
ISC
2015
中国互联网安全大会
360互联网安全中心
ISC
2015
数据驱动安全
2015 中国互联网安全大会
China Internet Security Conference
APT与新威胁论坛

0day动态检测之插桩下的ROP检测

ROP

Return Orientated Programming（面向返回的编程）

攻击者扫描已有的动态链接库和可执行文件，提取出可以利用的指令片段(gadget)，这些指令片段均以ret指令结尾，即用ret指令实现指令片段执行流的衔接

ROP例子

```
7C34402E   5A              pop     edx
7C34402F   C3              retn
7C344030   833D 00B3387C   cmp     dword ptr [7C38B300], 0
```

```
7C374013   5D              pop     ebp
7C374014
```

```
7C34A459   FF1
7C34A45F   3BC
7C34A461   894
7C34A464   75
7C34A466   FF7
7C34A469   57
7C34A46A   FF3
7C34A470   FF1
7C34A476   EB
7C34A478   834
7C34A47C   893
7C34A47E   897
7C34A481   FF0
7C34A487   8B4
```

```
0012FE6C   7C38B198
0012FE70   7C374013
0012FE74   7C38B198
0012FE78   7C372F4F
0012FE7C   7C350538
0012FE80   00000000
0012FE84   7C3609C6
0012FE88   41414141
0012FE8C   7C38B198
0012FE90   41414141
0012FE94   7C36C019
0012FE98   7C34A459
0012FE9C   7C3425CE
0012FEA0   00000200
0012FEA4   00001000
0012FEA8   00000040
0012FEAC   41414141
```

```
ernel32.VirtualAlloc
cdll.RtlFreeHeap
```

VirtualAlloc

Args space

ROP

几种方式：

*直接调用系统代码段实施ROP攻击

*先调用关闭DEP保护函数再利用内存代码段的组合进行的攻击

*调用系统关键函数(如WinExec)实施的攻击

ROP

可能用到的API有：

先关闭DEP保护再进行攻击可能用到的API有：
VirtualAlloc(), HeapCreate(), SetProcessDEPPolicy(), NtSetInformationProcess(), VirtualProtect(), WriteProtectMemory()等
调用系统关键函数进行攻击可能用到的API有：
WinExec等

EMET

微软Enhanced **M**itigation **E**xperience**T**oolkit
增强减灾体验工具

EMET

EMET Security Mitigations	Included
Attack Surface Reduction (ASR) Mitigation	✓
Export Address Table Filtering (EAF+) Security Mitigation	✓
Data Execution Prevention (DEP) Security Mitigation	✓
Structured Execution Handling Overwrite Protection (SEHOP) Security Mitigation	✓
NullPage Security Mitigation	✓
Heapspray Allocation Security Mitigation	✓
Export Address Table Filtering (EAF) Security Mitigation	✓
Mandatory Address Space Layout Randomization (ASLR) Security Mitigation	✓
Bottom Up ASLR Security Mitigation	✓
Load Library Check – Return Oriented Programming (ROP) Security Mitigation	✓
Memory Protection Check – Return Oriented Programming (ROP) Security Mitigation	✓
Caller Checks – Return Oriented Programming (ROP) Security Mitigation*	✓
Simulate Execution Flow – Return Oriented Programming (ROP) Security Mitigation*	✓
Stack Pivot – Return Oriented Programming (ROP) Security Mitigation	✓

ROP的检测方法

1.StackPviot:check if stack is pviotted

2.Caller:check if critical functions was called and not returned into

3.SimExecFlow:Simulate the execution flow after the return address to detect subsequent ROP Gadgets

4.MemProt:specail check on memory protections API

5.LoadLib:check and prevent LoadLibrary calls againts UNC paths

EMET

EMET CALLER CHECKS 核心原理

EMET处理了一批关键函数(如VirtualProtect, VirtualAlloc等)，Caller是在运行到关键函数的基础之上进行检测

Caller:check if critical functions was called and not returned into

即检查关键函数的返回地址所指向的指令的前一条指令是否为CALL指令，如果不是则认为检测到ROP

EMET

EMET CALLER CHECKS 核心原理

32位操作系统的四种CALL类型：

①“call xxxxxxxx”形式的间接跳转

②“call AAAABBBBBBBB”形式的直接远跳,其“AAAA”代表16位的段选择子，“BBBBBBBB”代表32位偏移

③“call [内存地址]”形式，opcode为“FF15 [xxxxxxxx]”

④“call far[内存地址]”形式，opcode为“FF1D [xxxxxxxx]”

正常情况下的关键函数调用及返回

```
.   F3:AB              rep     stos dword ptr es:[edi]
.   8BF4               mov     esi, esp
.   8D85 FCFEFFFF      lea     eax, dword ptr [ebp-104]
.   50                 push    eax
.   6A 04              push    4
.   68 00010000        push    100
.   8D8D 00FFFFFF      lea     ecx, dword ptr [ebp-100]
.   51                 push    ecx
.   FF15 4CA14200      call    dword ptr [<&KERNEL32.VirtualProtect>]
.   3BF4               cmp     esi, esp
.   E8 3E000000        call    00401090
.   33C0               xor     eax, eax
```

ROP情况下的关键函数调用及返回

```
7C801AD3   90              7D5A30E2     FFFF          ???
7C801AD4   8BFF            7D5A30E4     D4 32         aam     32
7C801AD6   55              7D5A30E6     5A            pop     edx
7C801AD7   8BEC            7D5A30E7   v 7D 61         jge     short 7D5A314A
7C801AD9   FF75 14         7D5A30E9     FA            cli
7C801ADC   FF75 10         7D5A30EA     FFFF          ???
7C801ADF   FF75 0C         7D5A30EC     E4 CC         in      al, 0CC
7C801AE2   FF75 08         7D5A30EE     5A            pop     edx
7C801AE5   6A FF           7D5A30EF   ^ 7D BC         jge     short 7D5A30AD
7C801AE7   E8 75FFFFFF     7D5A30F1     325A 7D       xor     bl, byte ptr [edx+7D]
7C801AEC   5D              7D5A30F4   v 7F FF         jg      short 7D5A30F5
7C801AED   C2 1000         call VirtualProtectEx
                           pop  ebp
                           retn 10

                           7D5A30EB   - FFE4          jmp     esp
                           7D5A30ED     CC            int3
                           7D5A30EE     5A            pop     edx
                           7D5A30EF   ^ 7D BC         jge     short 7D5A30AD
                           7D5A30F1     325A 7D       xor     bl, byte ptr [edx+7D]
                           7D5A30F4   v 7F FF         jg      short 7D5A30F5
                           7D5A30F6     FFFF          ???
                           7D5A30F8     BC 325A7D7F   mov     esp, 7F7D5A32
                           7D5A30FD     FFFF          ???
                           7D5A30FF     FFE4          jmp     esp
                           7D5A3101     CC            int3
```

绕过

返回地址的上方是一个call的形式

```
7D72E0E5   54              push    esp
7D72E0E6   5D              pop     ebp
7D72E0E7   C2 0400         retn    4
```

```
7C809AE3   55              push    ebp
7C809AE4   8BEC            mov     ebp, esp
7C809AE6   FF75 14         push    dword ptr [ebp+14]
7C809AE9   FF75 10         push    dword ptr [ebp+10]
7C809AEC   FF75 0C         push    dword ptr [ebp+C]
7C809AEF   FF75 08         push    dword ptr [ebp+8]
7C809AF2   6A FF           push    -1
7C809AF4   E8 09000000     call    VirtualAllocEx
7C809AF9   5D              pop     ebp
7C809AFA   C2 1000         retn    10
```

```
7D72E0E0   8B08            mov     ecx, dword ptr [eax]
7D72E0E2   50              push    eax
7D72E0E3   FF51 54         call    dword ptr [ecx+54]
7D72E0E6   5D              pop     ebp
7D72E0E7   C2 0400         retn    4
```

微软CFG控制流保护

利用bitmap记录函数起始地址

存在的问题：

为考虑性能， 进行缓解措施

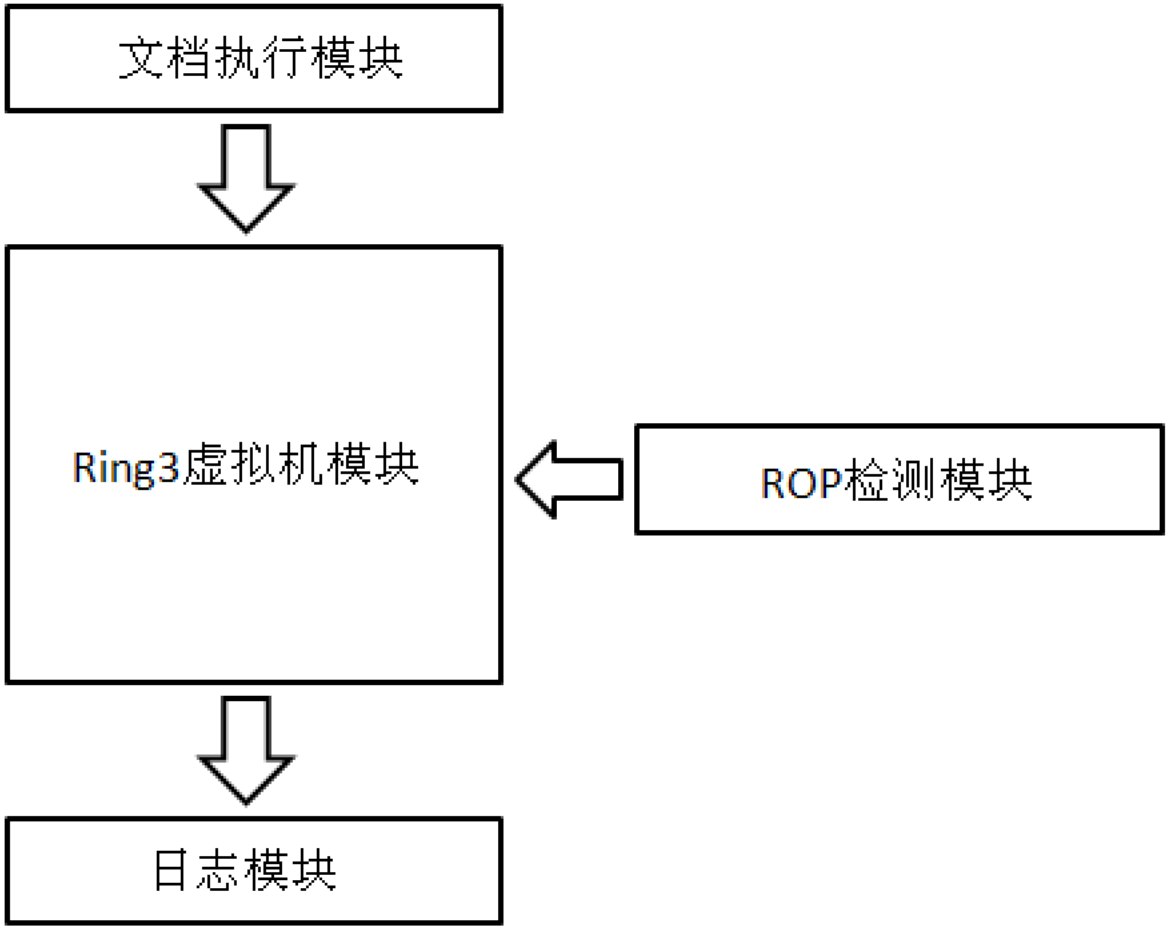
操作系统关联， 程序编译关联

使用插桩实现ROP检测

防御/检测

插桩实现ROP检测

原理图



CFI实现与改进

性能消耗较大

使用场景： 动态沙盒，虚拟机，非加壳程序

动态二进制插桩

动态二进制插桩原理：

“just in time” compiler

动态二进制插桩框架有:

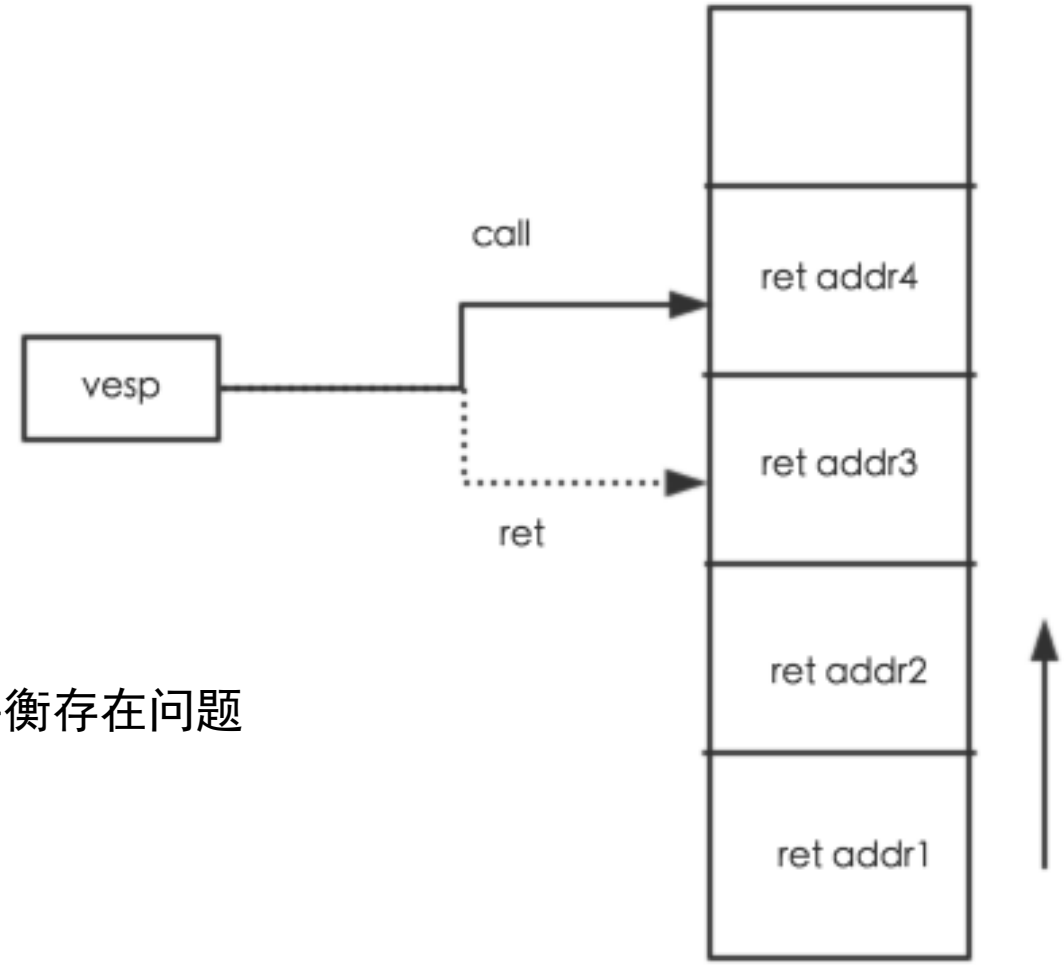
Pin ,DynamoRIO , Valgrind等

虚拟机堆栈检测ROP流程

1. 创建虚拟堆栈，用于记录调用CALL函数的地址，
2. 执行过程中记录RETN的返回地址，
3. RETN的返回地址和表中上个记录的CALL地址进行校验，
4. 即校验两者是否在一定范围之内，
5. 校验成功则删除虚拟堆栈中的地址，
6. 否则认为检测到ROP

虚拟机堆栈检测ROP

虚拟堆栈



出现虚拟堆栈不平衡存在问题

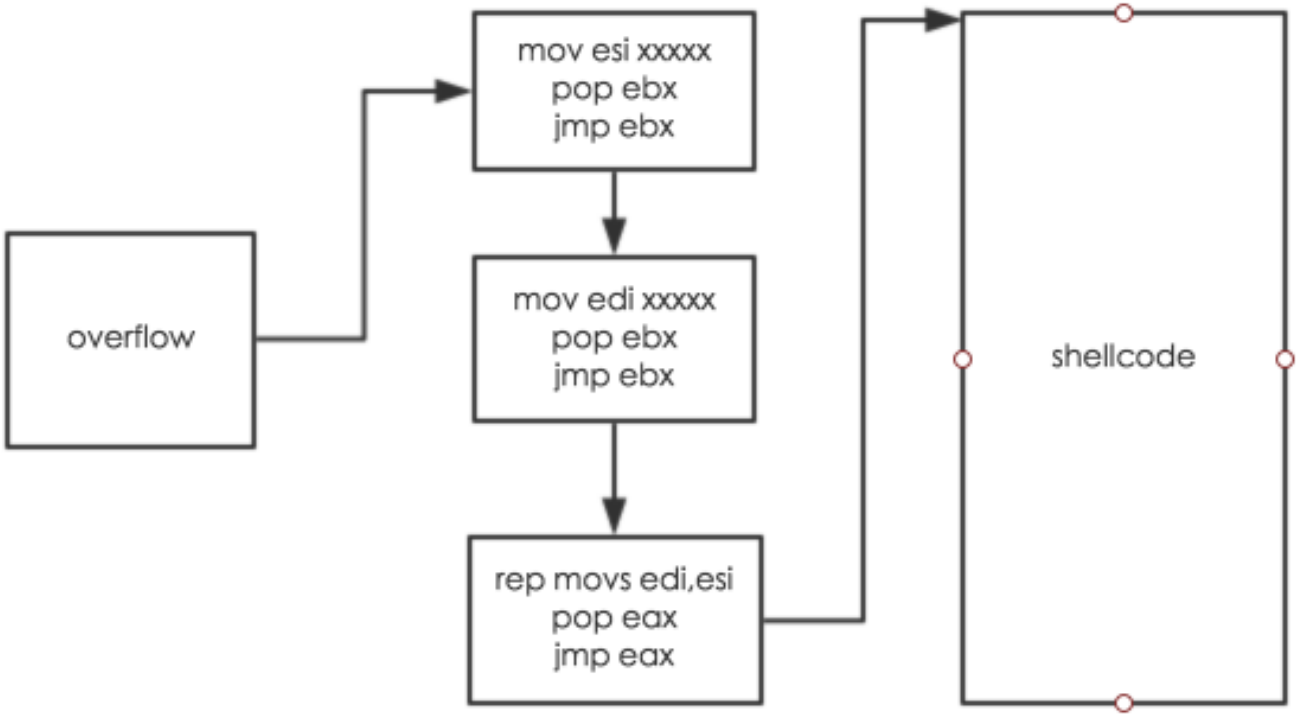
虚拟机堆栈

Ret 检测， 只要堆栈不平衡，就出产生告警， 但是无法防护JOP

JOP攻击

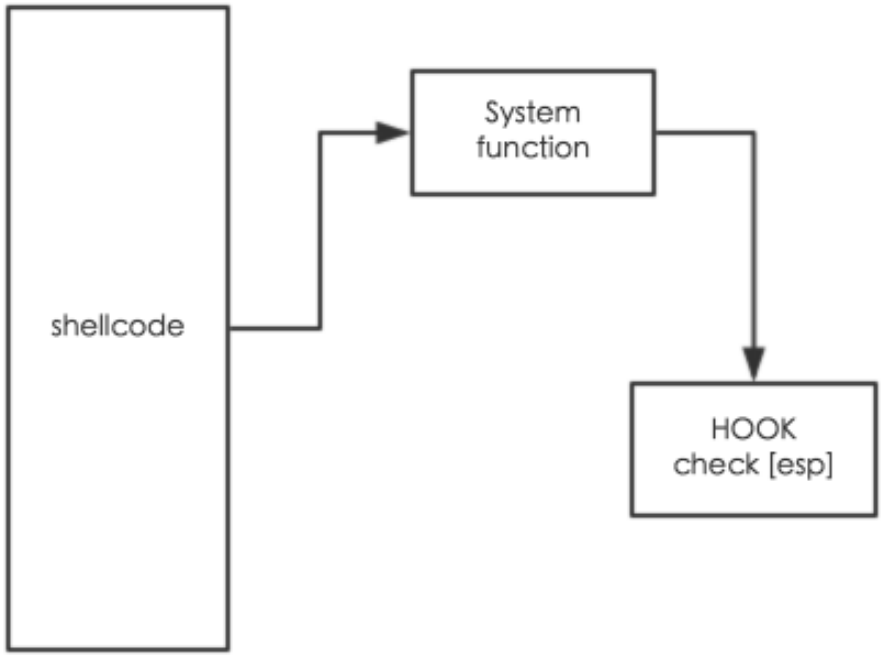
1. Jmp攻击/Call攻击
2. 无法直接调用系统API
3. 需要使用shellcode
4. 只能改写已有的堆空间

同时满足3个条件，
才能成功利用



堆栈的反向执行检测

1. 针对JMP进行插桩，性能消耗太大
2. 在系统关键函数插桩
3. 检测返回地址是否在堆中，如果存在在堆栈中

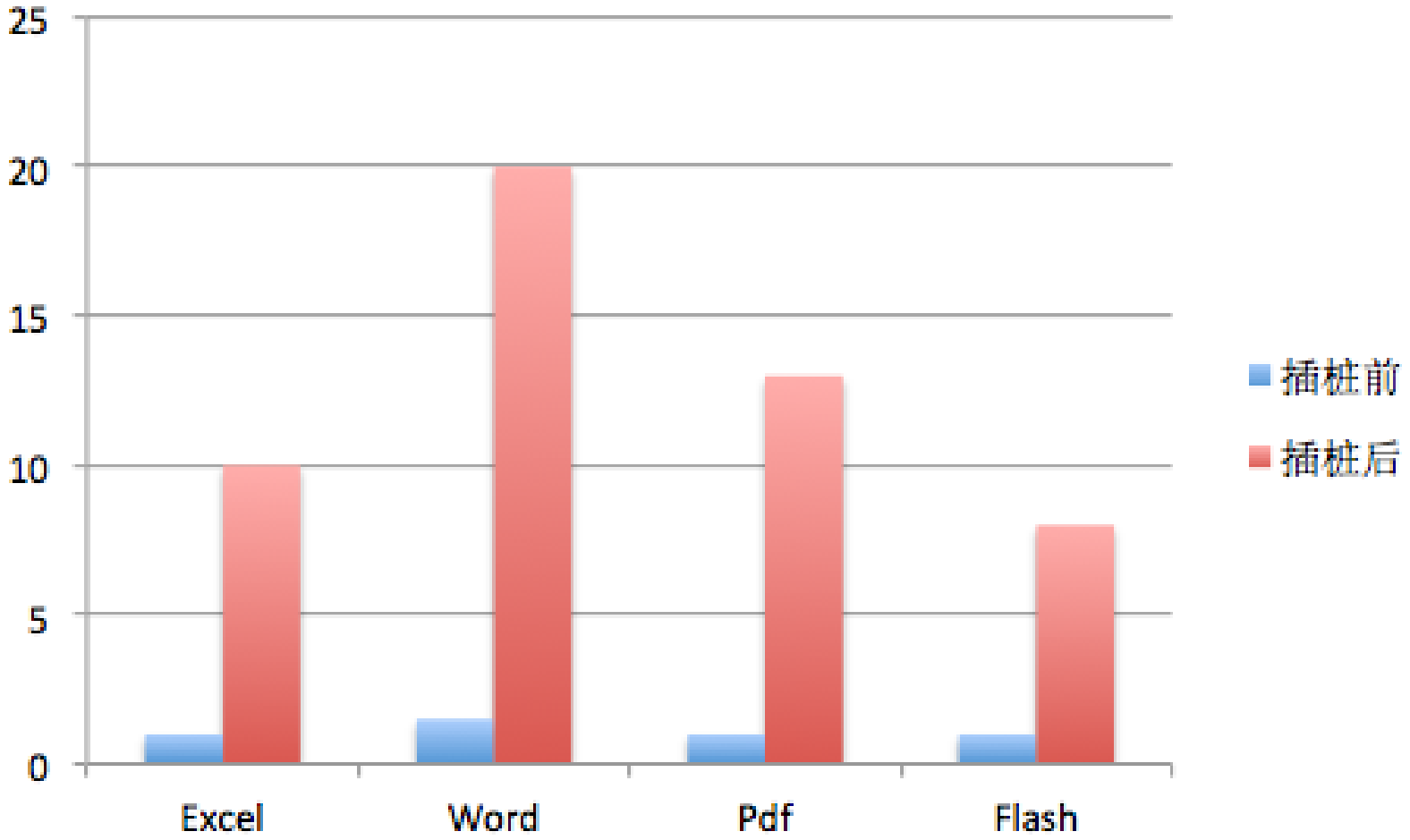


ROP检测

只要能找到合适的代码片段，rop总是可以绕过各种防护

虽然还存在问题，但是绕过的难度已经非常大了

性能



优化

过滤系统函数库

针对OFFICE 、Adobe Reader等软件做特殊处理，去除一些特殊的CALL

加载文件时进行动态插桩，减少性能消耗

缓存代高频率码片段

实例

该实例为memcpy()溢出实例，地址0x0040D496的 call 0040100A内即为memcpy()的拷贝过程，地址0x0040105A为执行完memcpy()后，对call 0040100A的返回

```
0040D496  .  E8 6F3BFFFF  call  0040100A
0040D49B  .  5F           pop   edi
0040D49C  .  5E           pop   esi
```

```
00401057  .  8BE5         mov   esp, ebp
00401059  .  5D           pop   ebp
0040105A  L. C3           retn
0040105B     CC           int3
返回到 7C34402E
```

```
Ret  After : IP = 7c34402e
Call Before: IP = 40d496
ROP detected!
```

谢谢！