

# Sentinel

by Nicolas A. Economou

# Intro

## - Why Sentinel ?

- Hundreds of 0-day **binary bugs** appear per year
- Many Antivirus can't stop 0-day exploits
- **AntiXploit tools** are relatively new
- The best **knowledge** comes from exploit writers

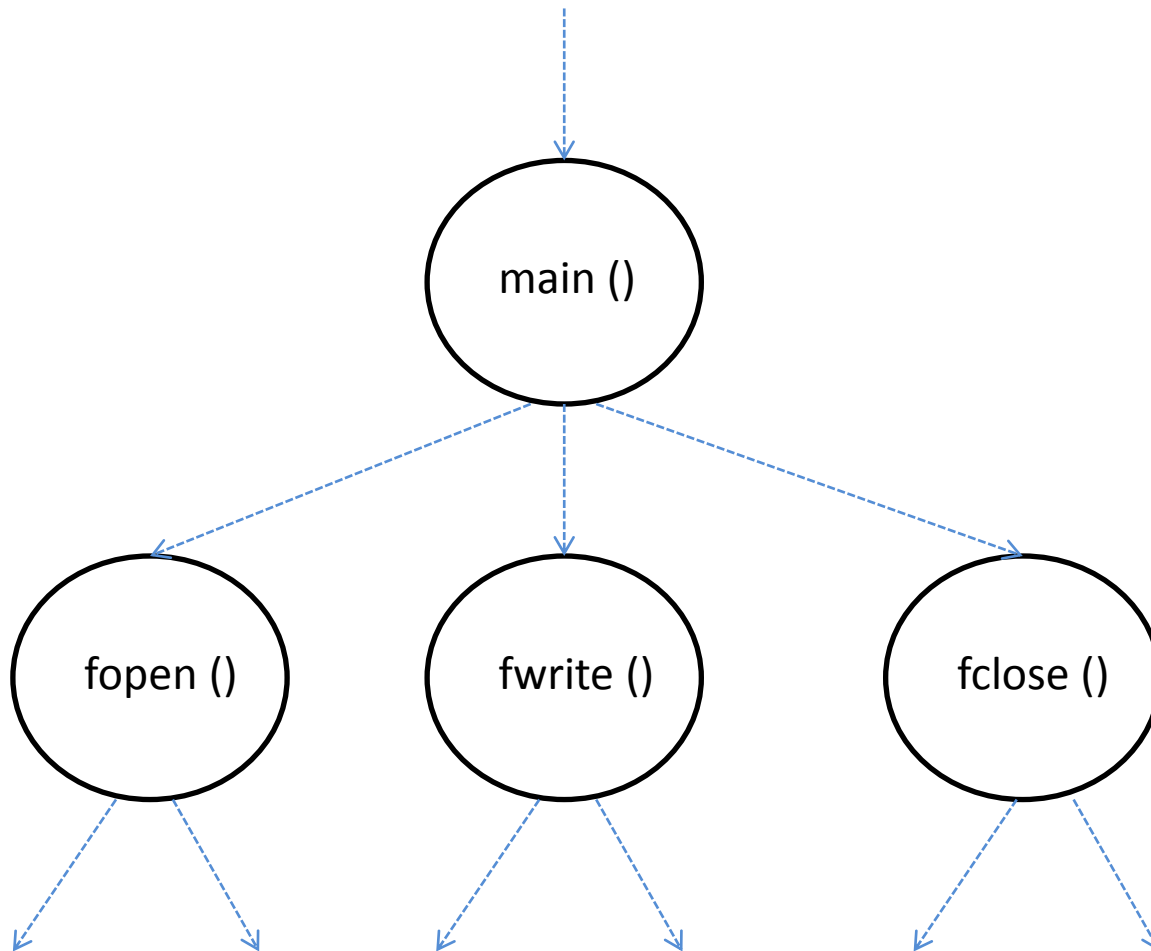
**Let's start ...**

# Calling functions

- E.g. a “C” program

```
main ( ... )  
{  
    printf ( ... );  
    fopen ( ... );  
    fwrite ( ... );  
    fclose ( ... );  
}
```

# Callgraph



# Calling functions

- E.g. a “C” program

→ main ()

→ printf ()

← printf ()

→ fopen ()

← fopen ()

→ fwrite ()

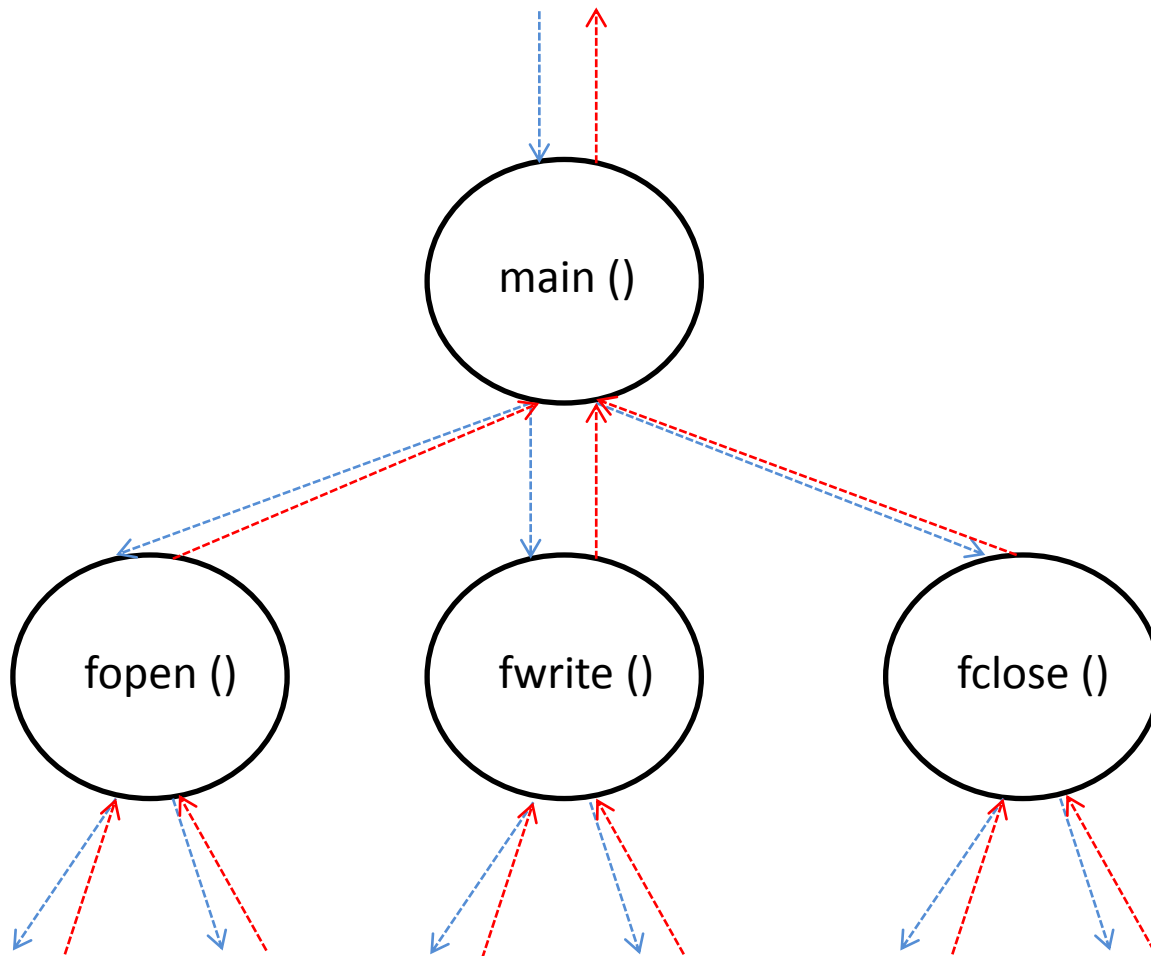
← fwrite ()

→ fclose ()

← fclose ()

← main ()

# Callgraph



# Breaking rules

- E.g. a “C” program

→ main ()

→ printf ()

← printf ()

→ fopen ()

← fopen ()

→ fwrite ()

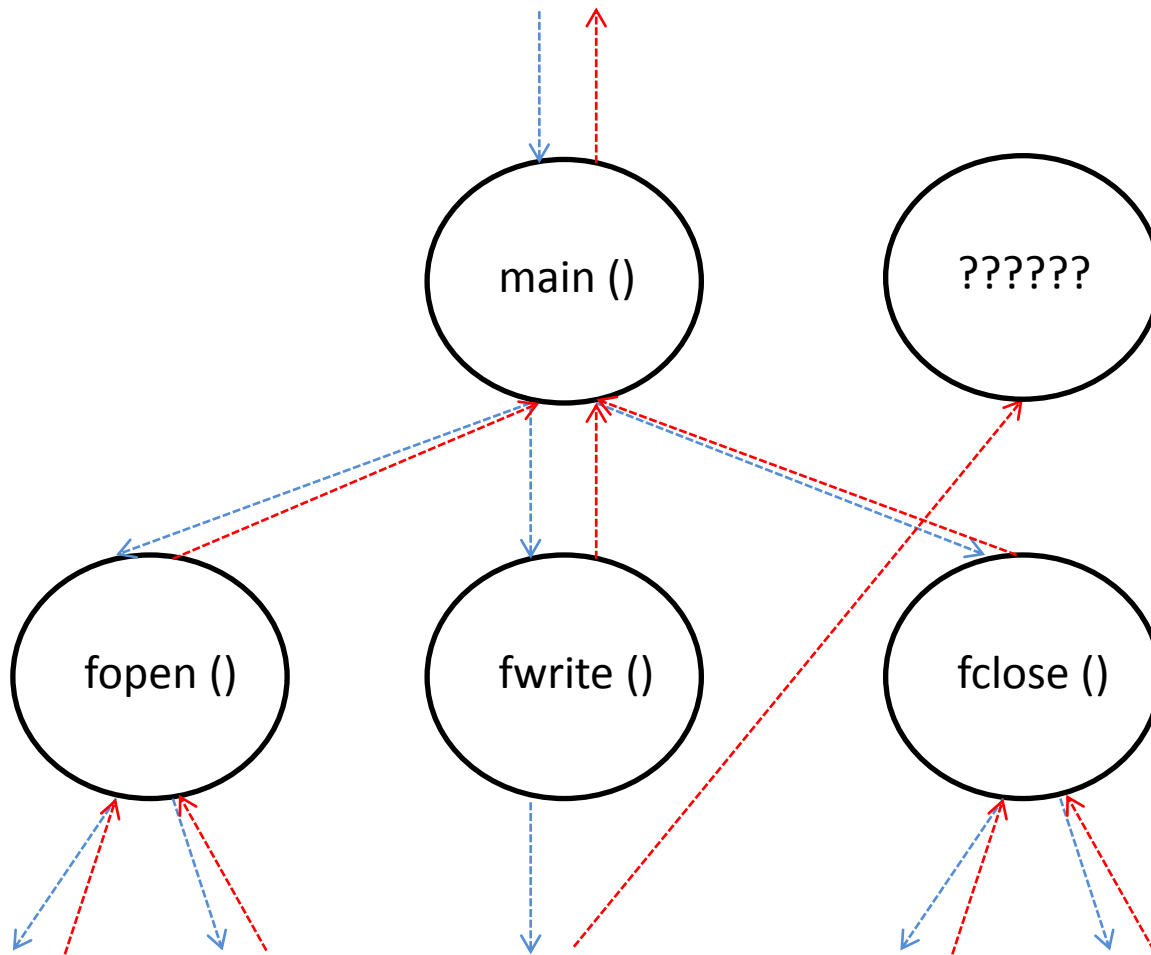
← system () or WinExec () or 0x41414141 ??????????????????

→ ...

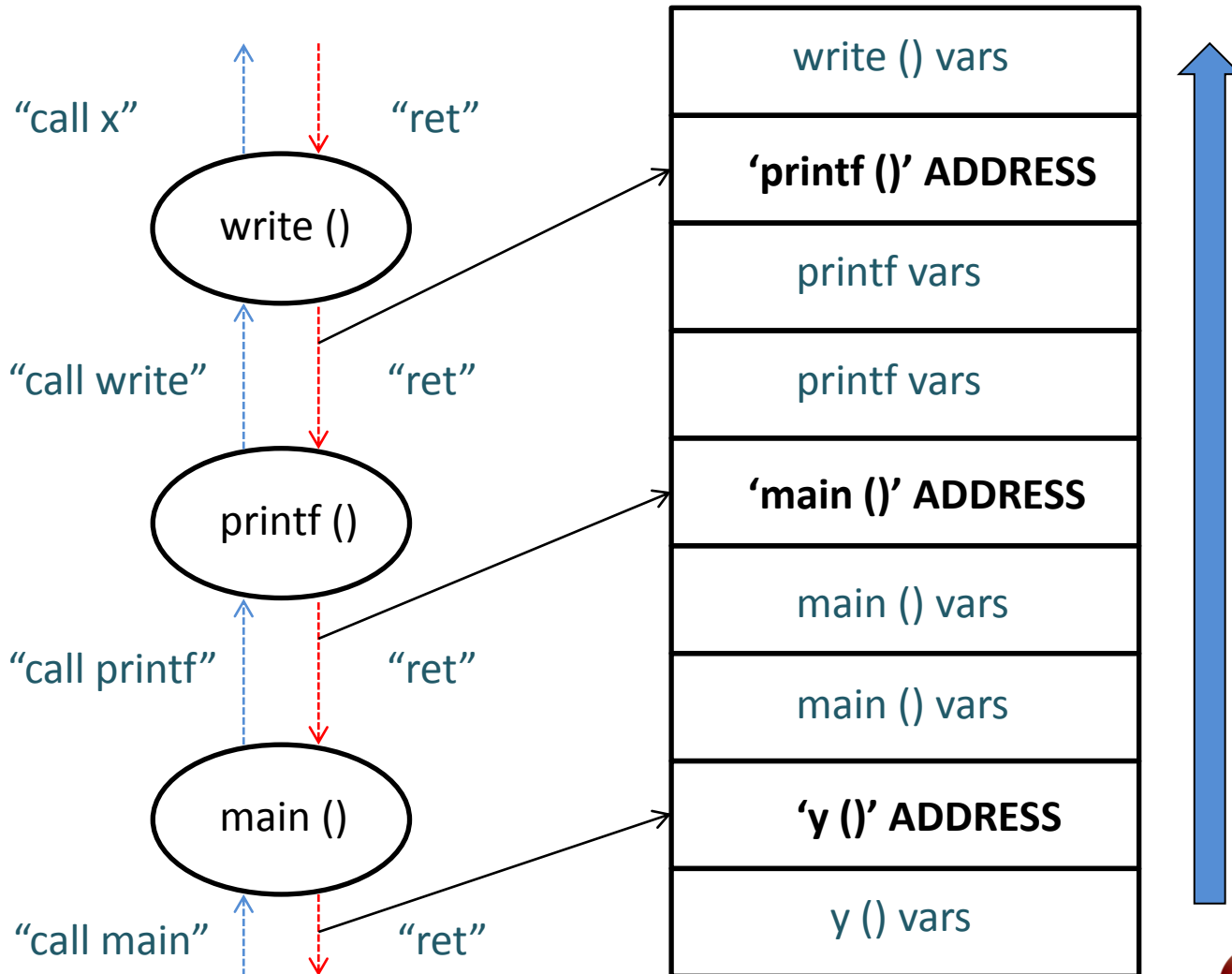
← main ()



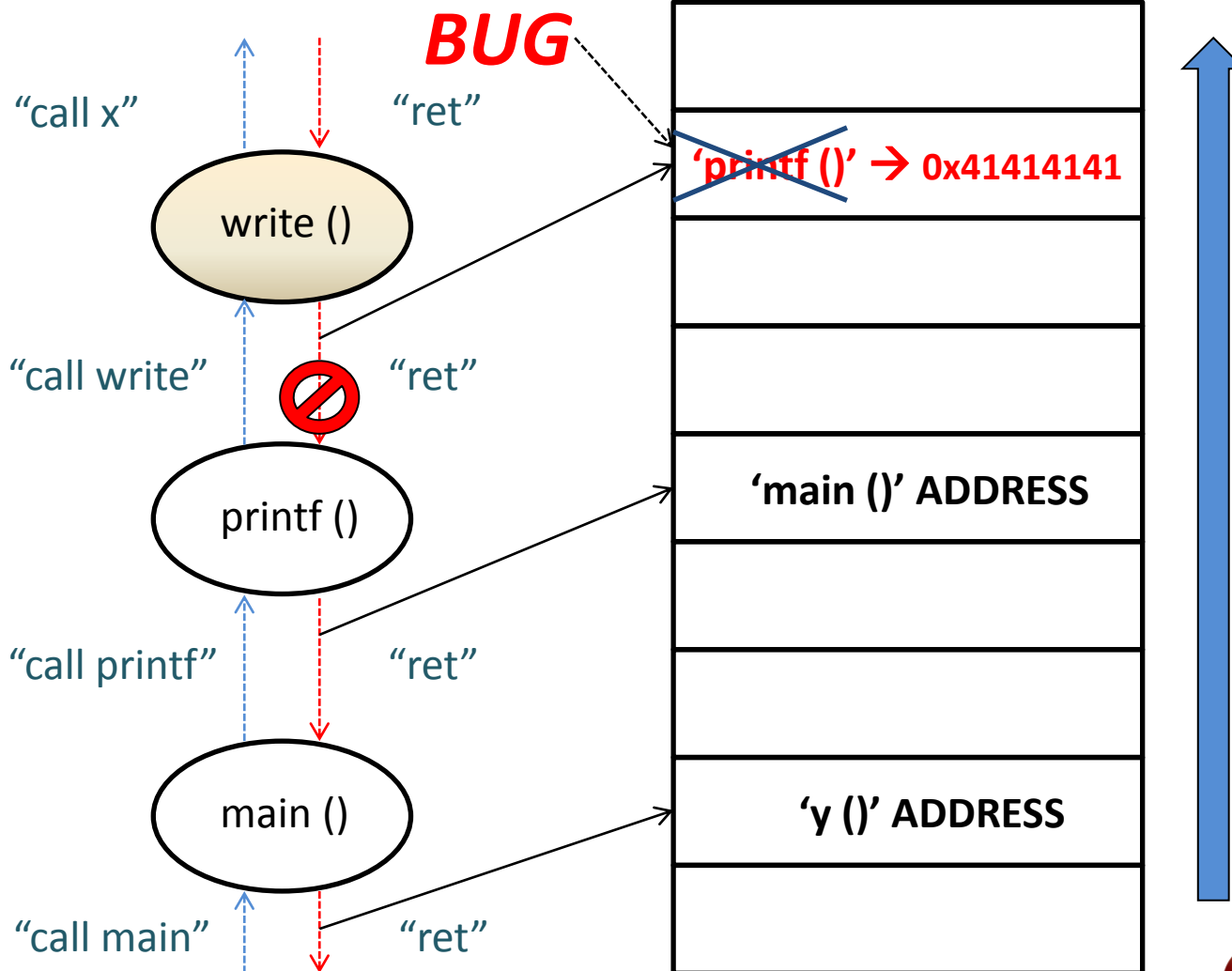
# Breaking rules



# Stack working



# Return address modified



# ROP

- Return-oriented programming
- Used by the most current exploits
- It needs **HARDCODED addresses** or **offsets**
- Usually the STACK is used to **reach** code execution

# ROP

- The idea is to:
  - Bypass protection mechanisms ( **unprotect** stack or heap )
  - Execute a new program ( e.g. “\\myip\rootkit.exe” )

## Targeted functions:

- VirtualProtect – VirtualProtectEx - ZwProtectVirtualMemory
- VirtualAlloc - VirtualAllocEx - ZwAllocateVirtualMemory
- ZwSetInformationProcess
- CreateProcess – CreateProcessEx
- LoadLibrary

## -Final target:

- **Code execution** ( to take control of the target )

# ROP

- In most cases, the “**RET**” instruction is required in each step
- Generally, “**RET**” instructions are located in function **EPILOGUES**.
- An **EPILOGUE** is the last part of a function
  - It's used to:
    - Check the canary ( if it exists )
    - Restore some registers
    - Return to the **PARENT** function

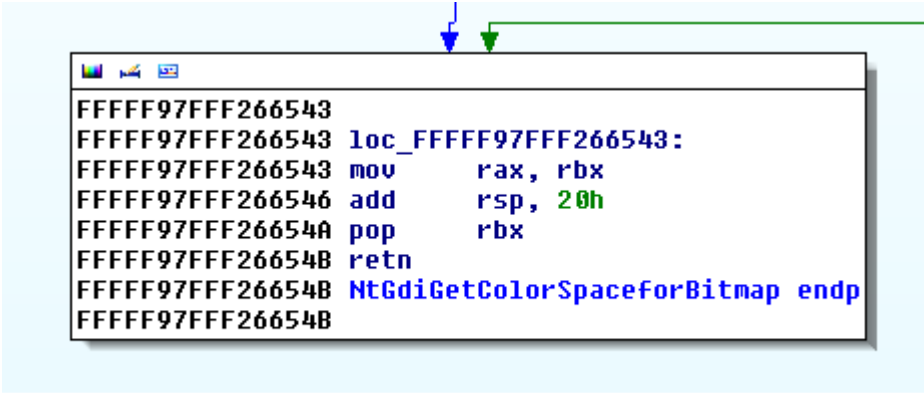
# ROP

- Epilogue example:
  - Intel 32 bits - Windows

```
7C86266A  
7C86266A          loc_7C86266A:  
7C86266A 5F          pop     edi  
7C86266B 5E          pop     esi  
7C86266C 5B          pop     ebx  
7C86266D C9          leave  
7C86266E C2 08 00    retn   8  
7C86266E          __stdcall WinExec(x, x) endp  
7C86266E
```

# ROP

- Epilogue example:
  - Intel 64 bits - Windows

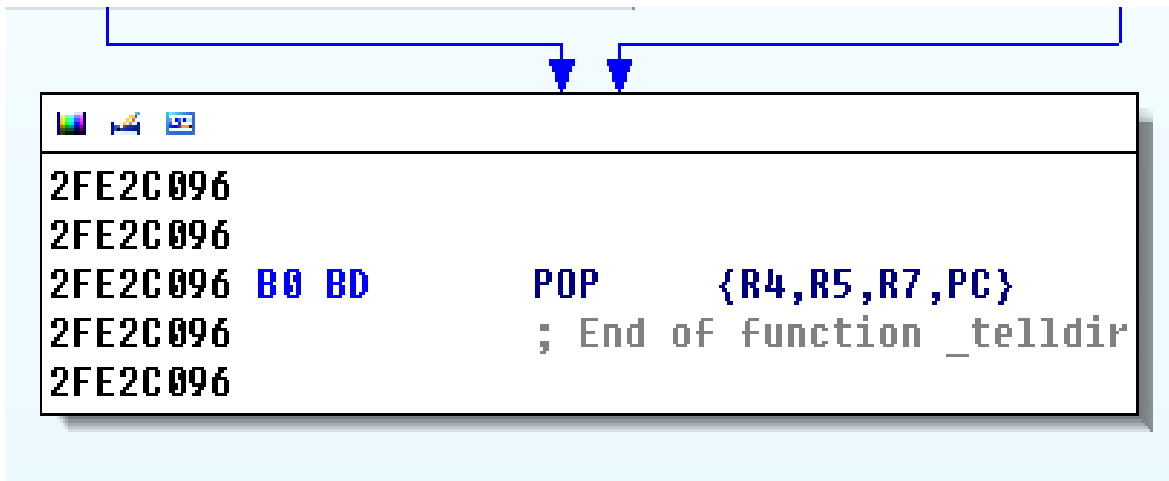


```
FFFFF97FFF266543  
FFFFF97FFF266543 loc_FFFFF97FFF266543:  
FFFFF97FFF266543 mov     rax, rbx  
FFFFF97FFF266546 add     rsp, 20h  
FFFFF97FFF26654A pop     rbx  
FFFFF97FFF26654B retn  
FFFFF97FFF26654B NtGdiGetColorSpaceforBitmap endp  
FFFFF97FFF26654B
```



# ROP

- Epilogue example:
  - ARM - iPhone



```
2FE2C096  
2FE2C096  
2FE2C096 B0 BD      POP      {R4,R5,R7,PC}  
2FE2C096          ; End of function _telldir  
2FE2C096
```

# ROP

- A “subgroup” of the instructions contained in an **EPILOGUE** can be called **GADGET**
- **GADGETS** are little operations used to set register/memory values
- Sometimes it’s necessary to **chain** many **GADGETS** to reach the final target

# ROP

- GADGET examples:
  - “xchg eax,ebx”
  - “pop esi”
  - “pop edi”
  - “**ret**”
  
  - “xor eax,eax”
  - “leave”
  - “**ret** 4”

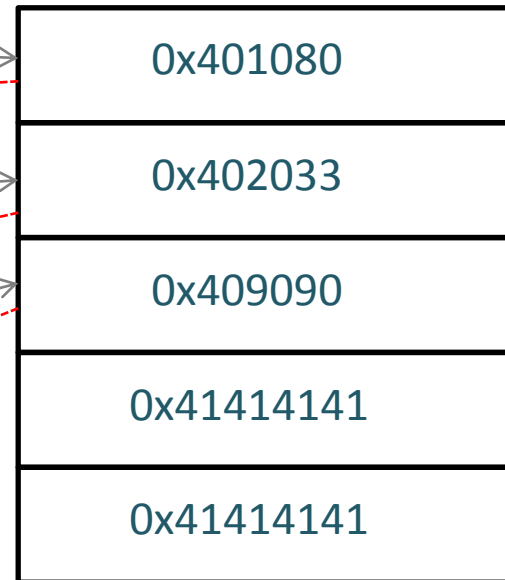
# ROP example

**0x123456:**

"pop eax"  
"pop ebx"  
"ret"

Objective: EAX=1, EBX=2, ECX=3

## CONTROLABLE STACK



0x401080: ←  
"mov eax,1"  
"ret"

0x402033: ←  
"mov ebx,2"  
"ret"

0x409090: ←  
"mov ecx,3"  
"ret"

# The Sentinel Project

# Sentinel

- Command line **TOOL** able to **INTERCEPT exploit attacks** (usually with ROP activity)
- It can **protect**:
  - **Running** programs
  - Launch a program and protect it
- It **RUNS** completely in **USER MODE**

# Sentinel

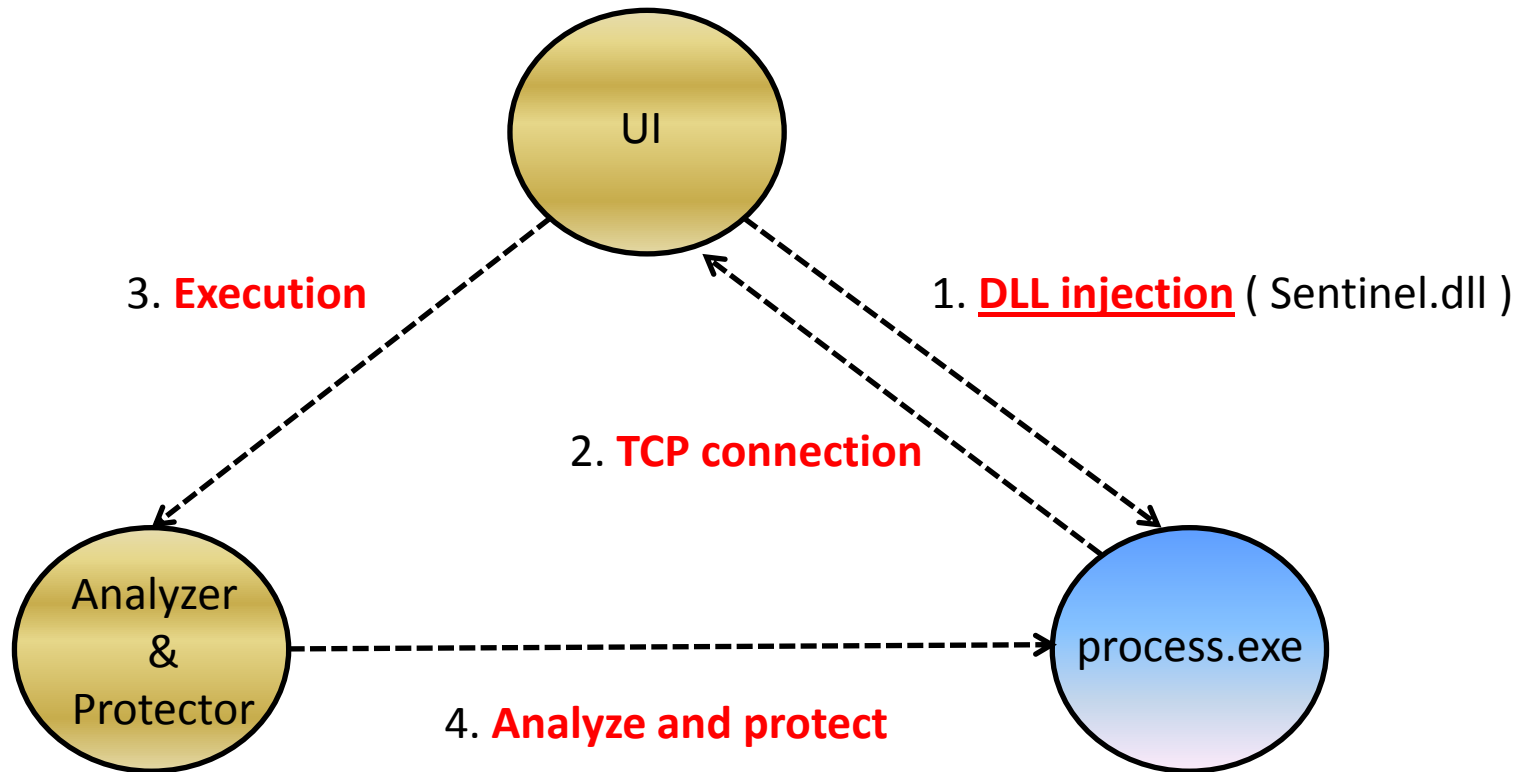
- It **doesn't need feedback** about the module to protect
- It has 2 level protections:
  - Default: Minimal ( only “critical” functions )
  - Custom: Default + targeted modules
- Written in C/C++/ASM
- Download: <http://corelabs.coresecurity.com> ( find Sentinel )

# Similar Projects

- EMET ('Enhanced Mitigation Experience Toolkit')
  - Author: Microsoft
  
- Anti-Exploit
  - Author: Malwarebytes



# Sentinel working scheme



# Sentinel analyzer

- Read the process memory
  - Using “ReadProcessMemory” function
- Disassemble the “.text” section
  - Using “Distorm” lib
- Program function detection
  - Identifying **CALLs** to functions
  - Identifying function **prologues**
  - Listing **exported** functions

# Sentinel analyzer

- Function analysis
  - Detect **prologues** and **epilogues**
- Basic block detection:
  - Conditional JUMPs ( “je”, “jne”, “jg”, “ja”, etc )
  - Unconditional JUMPs ( “jmp” )
  - Jump tables ( “jmp [addr+reg\*4]” )

# Disassembled function

```
004011AD ; int __cdecl sub_4011AD(int, char *buf, int len)
004011AD sub_4011AD      proc near                               ; CODE XREF
004011AD                                                    ; funcion_
004011AD
004011AD arg_0          = dword ptr 8
004011AD buf           = dword ptr 0Ch
004011AD len          = dword ptr 10h
004011AD
004011AD          push    ebp
004011AE          mov     ebp, esp
004011B0          push    ebx
004011B1          mov     eax, [ebp+arg_0]
004011B4          xor     ebx, ebx
004011B6          cmp     dword ptr [eax], 1
004011B9          jnz    short loc_4011E8
004011BB          push    0 ; flags
004011BD          push    [ebp+len] ; len
004011C0          push    [ebp+buf] ; buf
004011C3          push    dword ptr [eax+4] ; s
004011C6          call   send
004011CB          inc     eax
004011CC          jz     short loc_4011D5
004011CE          mov     ebx, 1
004011D3          jmp    short loc_4011E8
004011D5 ; -----
004011D5
004011D5 loc_4011D5: ; CODE XREF
004011D5          call   WSAGetLastError
004011DA          push    eax
004011DB          push    offset format ; "E1 codi
004011E0          call   _printf
004011E5          add     esp, 8
004011E8
004011E8 loc_4011E8: ; CODE XREF
004011E8 ; sub_4011
004011E8          mov     eax, ebx
004011EA          pop     ebx
004011EB          pop     ebp
004011EC          retn
004011EC sub_4011AD      endp
```

# Flowchart function

## PROLOGUE

```
004011AD
004011AD
004011AD ; Attributes: bp-based frame
004011AD
004011AD ; int __cdecl sub_4011AD(int, char *buf, int len)
004011AD sub_4011AD proc near
004011AD
004011AD arg_0= dword ptr 8
004011AD buf= dword ptr 0Ch
004011AD len= dword ptr 10h
004011AD
004011AD 55          push    ebp
004011AE 8B EC      mov     ebp, esp
004011B0 53          push    ebx
004011B1 8B 45 08   mov     eax, [ebp+arg_0]
004011B4 33 DB      xor     ebx, ebx
004011B6 83 38 01   cmp    dword ptr [eax], 1
004011B9 75 2D      jnz    short loc_4011E8
```

```
004011BB 6A 00      push    0 ; flags
004011BD FF 75 10   push    [ebp+len] ; len
004011C0 FF 75 0C   push    [ebp+buf] ; buf
004011C3 FF 70 04   push    dword ptr [eax+4] ; s
004011C6 E8 A5 8C 00 00 call    send
004011CB 40          inc     eax
004011CC 74 07      jz     short loc_4011D5
```

```
mov     ebx, 1
jmp     short loc_4011E8
```

```
004011D5
004011D5 loc_4011D5:
004011D5 E8 60 8C 00 00 call    WSAGetLastError
004011DA 50          push    eax
004011DB 68 88 A0 40 00 push    offset format
004011E0 E8 9B 2A 00 00 call    _printf
004011E5 83 C4 08   add     esp, 8
```

## EPILOGUE

```
004011E8
004011E8 loc_4011E8:
004011E8 8B C3      mov     eax, ebx
004011EA 5B          pop     ebx
004011EB 5D          pop     ebp
004011EC C3          retn
004011EC sub_4011AD endp
004011EC
```

# Sentinel protector

- Protect the **chosen** module/modules
  - Using “WriteProcessMemory” function
- Use the analyzer information
- Choose **PATCHABLE** functions
- Create the **STUB** area

# Sentinel protector

- Patchable function requirements:
  - Prologue  $\geq$  5 bytes
  - All Epilogues  $\geq$  5 bytes
  - **Shared** epilogues only with **VALID prologues**
  - Epilogues only finishing with “**RET/RET n**” instruction
  - more ...

# E.g. Patched Prologue

```
ntsd calc
kernel32!VirtualProtectEx:
7c801a61 8bff      mov     edi,edi
7c801a63 55       push   ebp
7c801a64 8bec     mov     ebp,esp
7c801a66 56       push   esi
7c801a67 8b35c412807c mov    esi,[kernel32+0x12c4 (7c8012c4)]
7c801a6d 57       push   edi
7c801a6e ff7518   push   dword ptr [ebp+0x18]
7c801a71 8d4510   lea    eax,[ebp+0x10]
0:000>
```

```
ntsd calc
kernel32!VirtualProtectEx:
7c801a61 e9d9e57a84 jmp    00fb003f
7c801a66 56       push   esi
7c801a67 8b35c412807c mov    esi,[kernel32+0x12c4 (7c8012c4)]
7c801a6d 57       push   edi
7c801a6e ff7518   push   dword ptr [ebp+0x18]
7c801a71 8d4510   lea    eax,[ebp+0x10]
7c801a74 ff7514   push   dword ptr [ebp+0x14]
7c801a77 50       push   eax
0:001>
```



# E.g. Patched Prologue

```
ntsd calc
kernel32!VirtualProtectEx:
7c801a61 e9d9e57a84 jmp 00fb003f
7c801a66 56 push esi
7c801a67 8b35c412807c mov esi,[kernel32+0x12c4 (7c8012c4)]
7c801a6d 57 push edi
7c801a6e ff7518 push dword ptr [ebp+0x18]
7c801a71 8d4510 lea eax,[ebp+0x10]
7c801a74 ff7514 push dword ptr [ebp+0x14]
7c801a77 50 push eax
0:001>
```

```
ntsd calc
0:001> u 00fb003f
00fb003f 68611a807c push 0x7c801a61
00fb0044 e83b13a6ff call Sentinel+0x1384 (00a11384)
00fb0049 83c404 add esp,0x4
00fb004c 8bff mov edi,edi
00fb004e 55 push ebp
00fb004f 8bec mov ebp,esp
00fb0051 e9101a857b jmp kernel32!VirtualProtectEx+0x5 (7c801a66)
00fb0056 e8b79a857b call kernel32!VirtualAllocEx (7c809b12)
0:001>
```

**PROLOGUE checker**

**Original instructions**

# E.g. Patched Prologue

```
ntsd calc
0:001> u 00fb003f
00fb003f 68611a807c      push    0x7c801a61
00fb0044 e83b13a6ff      call   Sentinel+0x1384 (00a11384)
00fb0049 83c404         add    esp,0x4
00fb004c 8bff         mov    edi,edi
00fb004e 55         push  ebp
00fb004f 8bec         mov    ebp,esp
00fb0051 e9101a857b      jmp    kernel32!VirtualProtectEx+0x5 (7c801a66)
00fb0056 e8b79a857b      call  kernel32!VirtualAllocEx (7c809b12)
0:001>
```

```
ntsd calc
kernel32!VirtualProtectEx:
7c801a61 e9d9e57a84      jmp    00fb003f
7c801a66 56         push  esi
7c801a67 8b35c412807c    mov    esi,[kernel32+0x12c4 (7c8012c4)]
7c801a6d 57         push  edi
7c801a6e ff7518         push  dword ptr [ebp+0x18]
7c801a71 8d4510         lea   eax,[ebp+0x10]
7c801a74 ff7514         push  dword ptr [ebp+0x14]
7c801a77 50         push  eax
0:001>
```

# E.g. Patched Epilogue

```
ntsd calc
kernel32!VirtualProtectEx+26:
7c801a87 33c0      xor     eax, eax
7c801a89 40        inc     eax
7c801a8a 5f        pop     edi
7c801a8b 5e        pop     esi
7c801a8c 5d        pop     ebp
7c801a8d c21400    ret     0x14
7c801a90 81ff450000c0  cmp     edi, 0xc0000045
7c801a96 752d     jnz     kernel32!VirtualProtectEx+0x64 (7c801ac5)
0:000>
```

```
ntsd calc
kernel32!VirtualProtectEx+26:
7c801a87 33c0      xor     eax, eax
7c801a89 40        inc     eax
7c801a8a 5f        pop     edi
7c801a8b e99de57a84 jmp     00fb002d
7c801a90 81ff450000c0  cmp     edi, 0xc0000045
7c801a96 752d     jnz     kernel32!VirtualProtectEx+0x64 (7c801ac5)
7c801a98 837d08ff  cmp     dword ptr [ebp+0x8], 0xffffffff
7c801a9c 7527     jnz     kernel32!VirtualProtectEx+0x64 (7c801ac5)
0:001>
```

# E.g. Patched Epilogue

```
ntsd calc
kernel32!VirtualProtectEx+26:
7c801a87 33c0      xor     eax, eax
7c801a89 40        inc     eax
7c801a8a 5f        pop     edi
7c801a8b e99de57a84 jmp     00fb002d
7c801a90 81ff45000c0 cmp    edi, 0xc0000045
7c801a96 752d      jnz    kernel32!VirtualProtectEx+0x64 (7c801ac5)
7c801a98 837d08ff  cmp    dword ptr [ebp+0x8], 0xffffffff
7c801a9c 7527      jnz    kernel32!VirtualProtectEx+0x64 (7c801ac5)
0:001>
```

```
ntsd calc
0:001> u 00fb002d
00fb002d 5e        pop     esi
00fb002e 5d        pop     ebp
00fb002f 688a1a807c push   0x7c801a8a
00fb0034 e89f13a6ff call   Sentinel+0x13d8 (00a113d8)
00fb0039 83c404    add     esp, 0x4
00fb003c c21400    ret    0x14
00fb003f 68611a807c push   0x7c801a61
00fb0044 e83b13a6ff call   Sentinel+0x1384 (00a11384)
0:001>
```

Original instructions

EPILOGUE checker

# Intercepting exploit attacks

# Sentinel check mechanisms

- Parallel STACK
- PARENT CALL checker
- STACK checker
- EPILOGUE replacing ...

# Mechanism: Parallel STACK

- Inspired on **CANARY** checking
- When a protected function is **called**
  - Some values are **saved** in the function **PROLOGUE**
- When a protected function is **ending**
  - The **saved** values are **compared** with the **CURRENT VALUES** in the **EPILOGUE**

# Mechanism: Parallel STACK

- If a **difference** exists, it could be:
  - **STACK OVERFLOW**
  - **MEMORY CORRUPTION**
  - **ROP activity !!!!!!!**
  - Problematic functions:
    - “SEH handling”, “canary checks”, “alloca\_probe”, etc



# Mechanism: Parallel STACK

## - Details:

- Area allocated by **VirtualAlloc**
- 4 kb per **THREAD** (to be improved)
- Support up to 256 **logs** (entries)
- Pointer to the area located in the **TEB**

# Mechanism: Parallel STACK

- Values to **save** and **check** (entry):
  - **Base pointer** register (EBP)
  - **Stack pointer** register (ESP)
  - **Return address**





# Mechanism: Parallel STACK

With  
Sentinel working  
&  
ROP activity!

```

004011AD
004011AD
004011AD ; Attributes: bp-based frame
004011AD
004011AD ; int __cdecl sub_4011AD(int, char *buf, int len)
004011AD sub_4011AD proc near
004011AD
004011AD arg_0= dword ptr 8
004011AD buf= dword ptr 0Ch
004011AD len= dword ptr 10h
004011AD
004011AD 55
004011AE 8B EC
004011B0 53
004011B1 8B 45 08
004011B4 33 DB
004011B6 83 38 01
004011B9 75 2D
mov     eax, [ebp+arg_0]
xor     ebx, ebx
cmp     dword ptr [eax], 1
jnz    short loc_4011E8
    
```

**JUMP Sentinel\_STUB**

NON  
SAVED VALUES

```

004011BB 6A 00      push    0 ; flags
004011BD FF 75 10   push    [ebp+len] ; len
004011C0 FF 75 0C   push    [ebp+buf] ; buf
004011C3 FF 70 04   push    dword ptr [eax+4] ; s
004011C6 E8 A5 8C 00 00 call    send
004011CB 40        inc     eax
004011CC 74 07     jz     short loc_4011D5
    
```

```

mov     ebx, 1
jmp    short loc_4011E8
    
```

```

004011D5
004011D5 loc_4011D5:
004011D5 E8 60 8C 00 00 call    WSAGetLastError
004011DA 50        push   eax
004011DB 68 88 A0 40 00 push   offset format
004011E0 E8 9B 2A 00 00 call    _printf
004011E5 83 C4 08   add     esp, 8
    
```

COMPARE  
AGAINST  
WHAT ???

ROP to EPILOGUE

“ret caller\_function”

```


004011E8
004011E8
004011E8
004011EA
004011EB
004011EC
004011EC
004011EC
004011EC
    
```

**JUMP**  
**Sentinel\_STUB**




# Mechanism: CALLER checker

## Caller point of view:

1. “call function\_A”
  2. “mov eax, value”
  3. ...
- function\_A ()**
- 

## Function point of view (like a linked LIST !):

1. “????????????????”
  2. “mov eax, value”
  3. ...
- function\_A ()**
- 

# Mechanism: CALLER checker

- **Idea:**
  - **Check** if the **CALLER instruction** is **VALID**
- Why ?
  - Usually, **ROP doesn't use CALL instructions**
- Looking **BACK**
  - Using ~~BRANCH TRACING~~ (**BlueHat winner !**)
  - Using **CALLER checking** (**BlueHat 2nd !**)

# Mechanism: CALLER checker

- Idea:
  - Backwards Disassembling from 2 to 8/9 bytes
- If a **VALID** CALL exists
  - The check is **passed**
- Problems:
  - False positives/negatives
  - Slow down the speed of checking performance



# Mechanism: STACK checker

- Stack pivoting:
  - When **ESP** is out of the **VALID STACK RANGE** ( check against the TEB )
  - Usually ESP is pointed to the HEAP
  - It's very common in **browser exploitations**

# Mechanism: STACK checker

- Return address checking:
  - **Detect** if the **RETURN ADDRESS** is in the **STACK** (last **ROP** step or stack execution)
- Stack execution prevention:
  - Set **NX** for each intercepted **THREAD** (useful OLD OSs/OLD applications)

# Mechanism: Epilogue replacing

- Idea ( not implemented yet ! ):
  - Replace **ALL VALID epilogues** by:
    - A **JUMP** to the relocated **EPILOGUE**
    - **Fill** the rest of the bytes with **BREAKPOINTS**
    - Check relocated **EPILOGUES** as well
- When **ROP** is working:
  - A **breakpoint** will be **executed**
  - Or it will **CRASH** ( executing a part of the **new JUMP** )

# Bypassing ?

- Parallel stack:
  - **ROP** to **OUT OF PHASE** gadgets
- Caller check:
  - **ROP** to **VALID CALL** instructions
  - Return address – 1 inst. = **VALID CALL**
- Stack check:
  - **Not** to **return directly** to the **STACK**
  - **Not** to do **CALLs** from the **STACK**

# Bypassing ?

- Stack pivoting:
  - Try to **revert** it and came back to the **VALID STACK** ( very tricky ! )
- Epilogue replacing:
  - **ROP** to **OUT OF PHASE** gadgets

# Demo Time

# Demo 1

- **Performance Test**

- Program: “vlc.exe” ( Media player )

- **Adding protections to:**

- Modules: “vlc.exe”, “kernel32.dll” and “user32.dll”

# Demo 2

- **Jumping to 0x41414141:**
  - Program: “bug.exe”
  - Vector: Remote attack
  - Vulnerability: **STACK OVERFLOW**
  
- **Objective:**
  - Stop the attack protecting “bug.exe” module



# Demo 2 bis

- **Bypassing EMET v4.0:**
  - Program: “bug.exe”
  - Vector: Remote attack
  - Vulnerability: **STACK OVERFLOW**
  
- **Objective:**
  - Execute remotely “calc.exe”

# Demo 3

## - Exploiting:

- Program: “svchost.exe” ( Server service )
- CVE: 2008-4250
- Vector: Remote attack
- Vulnerability: **STACK OVERFLOW**

## - Special comments:

- Used by **Win32/Conficker** ( WORM )

## - Adding protections to:

- Modules: “acgenral.dll” ... ??? ( **netapi32.dll** is vuln ! )

# Demo 4

## - Exploiting:

- Program: “Adobe Reader v9.5” (PDF files)
- CVE: 2013-0640 and 2013-0641
- Vector: Client Side attack ( Remote )
- Vulnerability: **HEAP OVERFLOW**

## - Special comments:

- Filename: “Visaform Turkey.pdf”
- In later versions the Adobe Sandbox **protects** the application

## - Adding protections to:

- Modules: **only patching critical functions used by exploits !!!**

# Demo 5

## - Exploiting:

- Program: “Internet Explorer 8.0”
- CVE: 2013-2551
- Vector: Client Side attack ( Remote )
- Vulnerability: **USER AFTER FREE**

## - Special comments:

- 0-day used in **Pwn2own 2013** competition
- **Used together** with a **kernel** privilege escalation **exploit**

## - Adding protections to:

- Modules: **only patching critical functions used by exploits !!!**

Questions ?