QCon[上海站]
全球软件开发大会2016

浅谈代码复用攻击与防御

SPEAKER

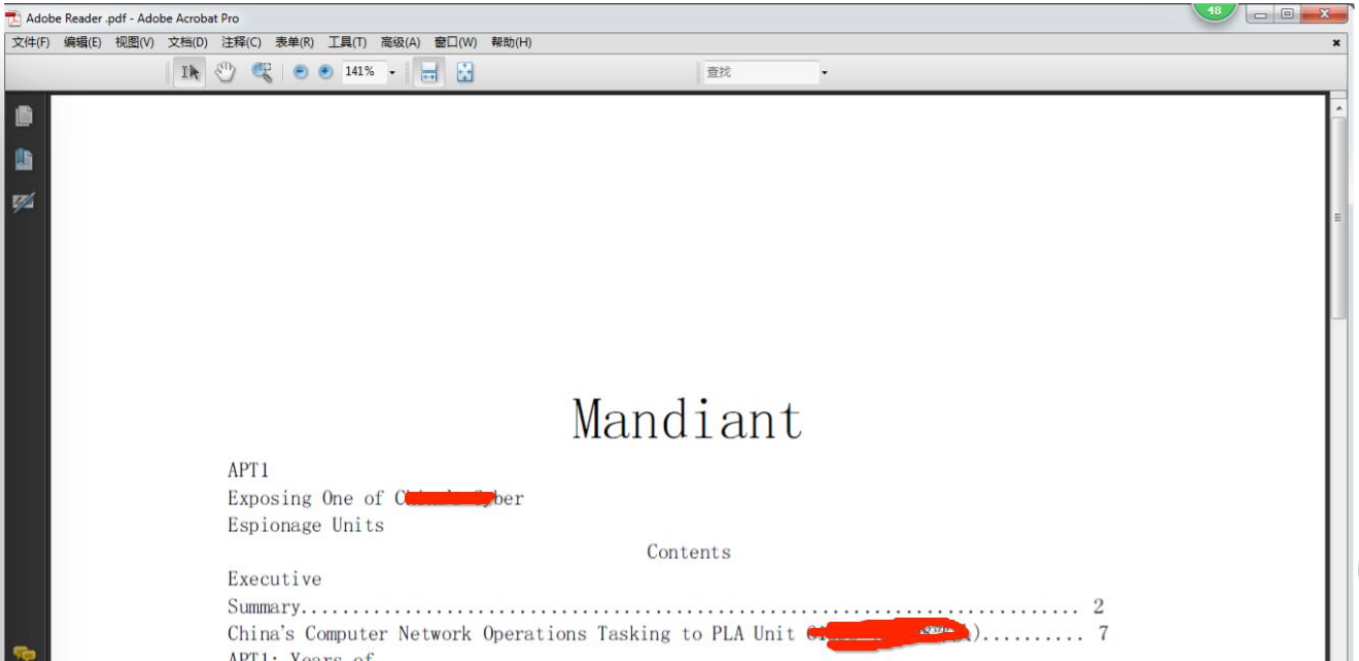
陈平
点融网
2016-10-20

International Software
Development Conference

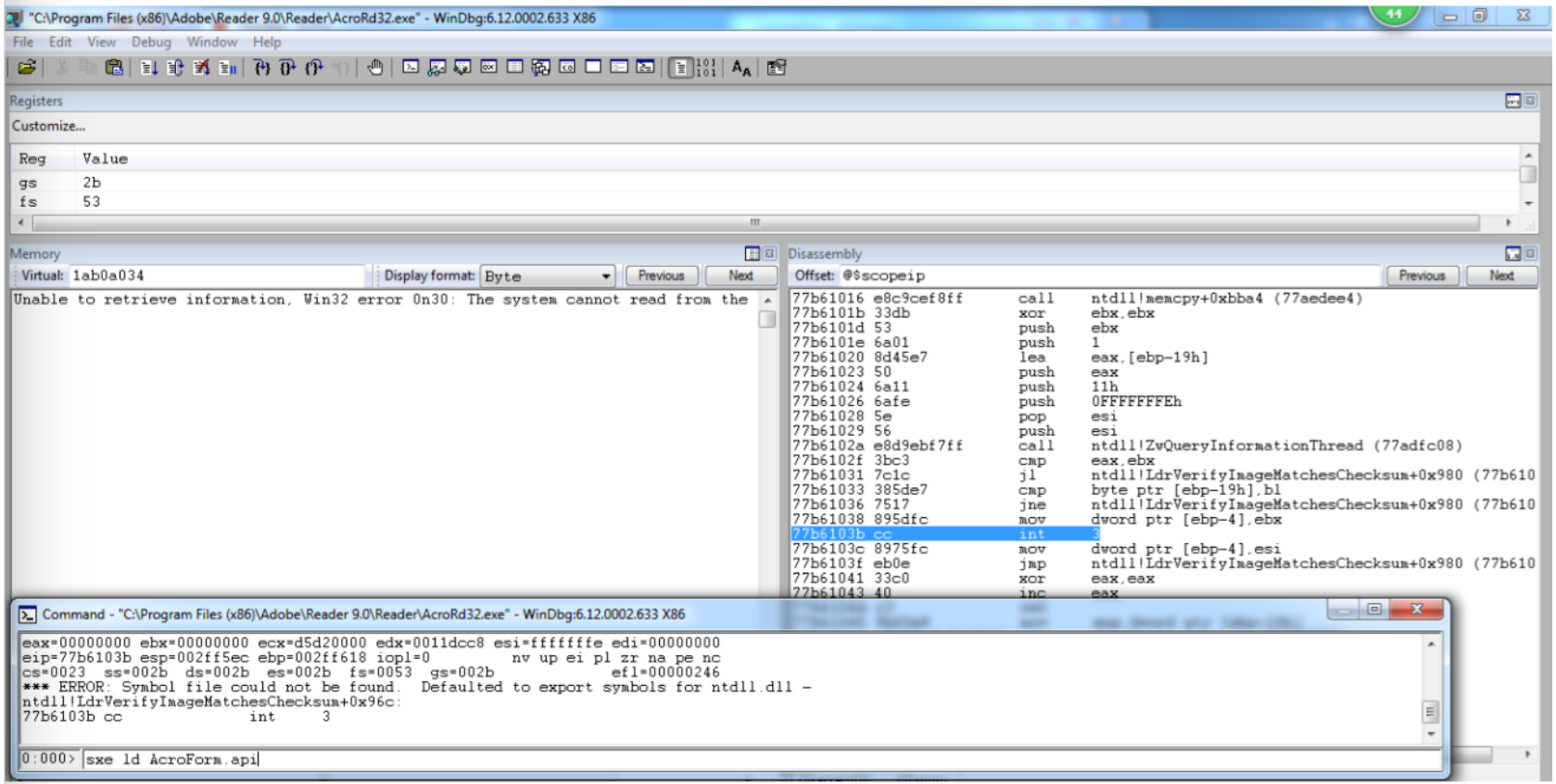
主办方 Geekbang. 极客邦科技 InfoQ

一份钓鱼邮件的PDF文件

- 2015.4 我收到一份陌生人发来的带有PDF附件的邮件。
  Windows 7 & Adobe Reader 9.5.0.270



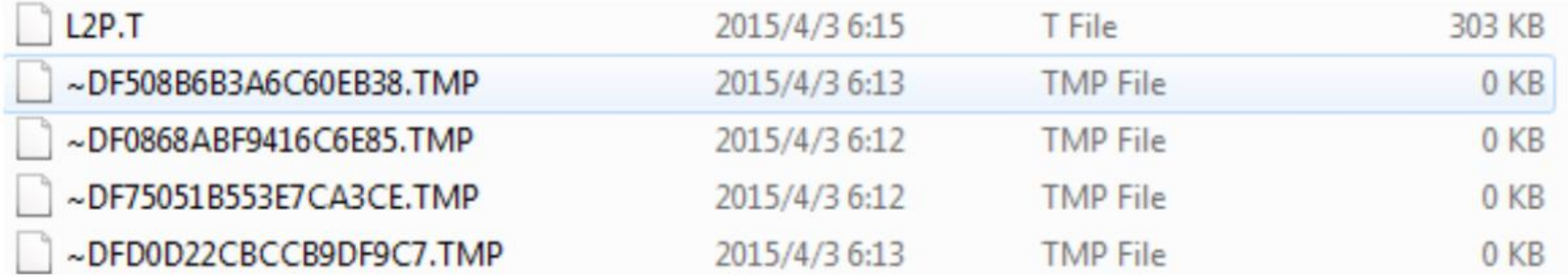
CVE-2013-0640



使用ROP 绕过沙盒

```
r+=getUnescape(AcroForm+0x1029); //ret
......
r+=getUnescape(AcroForm+0x1029); //ret
r+=getUnescape(AcroForm!PlugInMain+0x4a6547); //push esp/pop esi/ret
......
r+=getUnescape(AcroForm!PlugInMain+0x4697ca); // xchg    eax,ecx/ret
r+=getUnescape(AcroForm!PlugInMain+0x511185); // pop eax/ret
r+=getUnescape(AcroForm!PlugInMain+0x4945e0); // movzx   eax,ax/ret
r+=getUnescape(AcroForm!PlugInMain+0x4e2a9e); // xchg    eax,edi/ret
r+=getUnescape(AcroForm!PlugInMain+0x3239a); // add     edi,esi/ret
r+=getUnescape(AcroForm!PlugInMain+0x547ce); // lea     eax,[edi]/ret
r+=getUnescape(AcroForm!PlugInMain+0x4ce342); //mov [eax], ecx/ret
r+=getUnescape(AcroForm!PlugInMain+0x511185); // pop eax/ret
// 204aa15c= kernel32!LoadLibraryA (759c499f))}
r+=getUnescape(AcroForm!PlugInMain+0x4b6642); //call [eax]/ret
//ptr to "MSVCR80.dll"
r+=getUnescape(AcroForm!PlugInMain+0x4697ca); //xchg    eax,ecx/ret
r+=getUnescape(AcroForm!PlugInMain+0x511185); // pop eax/ret
r+=getUnescape(AcroForm!PlugInMain+0x4945e0); // movzx   eax,ax/ret
r+=getUnescape(AcroForm!PlugInMain+0x4e2a9e); // xchg    eax,edi/ret
r+=getUnescape(AcroForm!PlugInMain+0x3239a); // add     edi,esi/ret
r+=getUnescape(AcroForm!PlugInMain+0x547ce); // lea     eax,[edi]/ret
r+=getUnescape(AcroForm!PlugInMain+0x4ce342); //mov [eax], ecx/ret
r+=getUnescape(AcroForm!PlugInMain+0x511185); // pop eax/ret
// kernel32!GetProcAddress (759c1222)
r+=getUnescape(AcroForm!PlugInMain+0x4b6642); //call [eax]/ret
r+=getUnescape(AcroForm+0x1d84); //jmp     eax { MSVCR80!wcsstr (6dcb0c0a)}
```

木马文件



L2P.T	2015/4/3 6:15	T File	303 KB
~DF508B6B3A6C60EB38.TMP	2015/4/3 6:13	TMP File	0 KB
~DF0868ABF9416C6E85.TMP	2015/4/3 6:12	TMP File	0 KB
~DF75051B553E7CA3CE.TMP	2015/4/3 6:12	TMP File	0 KB
~DFD0D22CBCCB9DF9C7.TMP	2015/4/3 6:13	TMP File	0 KB

提纲

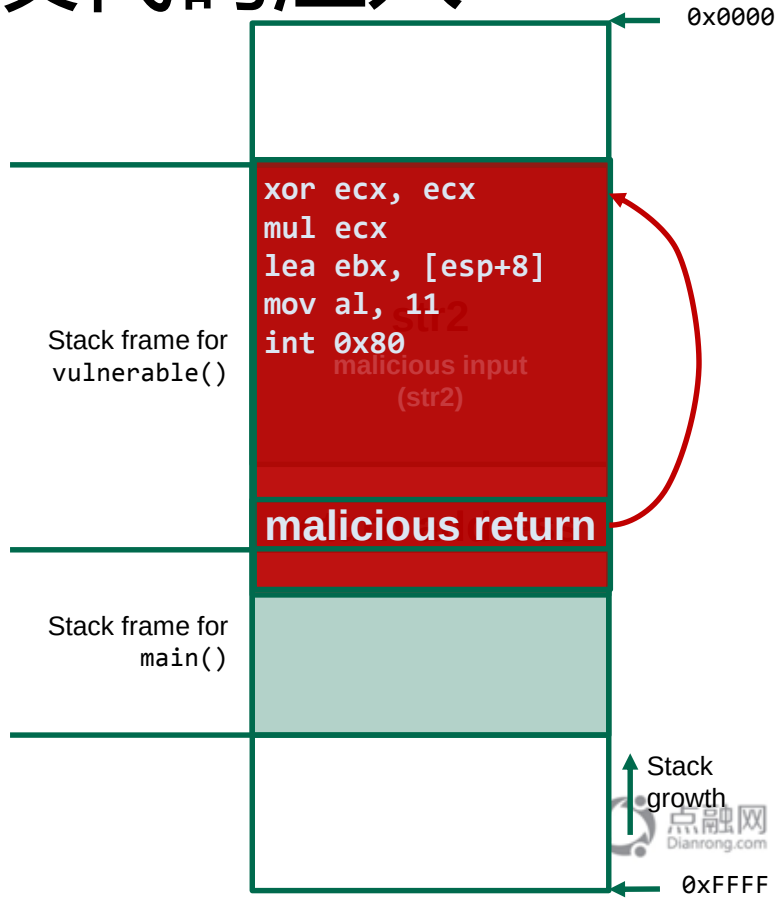
- 一、代码复用攻击简介及其危害
- 二、Return Oriented Programming (ROP)
- 三、Just-in-Time Code Reuse
- 四、JIT Spraying Attacks
- 五、Defenses

一、代码复用攻击简介及危害

一、常见漏洞分类

缓冲区溢出以及代码注入

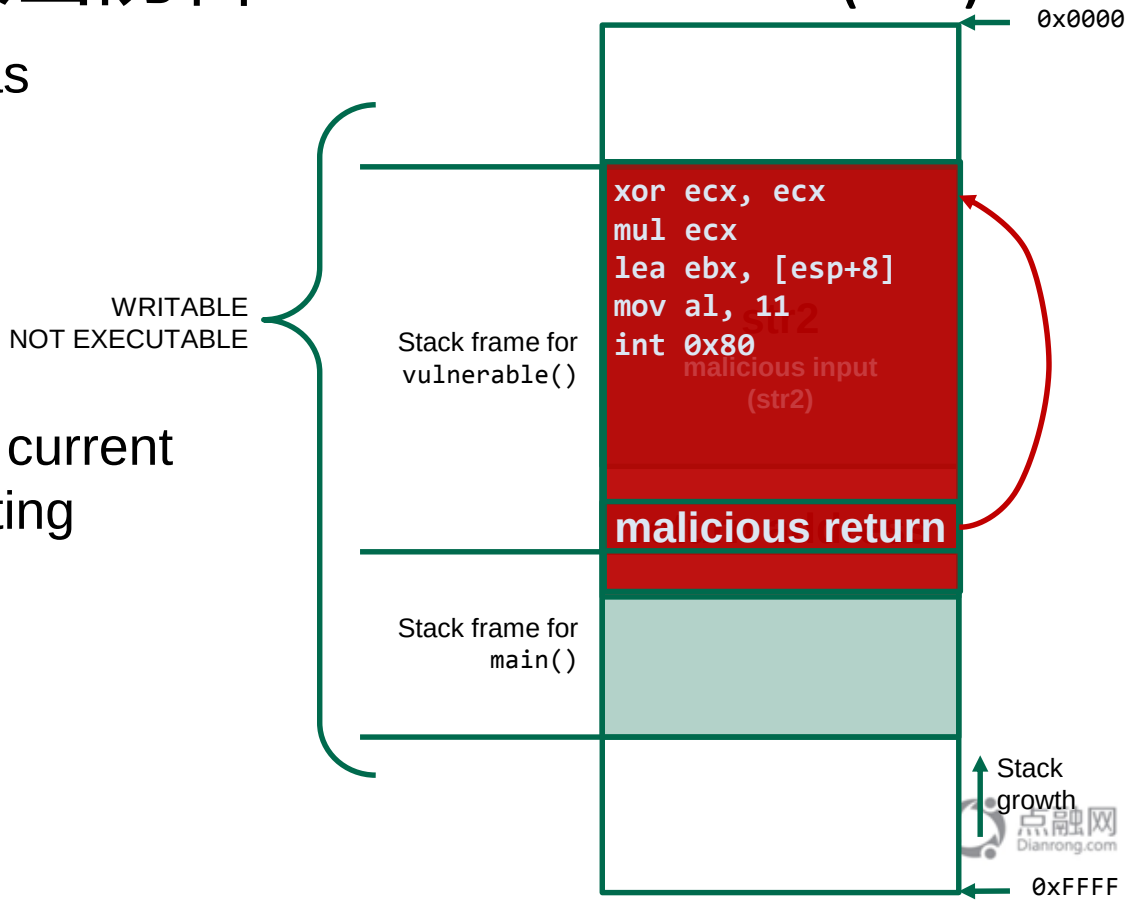
```
main (int argc, char **argv)
{
   ...
   vulnerable(argv[1]);
   ...
}

vulnerable(char *str1)
{
   char str2[100];
   strcpy(str2,str1);
   return;
}
```



代码注入攻击防御: No Execute Bit (NX)

- Mark memory pages as
  - Either WRITABLE
  - Or EXECUTABLE
  - But not both

- Standard technique in current processors and operating systems
  - Intel XD bit
  - AMD XN bit
  - Windows DEP
  - Linux PaX



代码复用攻击Code Reuse Attacks

- 主要思想: 复用程序中已有的代码，而不引入额外的代码
- 可以绕过NX
- 常见攻击
  - Return Oriented Programming
  - Just-in-Time ROP
  - JIT Spraying

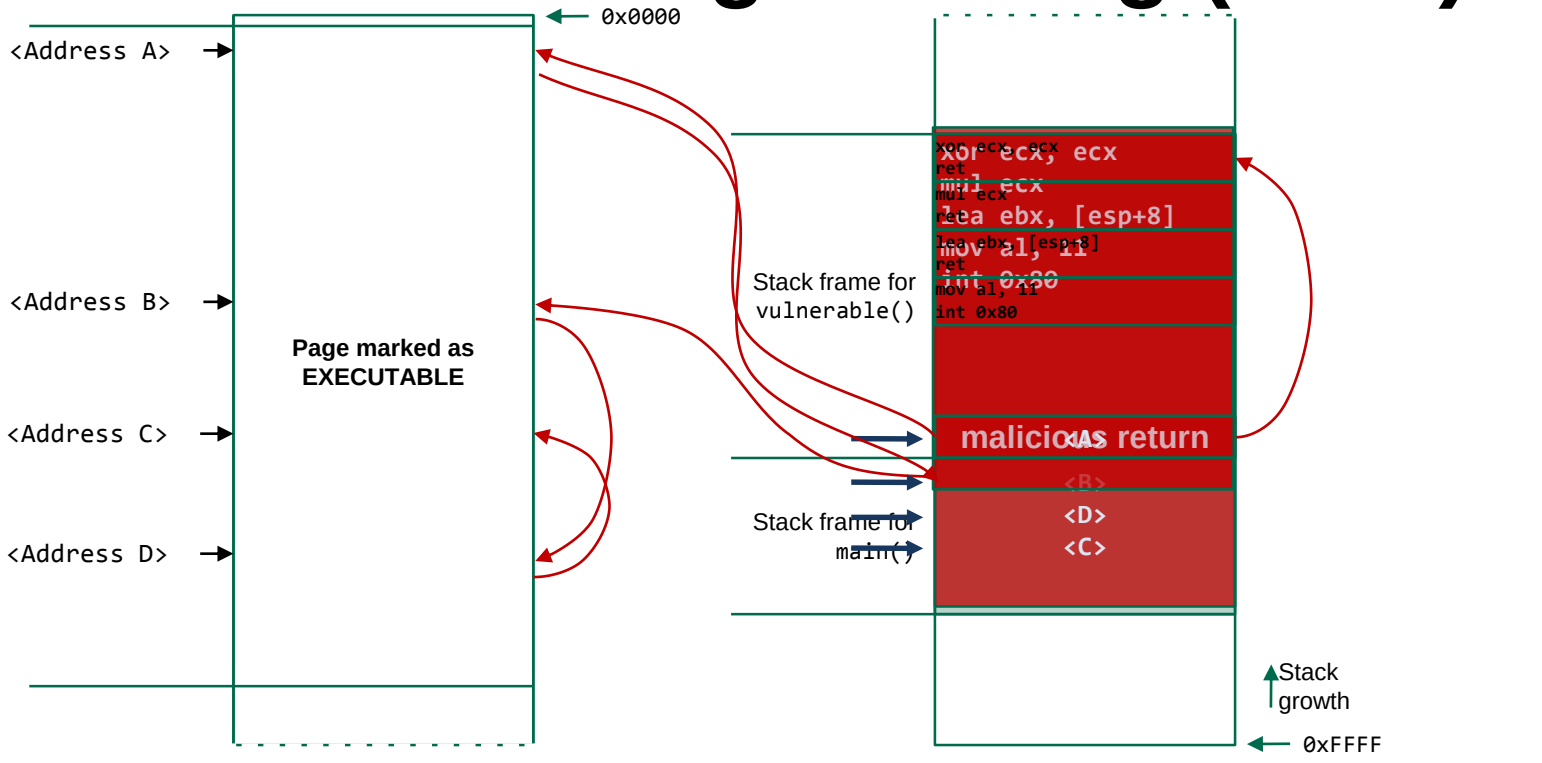
二、 Return Oriented Programming (ROP)

二、Return Oriented Programming (ROP)

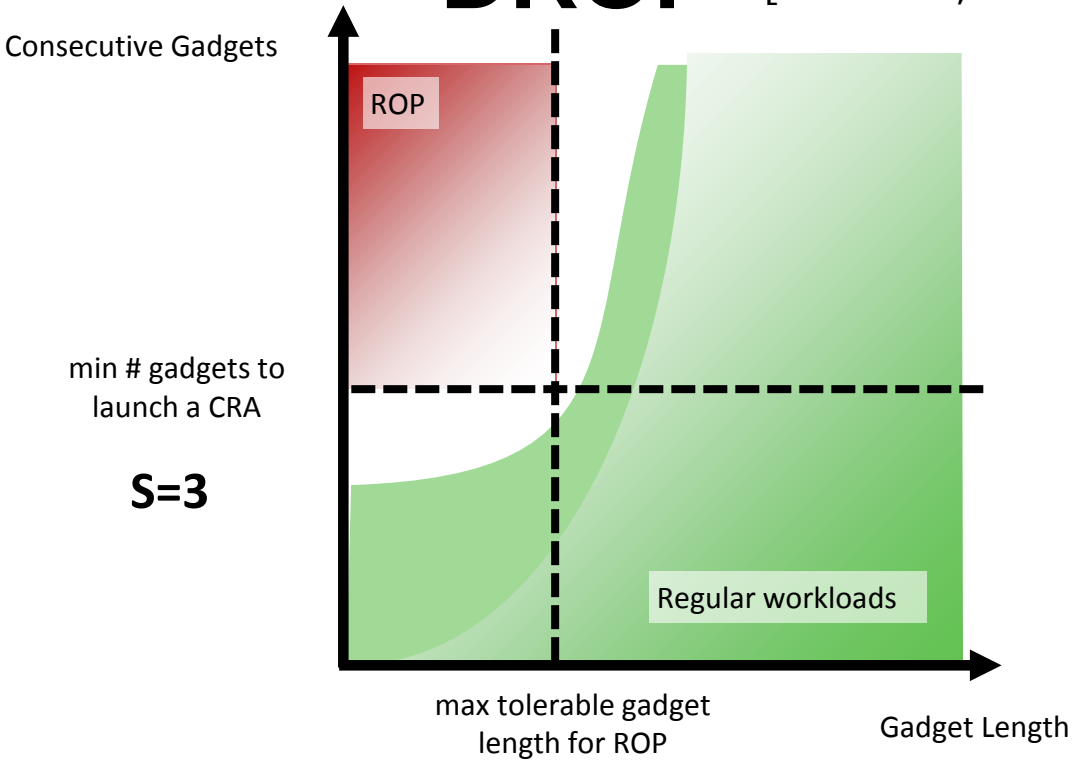
- Turing-complete
  - X86
  - SPARC
  - ARM
  - …
- Exploits
  - Cisco router
  - Xen hypervisor
  - Voting machine
  - Atmel sensor
  - Pwn2Own
- Automated tools

- Microsoft BlueHat Prize ($260K)

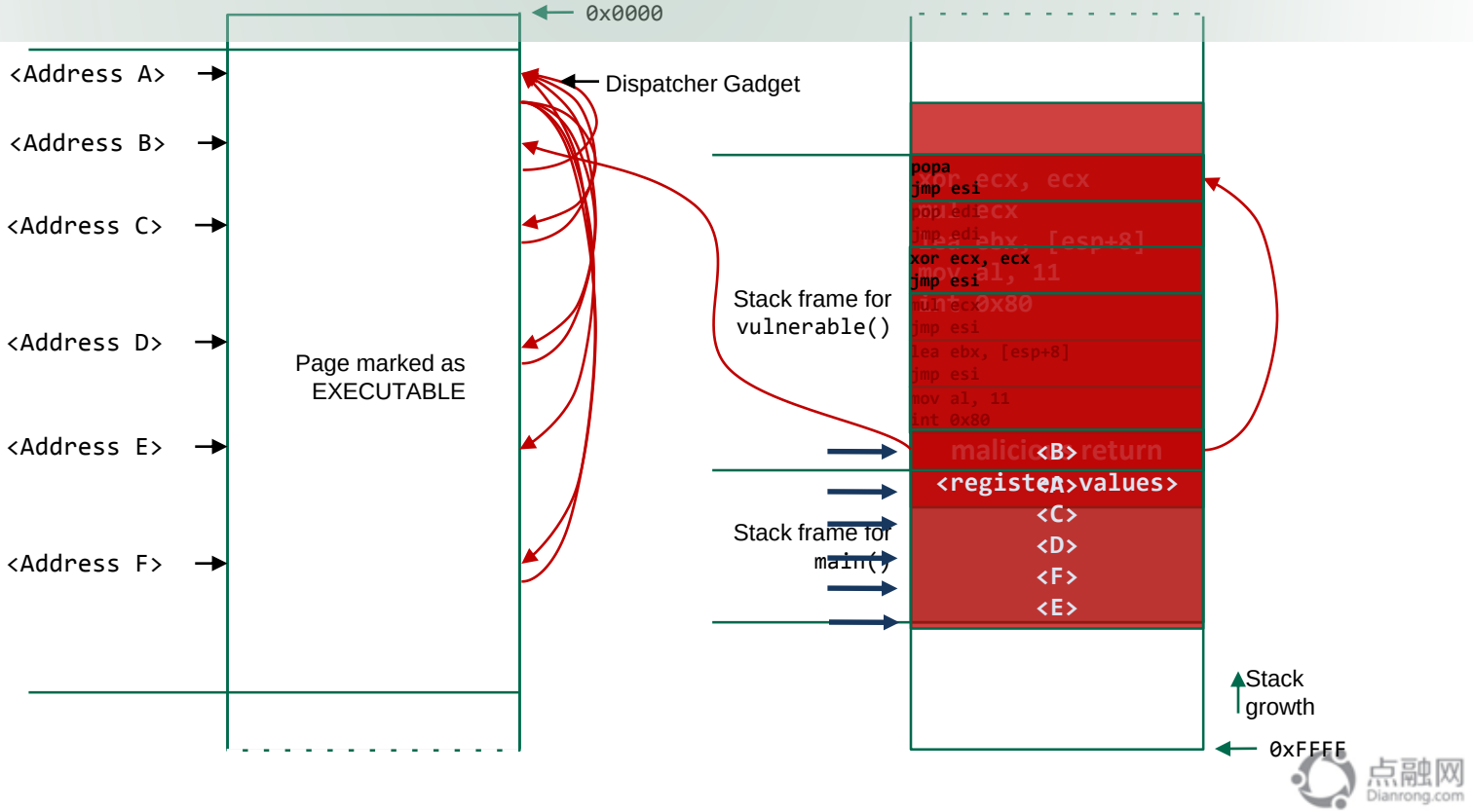
Return Oriented Programming (ROP)



DROP [Chen et al., ICISS 2009]



Jump Oriented Programming



In-Place Code Randomization

[Pappas et al., Oakland 2012]

- Instruction reordering

```
MOV EAX, &p1
MOV EBX, &p2
```

```
MOV EBX, &p2
MOV EAX, &p1
```

- Instruction substitution

```
MOV EBX, $0
```

```
XOR EBX, EBX
```

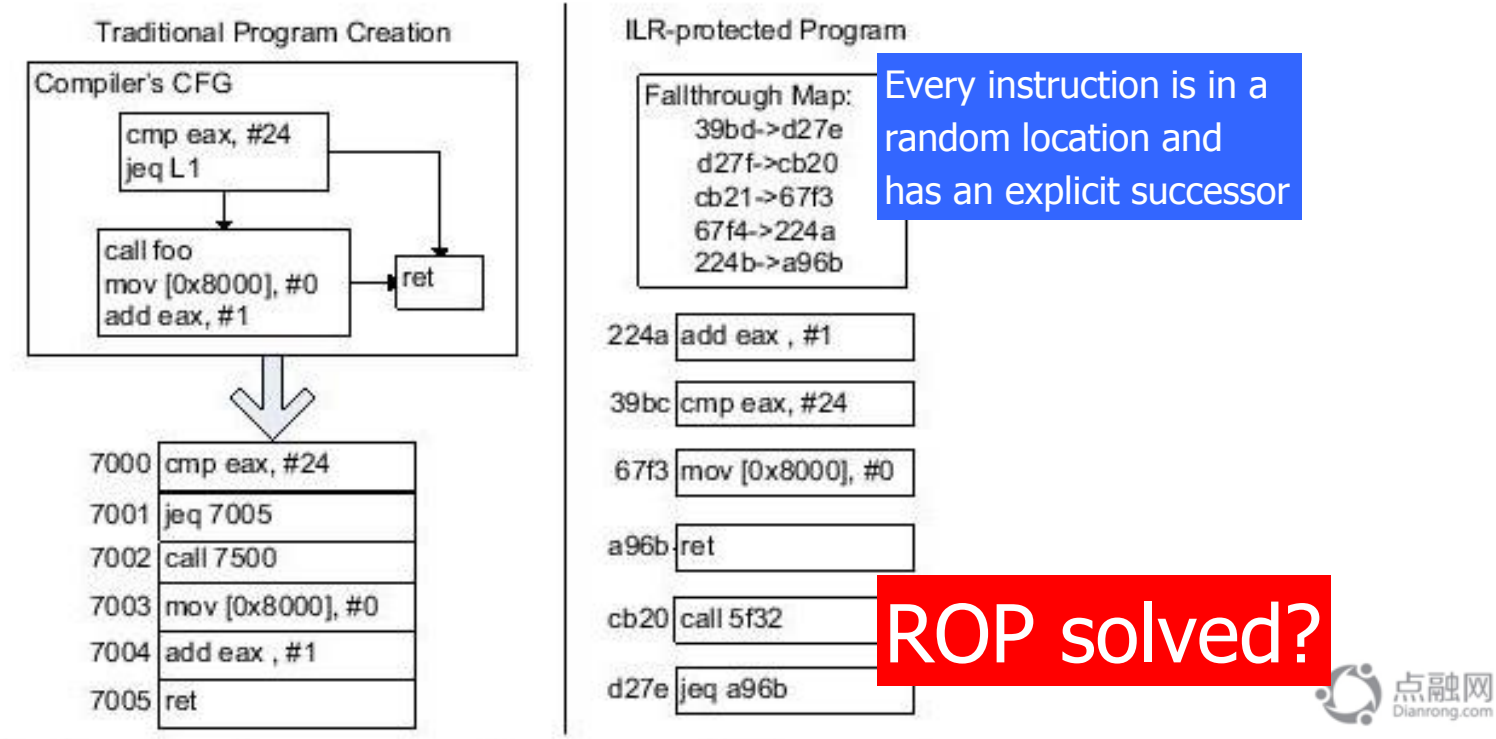
- Register re-allocation

```
MOV EAX, &p
CALL *EAX
```

```
MOV EBX, &p
CALL *EBX
```

Instruction Location Randomization

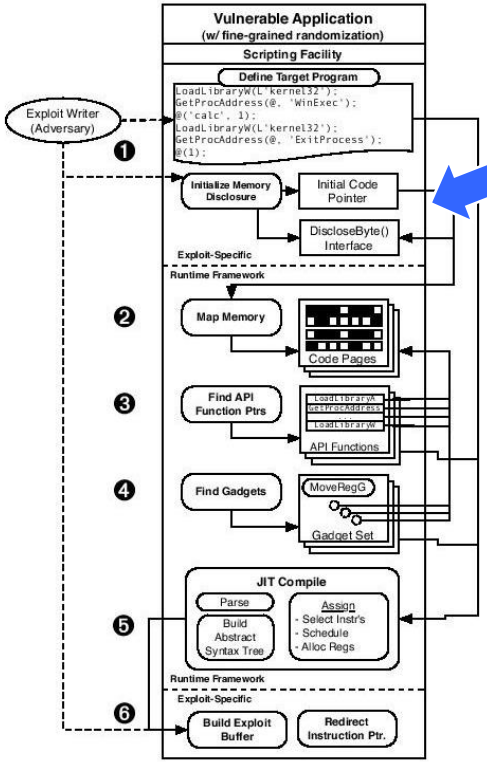
[Hiser et al., Oakland 2012]



三、Just-in-Time Code Reuse

Just-in-Time Code Reuse (1)

[Snow et al., Oakland 2013]



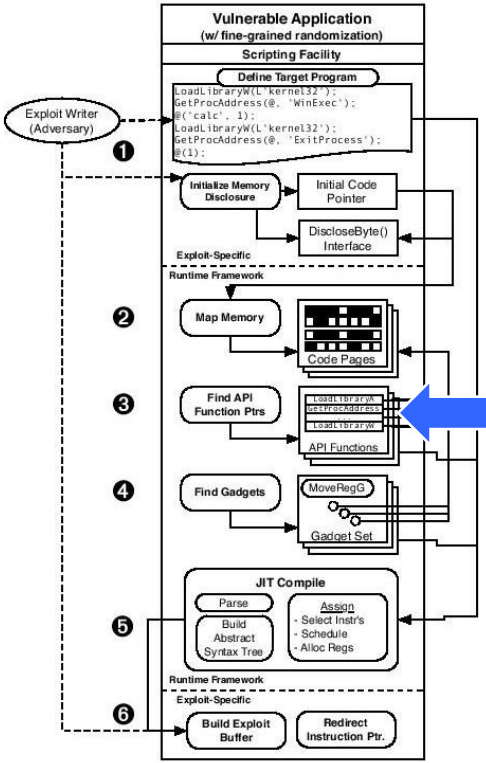
Find one code pointer
(using any disclosure vulnerability)

The entire page must be code…
Analyze the instructions to find
jumps and calls to other code pages…

Map out a big portion of
the application's code pages

Just-in-Time Code Reuse (2)

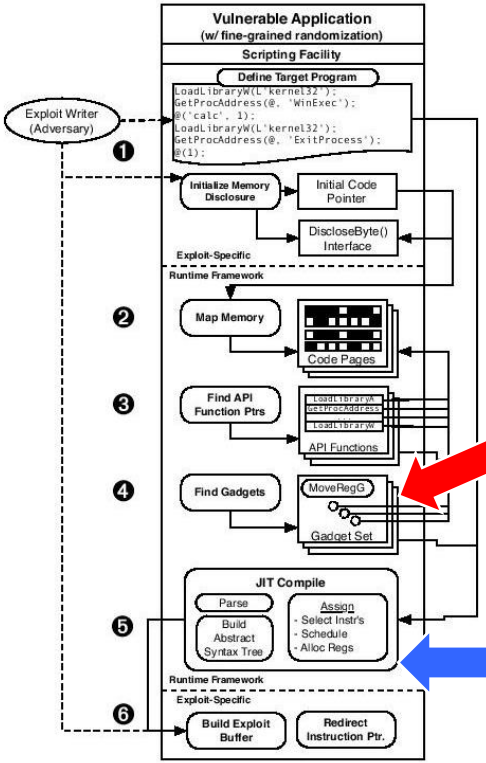
[Snow et al., Oakland 2013]



Use typical opcode sequences to find calls to LoadLibrary() and GetProcAddr()…
These can be used to invoke any library function by supplying the right arguments - don't need to discover the function's address!

Just-in-Time Code Reuse (3)

[Snow et al., Oakland 2013]



Collect gadgets in runtime by analyzing the discovered code pages (dynamic version of Shacham's "Galileo" algorithm)

Compile on the fly into shellcode

四、JIT Spraying Attacks

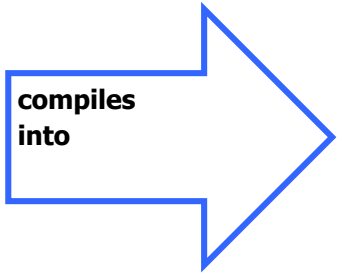
四、JIT Spraying Attacks

[Dion Blazakis et al., Black Hat 2010]

- Circumvent the existing defenses
  - Address space is randomized – where to point?
  - DEP – can't execute data on the heap!
- Remember ActionScript?
  - JavaScript-like bytecode in Flash files
- Just-in-time (JIT) compiler will allocate writable memory and write executable x86 code into it
  - But how to get ActionScript bytecode to compile into shellcode?

四、JIT Spraying Attacks

```
var y = (
   0x3c54d0d9 ^
   0x3c909058 ^
   0x3c59f46a ^
   0x3c90c801 ^
   0x3c9030d9
   …
```



MOV EAX, 3C54D0D9

XOR EAX, 3C909058

XOR EAX, 3C59F46A

XOR EAX, 3C90C801

XOR EAX, 3C9030D9

B8 D9 D0 54 3C

35 58 90 90 3C

35 6A F4 59 3C

35 01 C8 90 3C

35 D9 30 …

Unintended Execution

假设从这里执行

MOV EAX, 3C54D0D9

XOR EAX, 3C909058

XOR EAX, 3C59F46A

XOR EAX, 3C90C801

XOR EAX, 3C9030D9

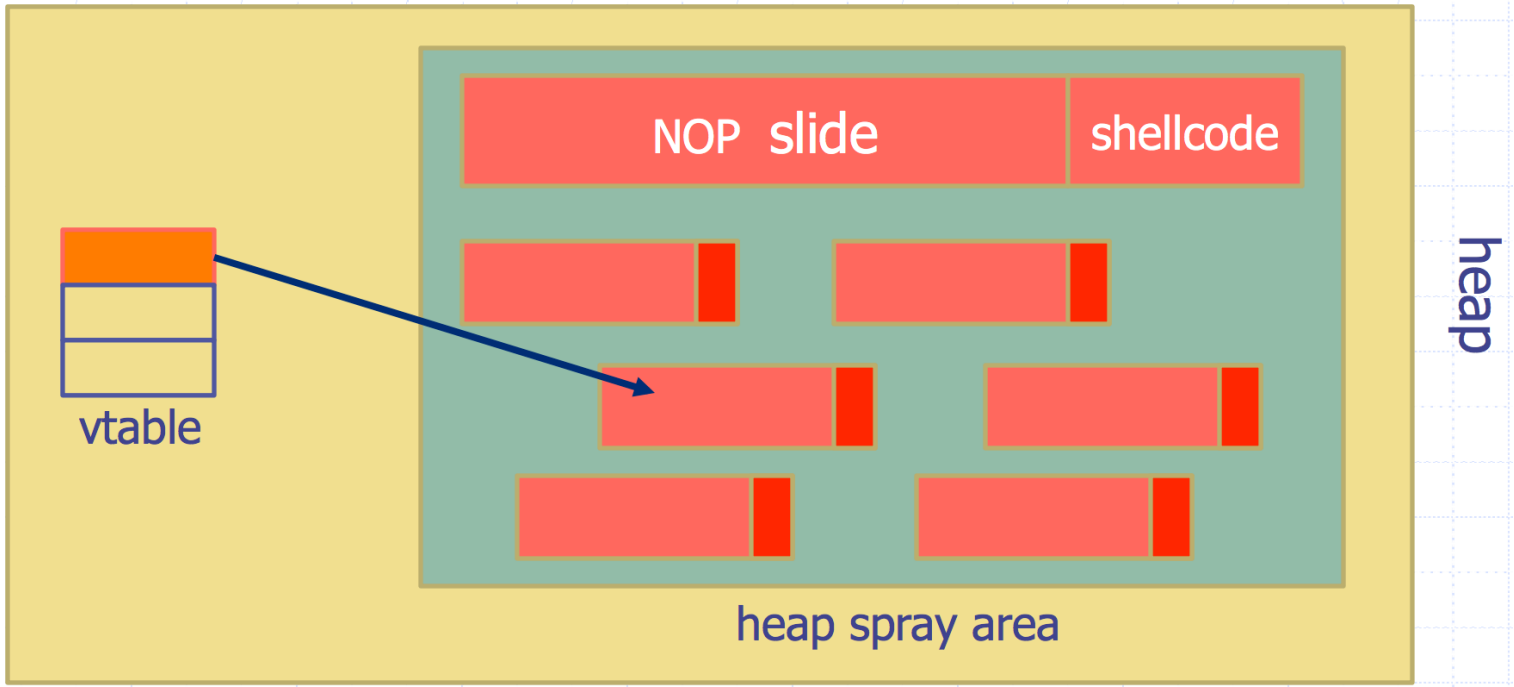
B8	
D9	FNOP
D0	
54	PUSH ESP
3C	
35	CMP AL, 35
58	POP EAX
90	NOP
90	NOP
3C	CMP AL, 35
35	
6A	PUSH -0C
F4	
59	POP ECX
3C	CMP AL, 35
35	
01	ADD EAX, ECX
C8	
90	NOP
3C	CMP AL, 35
35	
D9	FSTENV DS:[EAX]
30	
...	

这个shellcode的功能是获得它所在的位置，例如EIP:
将这个地址保存，然后再读出来。

Heap Spraying

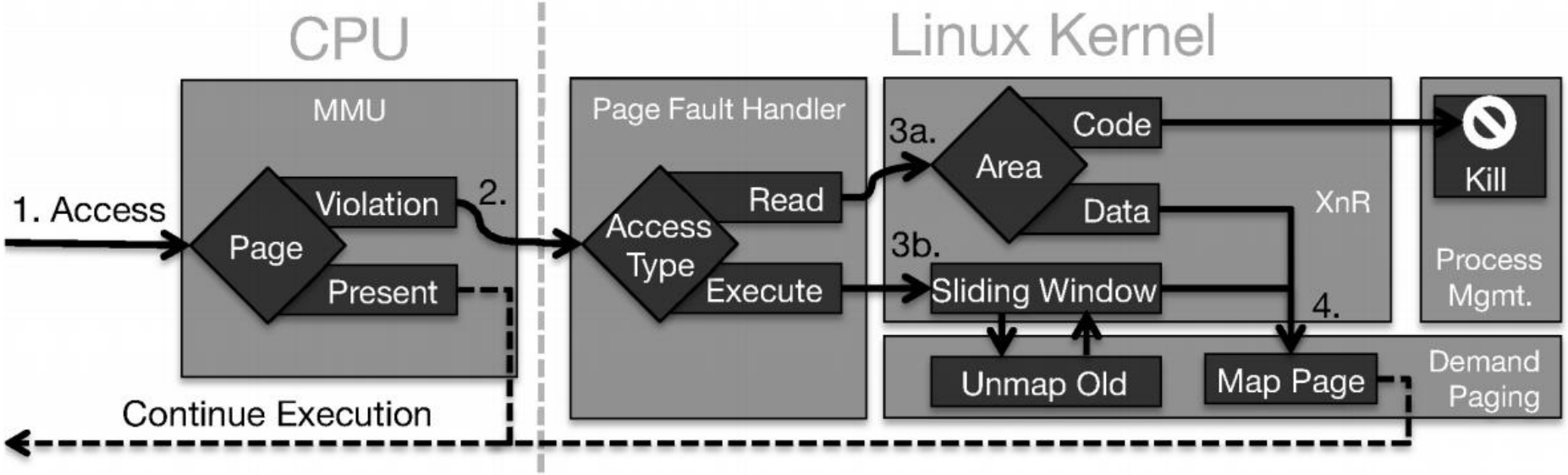
[SkyLined 2004]

Idea: 1. use Javascript to spray heap with shellcode (and NOP slides)
2. then point vtable ptr anywhere in spray area



五、Defenses

XnR



Control Flow Integrity

- **Direct jump targets** (e.g. `call 0x12345678`)
  - are all targets valid according to CFG?
- **IDs**
  - is there an ID right after every entry point?
  - does any ID appear in the binary by accident?
- **ID Checks**
  - is there a check before every control transfer?
  - does each check respect the CFG?

easy to implement correctly => trustworthy

ID Checks

Check dest label

```
FF 53 08                   call  [ebx+8]              ; call function pointer
```

is instrumented using `prefetchnta` destination IDs, to become:

```
8B 43 08                   mov   eax, [ebx+8]        ; load pointer into register
3E 81 78 04 78 56 34 12    cmp   [eax+4], 12345678h  ; compare opcodes at destination
75 13                      jne   error_label         ; if not ID value, then fail
FF D0                      call  eax                 ; call function pointer
3E 0F 18 05 DD CC BB AA    prefetchnta [AABBCCDDh]   ; label ID, used upon the return
```

Fig. 4. Our CFI implementation of a call through a function pointer.

Check dest label

Bytes (opcodes)	x86 assembly code	Comment
C2 10 00	ret 10h	; return

is instrumented using `prefetchnta` destination IDs, to become:

```
8B 0C 24                   mov   ecx, [esp]          ; load address into register
83 C4 14                   add   esp, 14h            ; pop 20 bytes off the stack
3E 81 79 04 DD CC BB AA    cmp   [ecx+4], AABBCCDDh  ; compare opcodes at destination
75 13                      jne   error_label         ; if not ID value, then fail
FF E1                      jmp   ecx                 ; jump to return address
```

Other Defenses

- Fix Bugs
  - Audit software
    - Automated tools: Coverity, Prefast/Prefix
  - Rewrite software in a type safe language (Java, ML)
    - Difficult for existing(legacy) code…

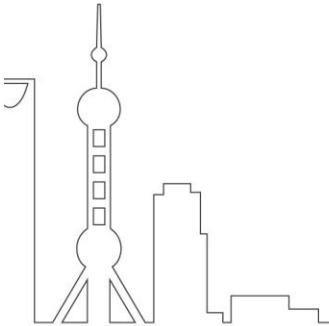
- Add runtime code to detect overflows exploits
  - Halt process when overflow exploit detected
  - StackGuard, LibSafe, …

点融安全：

邮箱：security@dianrong.com

微博：weibo.com/dianrongsec

点融安全应急响应中心：security.dianrong.com



Thanks!

International Software Development Conference

主办方 Geekbang. 极客邦科技 InfoQ