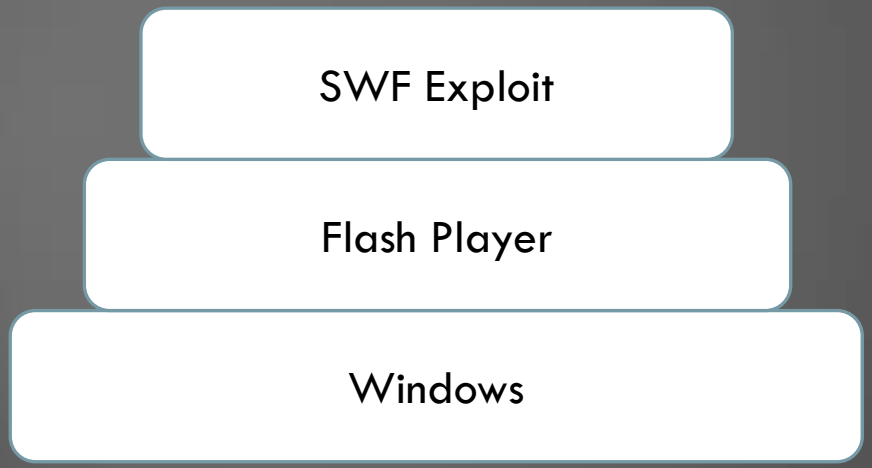
Exploit in real

- 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()

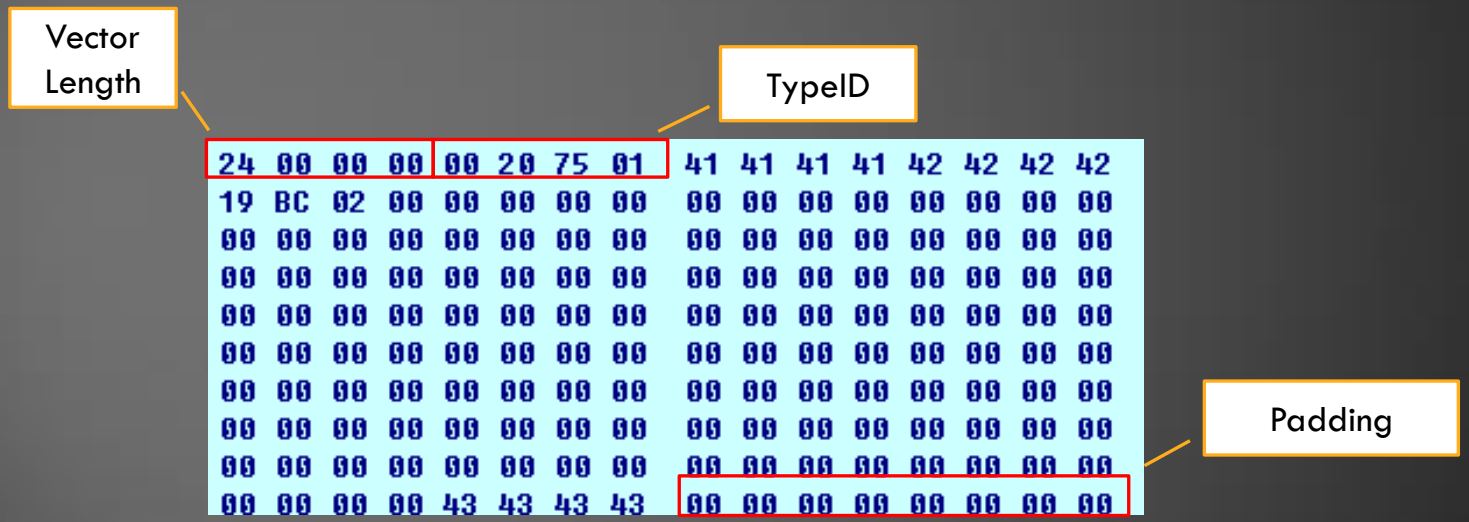
RtlAllocateheap()

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
    - …

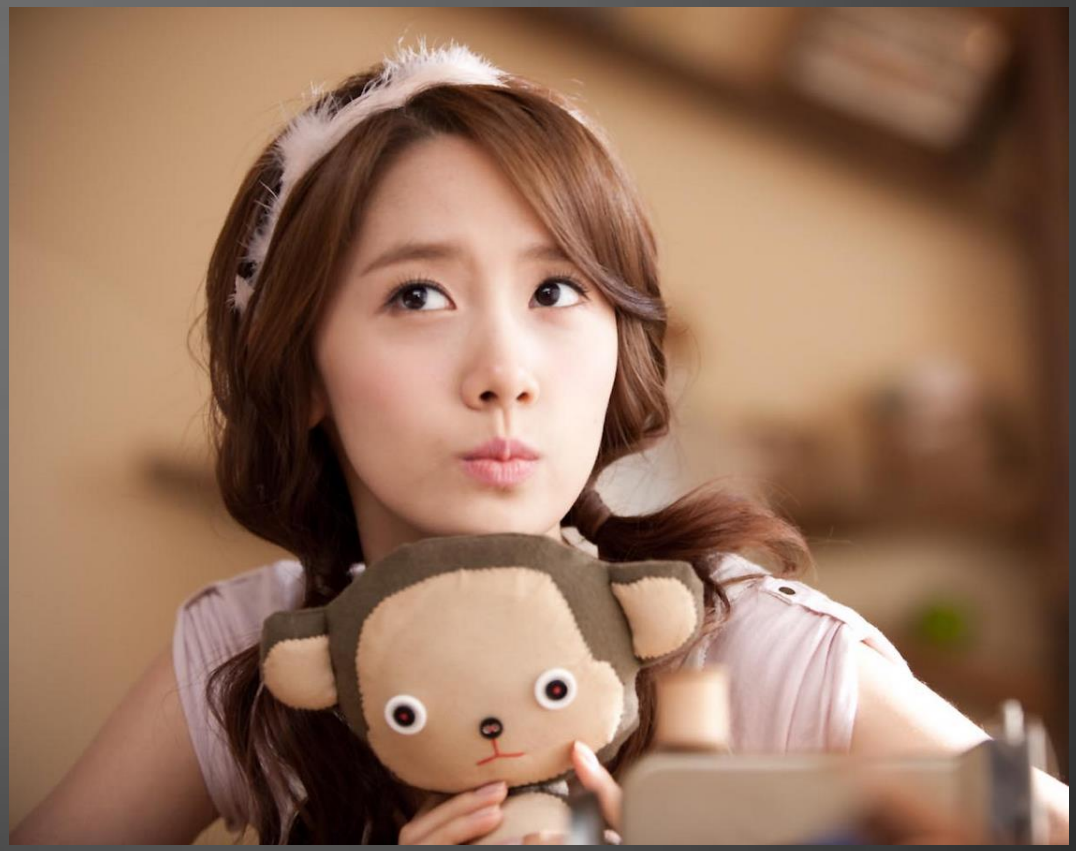
Application data

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

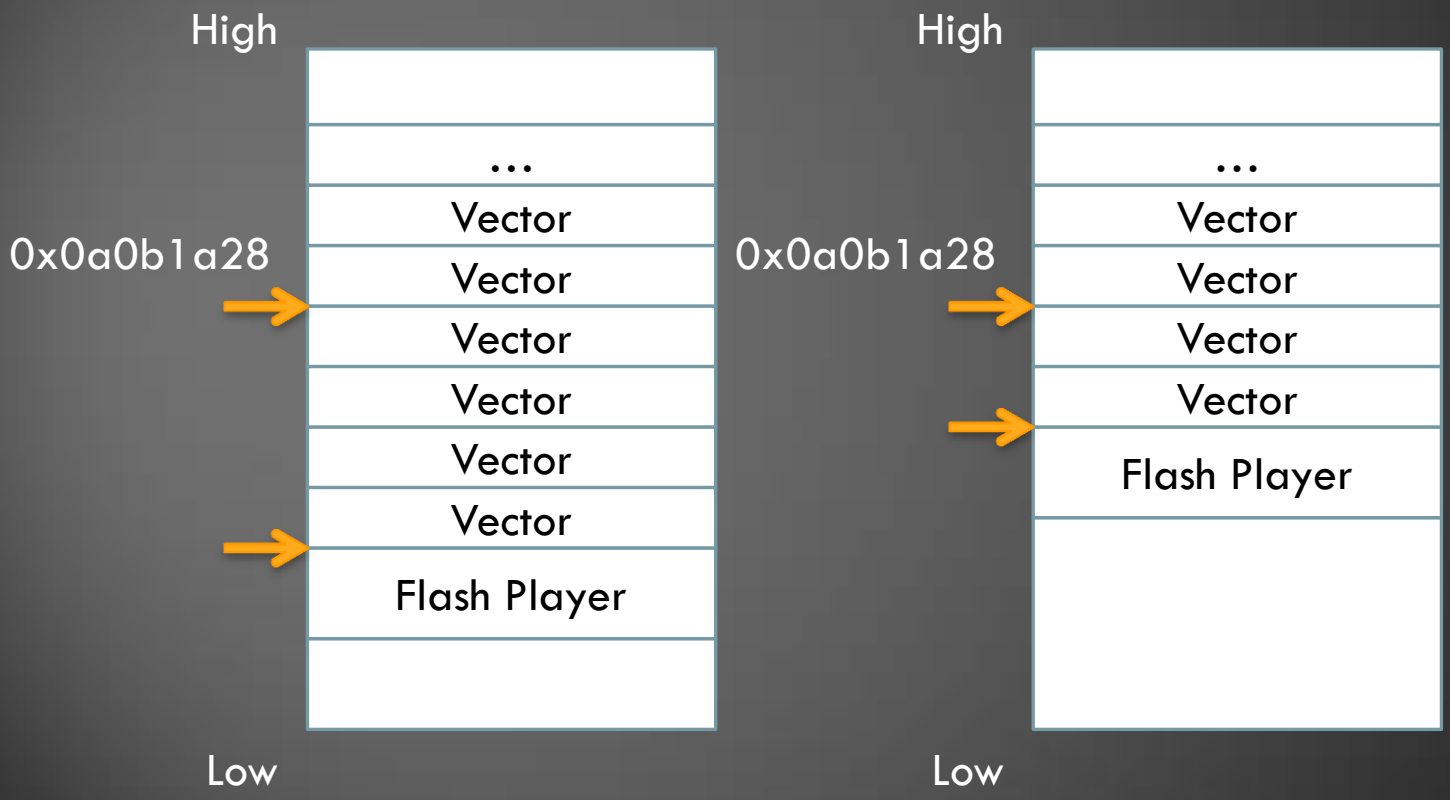


Application data

- How to place the RegExp object before vector object ?



Heap spray


High
0x0a0b1a28
...
Vector
Vector
Vector
Vector
Vector
Vector
Flash Player
Low
High
0x0a0b1a28
...
Vector
Vector
Vector
Vector
Flash Player
Low

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

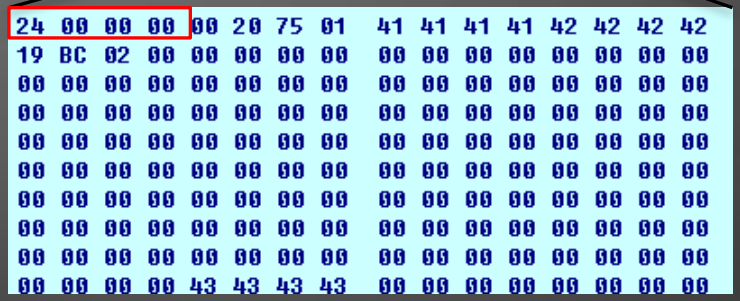
- The objective of heap massage is to control the memory layout
- Memory allocation is fewer than heap spray

Memory layout

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

- Allocate Vector objects in memory

Vector length



a Vector object with 0xA0 bytes

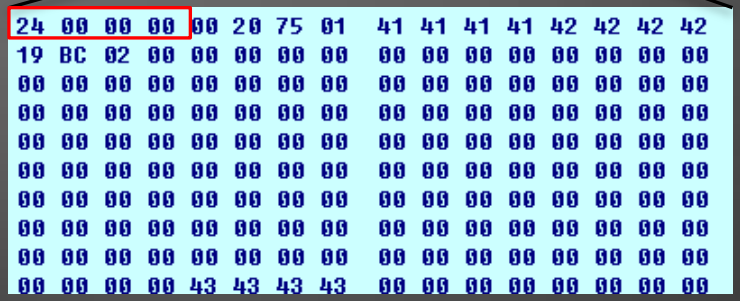
Memory Layout

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

- Free some of them

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



a Vector object with 0xA0 bytes

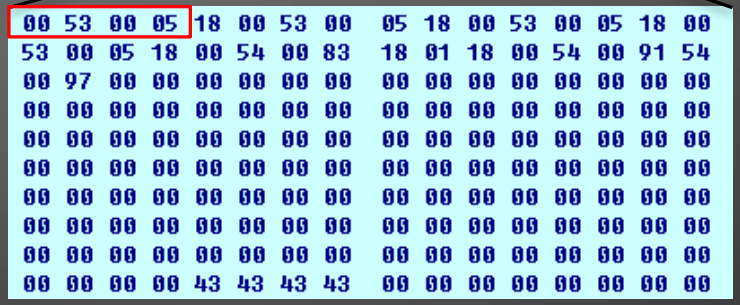
new RegExp(CVE_2014_0559, "");

Memory layout

- Allocate malicious RE object and trigger the overflow

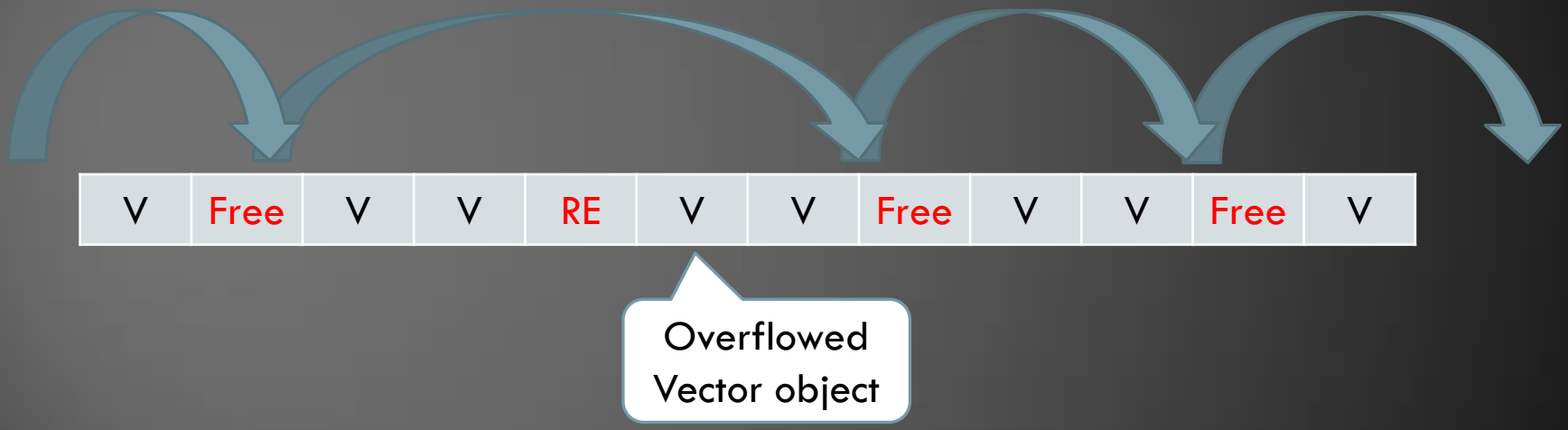
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	Free	V	V	V	Free	V
V	V	RE	V	V	V	Free	V

Vector length

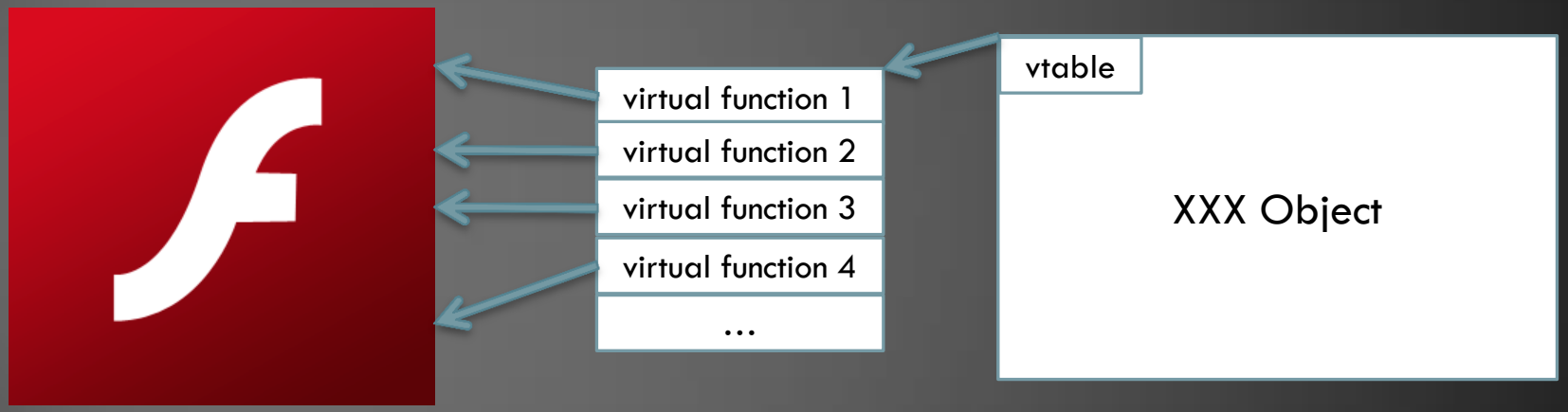


a Vector object with 0xA0 bytes

Locate the address

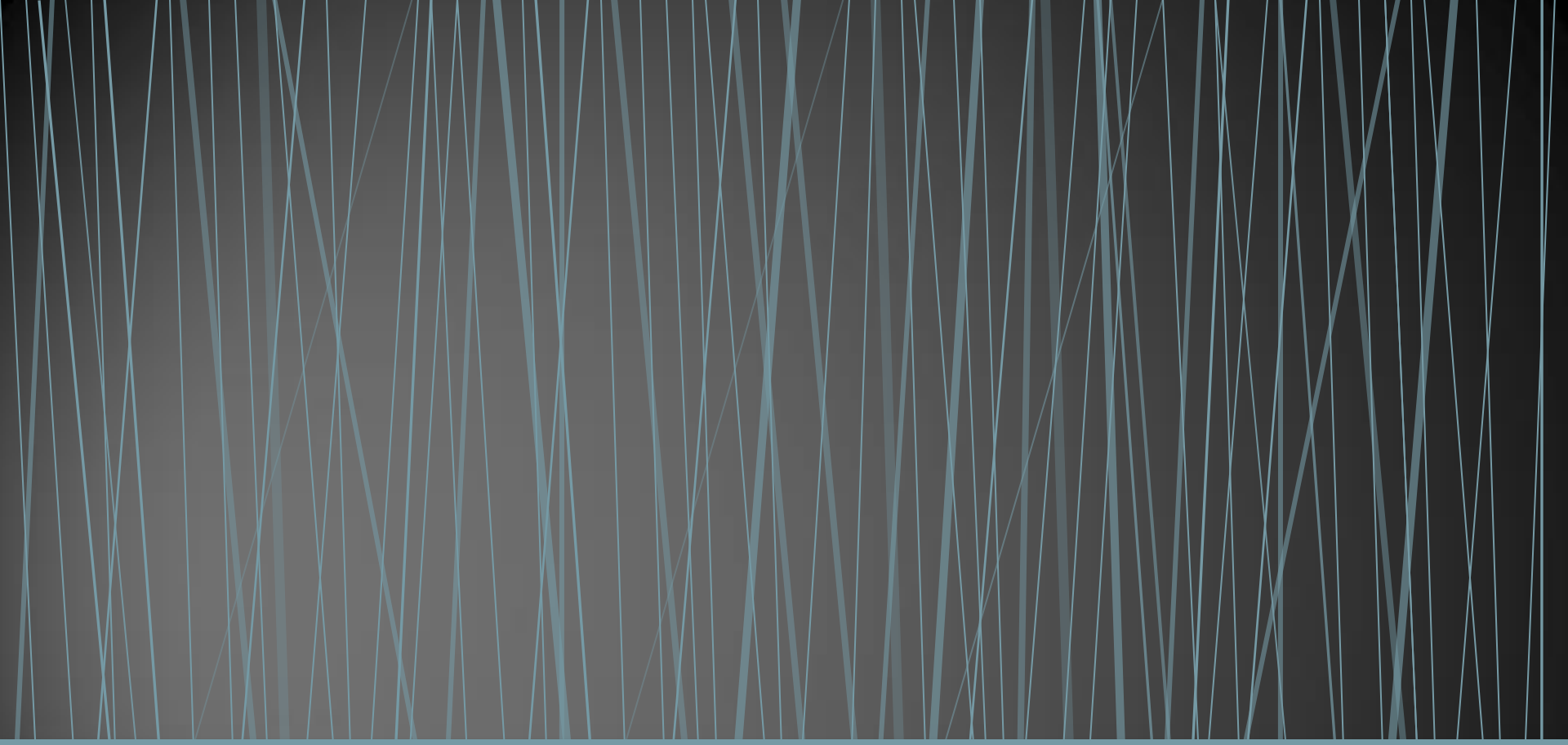


Bypass ASLR


virtual function 1
virtual function 2
virtual function 3
virtual function 4
...
vtable
XXX Object

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 flOldProtect; // [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();
  flOldProtect = 0;
  do
  {
    VirtualQuery(v3, &Buffer, 0x1Cu);
    v6 = Buffer.RegionSize;
    if ( v5 <= Buffer.RegionSize )
      v6 = v5;
    result = VirtualProtect((LPVOID)v3, v6, a3 != 0 ? 32 : 4, &flOldProtect);
    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_