PRACTICAL REVERSING VI - EXPLOIT DEVELOPMENT [ADVANCED]

Amit Malik

www.SecurityXploded.com

Disclaimer

The Content, Demonstration, Source Code and Programs presented here is "AS IS" without any warranty or conditions of any kind. Also the views/ideas/knowledge expressed here are solely of the trainer's only and nothing to do with the company or the organization in which the trainer is currently working.

However in no circumstances neither the trainer nor SecurityXploded is responsible for any damage or loss caused due to use or misuse of the information presented here.

Acknowledgement

- Special thanks to **Null** & **Garage4Hackers** community for their extended support and cooperation.
- Thanks to all the **Trainers** who have devoted their precious time and countless hours to make it happen.
- Thanks to **ThoughtWorks** for the beautiful and bigger venue.

Reversing & Malware Analysis Training

This presentation is a part of our **Reverse Engineering & Malware Analysis** training program. Currently it is delivered only during our local meet for FREE of cost.

For complete details of this course, visit our Security Training page.

Who am I #1

Amit Malik (sometimes DouBle_Zer0,DZZ)

- Member SecurityXploded & Garage4Hackers
- Security Researcher @ McAfee Labs
- RE, Exploit Analysis/Development, Malware Analysis
- Email: m.amit30@gmail.com

Agenda

- The material in this presentation is a bit complicated so I will be using the zig-zag approach.
  - Recap
  - Protections (GS and SAFESEH)
  - Client side exploits and Heap Spray
  - Protections (DEP)
  - Protections (ASLR)
- If time permits then few words on the following:
  - Heap buffer overflows

Recap

- In previous session we covered:
  - Stack based buffer overflow
    - EIP overwrite (saved return address)
    - SEH Overwrite
- We also discussed “why we need pop pop ret or other similar instruction in SEH overflow”
- Now Question: Which one is more reliable or considered to be more reliable in terms of exploitation ?
  - Consider we have overwritten EIP and SEH successfully.

Protections Enforced by OS and Processor

ASLR

DEP

SEHOP

GS Cookies

SAFESEH

Forced ASLR

*Safe unlinking, Heap cookies etc. are also protection methods added into the OS.

Protections for stack based buffer overflow (Primary)

- Fortunately or Unfortunately both protection schemes are based on compiler/Linker options.

- * SEHOP is a different protection scheme based on run time SEH chain validation, It is not based on compiler options however can be enabled or disabled through registry.

GS Cookie (/GS)

- Put some random value (cookie – 32 bit) on stack before return address.
- While returning, compare the value of saved cookie, if not same then we have an overwrite.
- Generate “ Security Exception (if any)”, terminate the Application.

/GS Cookie Cont...

Function Start:

```
mov     edi, edi
push    ebp
mov     ebp, esp
sub     esp, 0E0h
mov     eax, ___security_cookie
xor     eax, ebp        ; XOR Cookie with EBP
mov     [ebp+var_4], eax ; put on stack [ebp - 4]
mov     eax, [ebp+arg_4]
push    esi
```

Cookie check function (see "function end" in below picture.)

```
; __fastcall __security_check_cookie(x)
@__security_check_cookie@4 proc near
cmp     ecx, ___security_cookie
jnz     ___report_gsfailure
```

Function end:

```
mov     ecx, [ebp+var_4]
xor     ecx, ebp
pop     esi
call    @__security_check_cookie@4 ; __security_check_cookie(x)
leave
retn    0Ch
```

/GS Cookie Bypass

- Generate exception before cookie check
  - Code dependent – if some overwritten variables are used before function return.
  - Overwrite stack up to the end, further overwrite will generate exception
- Back to the question which exploitation (EIP or SEH) is more reliable ?
  - SEH method is considered to be a bit more safe and reliable regardless of this bypassing technique.

/GS Cookie Bypass Cont..

Leverage the implementation. Did you see something ?

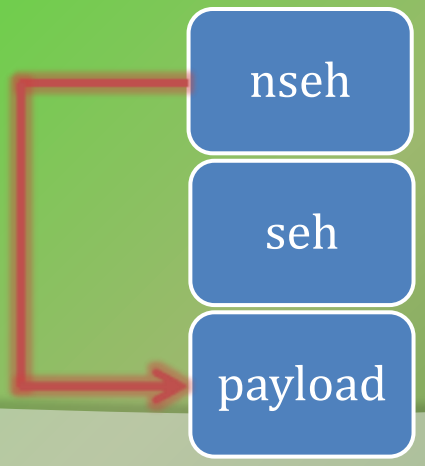
```
mov     dword_4AD24378, esi
mov     dword_4AD24374, edi
mov     word_4AD243A0, ss
mov     word_4AD24394, cs
mov     word_4AD24370, ds
mov     word_4AD2436C, es
mov     word_4AD24368, fs
mov     word_4AD24364, gs
pushf
pop     dword_4AD24398
mov     eax, [ebp+0]
mov     dword_4AD2438C, eax
mov     eax, [ebp+4]
mov     dword_4AD24390, eax
lea     eax, [ebp+arg_0]
mov     dword_4AD2439C, eax
mov     eax, [ebp+var_320]
mov     dword_4AD242D8, 10001h
mov     eax, dword_4AD24390
mov     dword_4AD24294, eax
mov     dword_4AD24288, 0C0000409h
mov     dword_4AD2428C, 1
mov     eax, ___security_cookie
mov     [ebp+var_328], eax
mov     eax, ___security_cookie_complement
mov     [ebp+var_324], eax
push    0                   ; lpTopLevelExceptionFilter
call    ds:__imp__SetUnhandledExceptionFilter@4 ; SetUnhandledExceptionFilter(x)
push    offset ExceptionInfo ; ExceptionInfo
call    ds:__imp__UnhandledExceptionFilter@4 ; UnhandledExceptionFilter(x)
push    0C0000409h          ; uExitCode
call    ds:__imp__GetCurrentProcess@0 ; GetCurrentProcess()
push    eax                 ; hProcess
call    ds:__imp__TerminateProcess@8 ; TerminateProcess(x,x)
leave
retn
```

SafeSEH

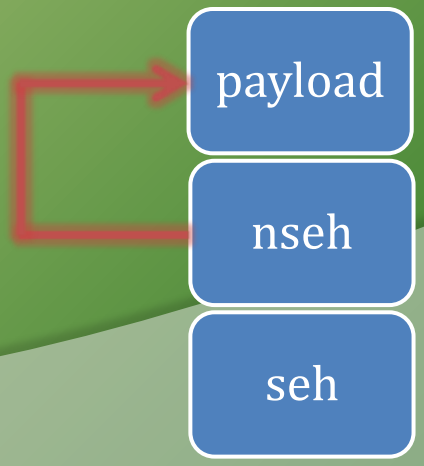
- Compiler [Linker] /SAFESEH option
- Static list of known good exception handlers for the binary.
- Checks every time when a handler is called against the static list, if not in the list then handler is invalid and takes preventive measures.
- Load configuration directory stores meta information about safe exception handlers.
- If any module is not compiled with /SAFESEH then no check is done to ensure the integrity of the handler for that module.

/SAFESEH Bypassing

- If any loaded module in the vulnerable binary is not /SAFESEH compiled then no check is done to ensure the integrity of the handler for that module, so we can use any p/p/r address from that module.
- Use the address that is outside the address range of loaded modules.
- Importance of forward and backward jump.



Forward jump



Backward jump

DEP (Data Execution Prevention)

- Two types:
  - Software DEP (forget it)
  - Hardware DEP (NX/XD enabled processors) – we will be talking about it in the rest of the session.
- We can't execute the code from non executable area anymore.
- We are directly dealing with processer in this case.

DEP (HW) Bypass

- ROP (Return Oriented Programming)
  - Use the system/existing code
  - How stack works ?
- Main theme
  - Either make non executable area executable
  - Or allocate new area with executable permissions
  - How ?
    - Well, use ROP ☺

Stack Heap Flipping (Stack Pivoting)

- I think this deserve a dedicated slide
- Depending on the conditions we may have large ROP payload while space on stack may be less or may be our entire payload is on heap.
- Flip the heap on to the stack so that we can get larger room.
- Instructions like XCHG ESP[REG], REG[ESP] can be used.
- We can also jump inside the valid instructions to change their meaning.
  - Example: jump one byte inside "**setz al**" instruction (From Adobe U3D exploit in wild)

DEP (HW) Bypass (DEMO)

- Methods
  - HeapCreate
  - VirtualAlloc
  - VirtualProtect
  - WriteProcessMemory (DEMO – simple, easy, demonstrate the entire concept – XpSp3)
- Often times the small code chunks in ROP are termed as “gadgets”

DEP (HW) Bypass (DEMO)

http://vimeo.com/49069964

ASLR

- Address Space Layout Randomization
- Involves randomly positioning the memory areas like base address of the binary, position of stack and heap.
- Compiler[linker] /DYNAMICBASE option

ASLR Bypass

- Search for Non-ASLR loaded libraries in the vulnerable application or if possible load one. ☺
  - JRE ?
- Memory leaks
- Brute force
- Heavily depends on vulnerability conditions

Client Side Exploits

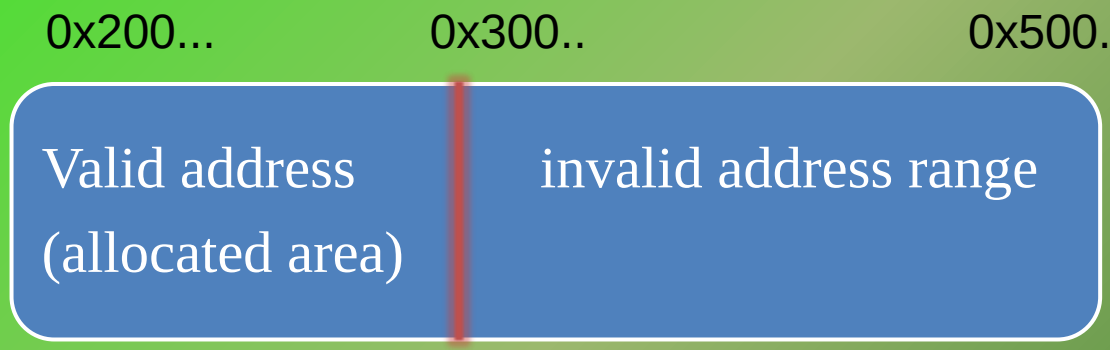
- Exploits that targets client applications like browsers, plugins, media players, readers etc.
- Much more dangerous then any other form of exploits
- Huge impact and landscape
- Provides solid infection vector
- Big malicious infrastructure.
  - Botnets, DDOS, Spam etc.

Heap Spray

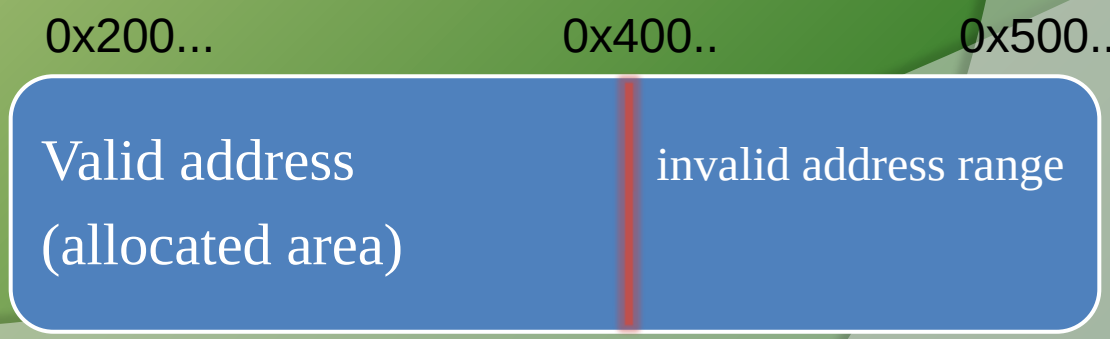
- A technique used in client side exploits
- IT'S NOT A VULNERABILITY or CLASS OF VUL.
- It's a technique used for code execution.
- Think about the followings again:
  - EIP overwrite
  - SEH overwrite
  - What we used in the above and why we used that ?
- Heap spray provides very simple method for code execution.

Heap Spray Cont...

- Fortunately or unfortunately client side scripting languages like javascript, vbscript etc. provides methods to allocate and deallocate memory on the client.
- Which means we can make invalid memory addresses valid.



Before Allocation



After Allocation (0x300.. To 0x400.. Is valid now)

Heap Spray Cont..

- Allocate memory and fill with nop + shellcode
- Overwrite the EIP or SEH with any address within the newly allocated area (the nop region).
- Here EIP overwrite or SEH overwrite can be by any means.
  - Stack buffer overflow, Heap buffer overflow, memory corruption, use after free etc..

Heap Spray (DEMO – IEPeers Vulnerability (IE6, IE7)

http://vimeo.com/49070337

Heap Spray (Stability Comments)

- Use intelligent guesses
- Stability depends on the exploitation conditions
- Fragmented heap, choose little higher addresses.
- Large number of allocations, choose little lower addresses ☺

Reference

- Complete Reference Guide for Reversing & Malware Analysis Training

Thank You !

www.SecurityXploded.com