Tutorial: Exploitation techniques

Clement "Hakril" Rouault

hakril@lse.epita.fr
http://lse.epita.fr

July 19, 2012

Plan

Why this tutorial ?

- Initiation to binary exploitation
- Initiation to analysis tools
- Practice is everything

Why Starcraft ?

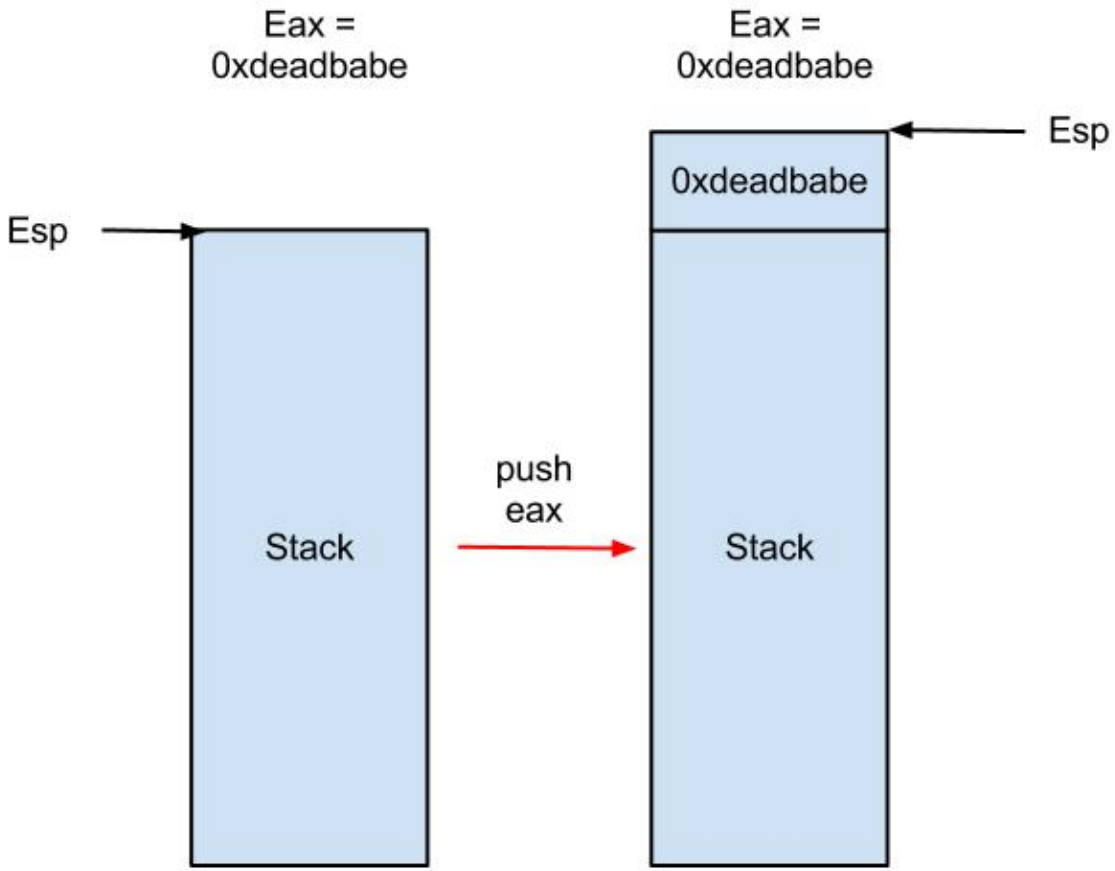
- Always fun to pown something people know
- The vulnerability is simple to trigger
- The vulnerability allow everything

Plan

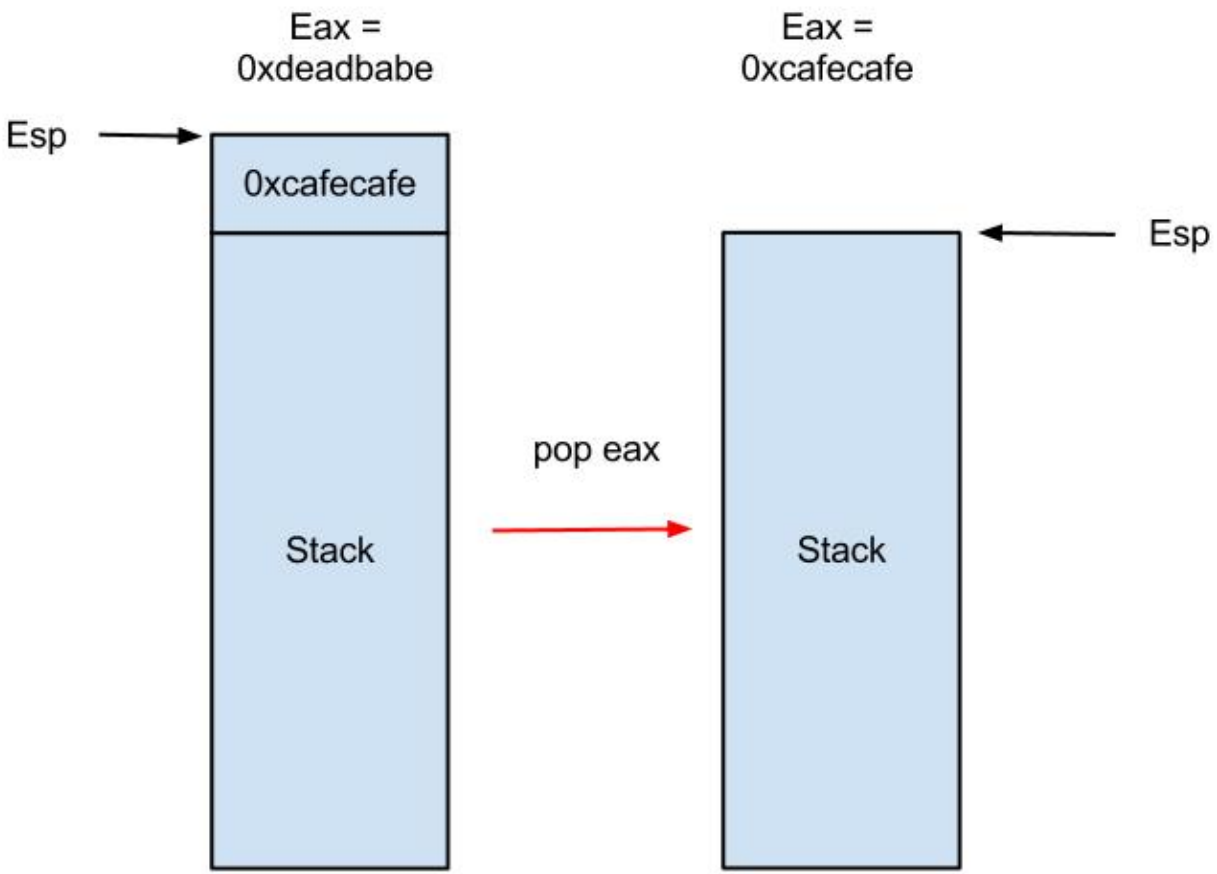
Important Registers

- EIP: Pointer to next instruction
- ESP: Stack Pointer

Push


Eax =
0xdeadbabe
Esp
Stack
push
eax
Eax =
0xdeadbabe
Esp
0xdeadbabe
Stack

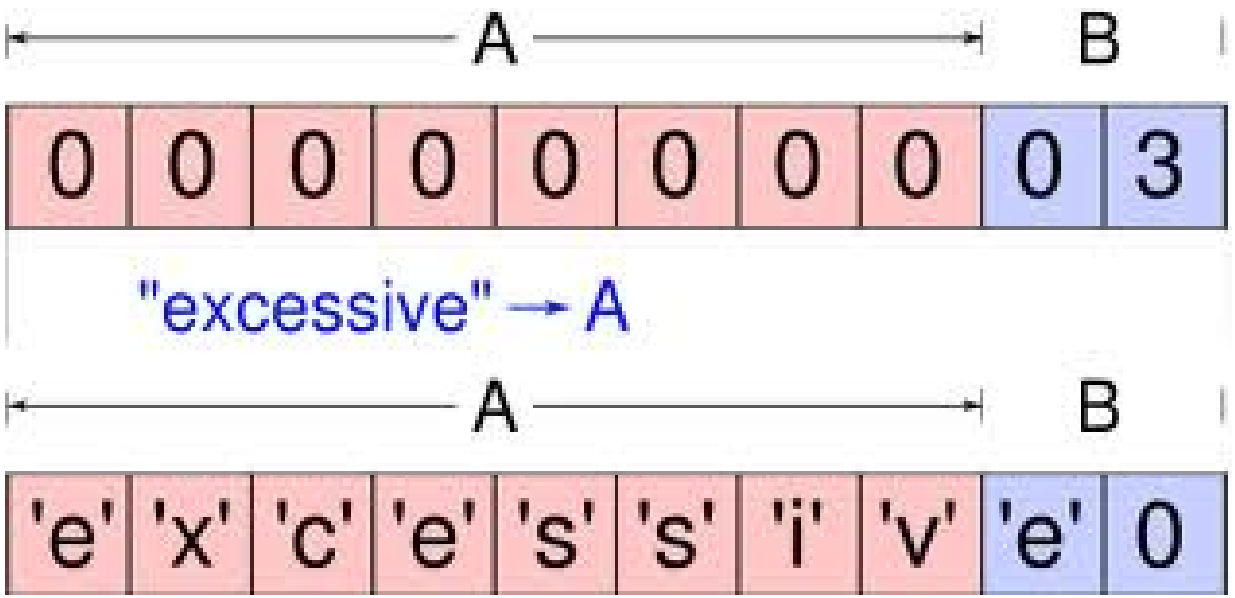
Pop


Eax =
0xdeadbabe
Esp
0xcafecafe
Stack
pop eax
Eax =
0xcafecafe
Esp
Stack

Call

- Call $addr:
  - Push ‘Next instr addr‘
  - jmp $addr
- Ret:
  - pop EIP

Plan



Starcraft case

- BOF when loading a save file

```
sub_44E540 proc near

DstBuf= byte ptr -34h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

xor     eax, eax
sub     esp, 34h
mov     dword_6241E0, eax
lea     edx, [esp+34h+DstBuf] ; DstBuf
mov     dword_6241E4, eax
mov     ecx, offset Str1 ; Str1
mov     dword_50A104, 0
mov     byte_633DBC, 0
mov     dword_6241E8, eax
call    Load_character
test    eax, eax
jnz     short loc_44E5AA
```

- Demo Time ! {1}

The basic way

- Put the shellcode on the save file
- Program load-it in the stack
- Find a way to jump on our code

The dummy way

- 'Hard-code' the stack return address
- Really bad idea !
- Solution : Gadgets !

The dummy way

- 'Hard-code' the stack return address
- Really bad idea !
- Solution : Gadgets !

The dummy way

- 'Hard-code' the stack return address
- Really bad idea !
- Solution : Gadgets !

What is a Gadget ?

- Reuse some part of the program's code.
- Some code-fragments can be useful out of context
- Some instructions can appear when splitting others

Splitting example

- "or ebp, 80h" => 81 CD 80 00 00 00
- CD 80 => "int 0x80"

Splitting example

- "or ebp, 80h" => 81 CD 80 00 00 00
- CD 80 => "int 0x80"

Gadgets in our case

- We want to jump on the stack
- jmp $esp / call $esp / push $esp ; ret
- Let's try the last one
- Demo Time ! {2}{3}

Gadgets in our case

- We want to jump on the stack
- jmp $esp / call $esp / push $esp ; ret
- Let's try the last one
- Demo Time ! {2}{3}

Gadgets in our case

- We want to jump on the stack
- jmp $esp / call $esp / push $esp ; ret
- Let's try the last one
- Demo Time ! {2}{3}

DEP

- Data Execution Prevention
- Can't execute writable page (W^X)
- Solution : More Gagdets !

DEP

- Data Execution Prevention
- Can't execute writable page (W^X)
- Solution : More Gagdets !

Plan

ROP ?

- Heavily use gadget of type '* ; ret'
- Chaining gadgets using 'ret'
- We are able to 'call' functions

Simple Rop on starcraft

- Finding Gadgets {4}
- Using 'write' gadget
- Call function

Simple Rop on starcraft

- Finding Gadgets {4}
- Using 'write' gadget
- Call function

Simple Rop on starcraft

- Finding Gadgets {4}
- Using 'write' gadget
- Call function

Explanation

- Write "calc.exe" in buffer at B_ADDR
  - {WinExec}
  - {Exit gagdet}
  - {B_ADDR}
  - {0}
  - {0xABCDEF00}
  - {1}
- Demo Time ! {5}

Explanation

- Write "calc.exe" in buffer at B_ADDR
  - {WinExec}
    {Exit gagdet}
    {B_ADDR}
    {0}
    {0xABCDEF00}
    {1}
- Demo Time ! {5}

Explanation

- Write "calc.exe" in buffer at B_ADDR
  - {WinExec}
    {Exit gagdet}
    {B_ADDR}
    {0}
    {0xABCDEF00}
    {1}
- Demo Time ! {5}

ASLR

- Randomize the address space.
- At each exec : system function addr. change
- can't use 'hard-coded' address

The Last ROP

- Using the IAT and VirtualAlloc
- Copy exploit in a W & R page
- Jump on it !
- Demo time ! {6}

The Last ROP

- Using the IAT and VirtualAlloc
- Copy exploit in a W & R page
- Jump on it !
- Demo time ! {6}

The Last ROP

- Using the IAT and VirtualAlloc
- Copy exploit in a W & R page
- Jump on it !
- Demo time ! {6}

Plan

What is SEH ?

- Structured Exception Handler
- Chain of Exception Handler
- Form : {Next Handler / SE Handle}
- First handler of the chain pointed by fs:[0]
- POC time !{7}

What is SEH ?

- Structured Exception Handler
- Chain of Exception Handler
- Form : {Next Handler / SE Handle}
- First handler of the chain pointed by fs:[0]
- POC time !{7}

What is SEH ?

- Structured Exception Handler
- Chain of Exception Handler
- Form : {Next Handler / SE Handle}
- First handler of the chain pointed by fs:[0]
- POC time !{7}

So what ?

- In a SEH Handler the address of pointer to next SEH is push at esp + 8
- As we overwrite the handler : we control the pointed data
- Need to jump at esp + 8
- pop pop ret ?

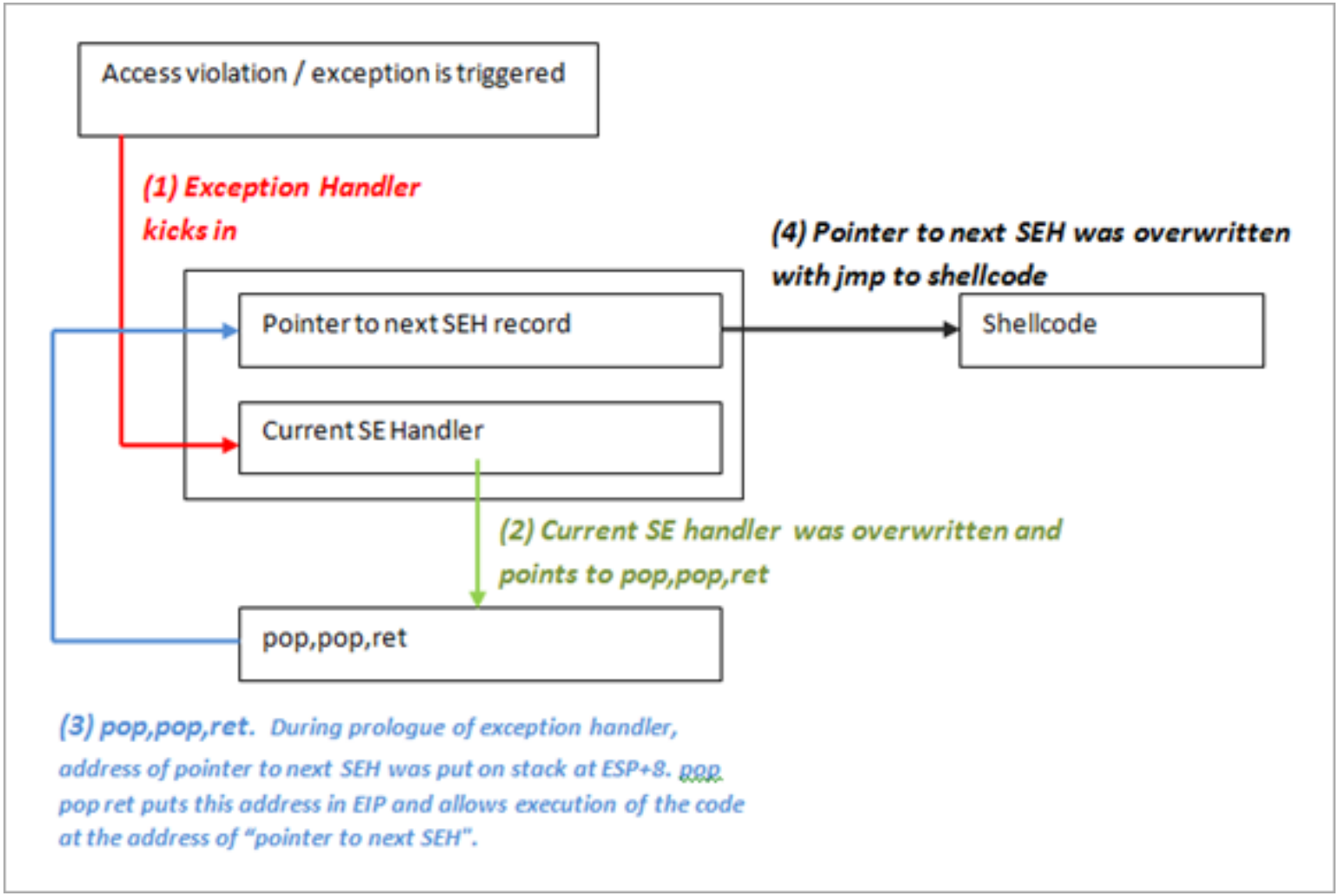
So what ?

- In a SEH Handler the address of pointer to next SEH is push at esp + 8
- As we overwrite the handler : we control the pointed data
- Need to jump at esp + 8
- pop pop ret ?

So what ?

- In a SEH Handler the address of pointer to next SEH is push at esp + 8
- As we overwrite the handler : we control the pointed data
- Need to jump at esp + 8
- pop pop ret ?

Draw me an exploit



Show me an exploit

- Demo time ! {8}

Plan

Contacts

- email: `hakril@lse.epita.fr`
- twitter: @hakril

Questions?