ARSENAL

UCSB

ShellNoob

Because writing shellcode is fun,
~~but sometimes painful~~

Black Hat USA
Arsenal 2013

Yanick Fratantonio
UC Santa Barbara

Who am I?

- PhD Student at UC Santa Barbara
- I play with the ShellPhish team
- What I do
  - I like low-level stuff
  - I worked on shellcode analysis
  - Now I'm on Android security
    - static / dynamic analysis

- Links
  - Website: http://cs.ucsb.edu/~yanick
  - Email: yanick [at] cs.ucsb.edu
  - Twitter: @reyammer

Writing shellcode - why?

- Sometimes, something ad-hoc is required
- Even when you need something simple, there are problems with already written ones
- How about Shellcode generators?
  - Even the most advanced ones sometimes fail
    - And if they fail, you are fucked (good luck debugging them!)
      - (any reference to Metasploit's shellcode generator is purely coincidental)
        - But please don't get me wrong, Metasploit is awesome :-)

What's the issue?

- We have incredibly awesome tools that try to do incredibly difficult tasks
  - Shellcode generators are just one example

- This might be too complicated to be infallible
  - Metasploit is written by uber smart guys
  - Why are shellcode generators still not bullet-proof?
    - Extremely difficult stuff!
    - We need a plan B

So what?

- It's good to have something crazy difficult that *sometimes* works

- But it's also good to have something simple that makes simple tasks even simpler

Shellcode writing facts

- We need to be prepared to write shellcode
- Writing shellcode is fun, but some steps are boring, error-prone, and hence painful
- Most of such steps can be automated once for all!

Examples of boring steps

- Shellcode on the web, that is almost exactly what you want, but still needs some tweaks
    - example: samples from shell-storm shellcode DB

- Sometimes they are not in the wanted format, and you need to "convert" them
    - assembly to hex
    - ELF to assembly
    - C to raw binary
        - I've seen VIM macros that you people wouldn't believe...
    - ...and all the other combinations

Examples of boring steps (2)

- Syscall numbers
  - *Which number was the "read" again?*
  - *3 you say? Is that on Linux or FreeBSD? duuude!*

- Resolving constants
  - *O_CREAT was 0, right? oh, on FreeBSD you say?*
  - *Aaah, that was O_RDWR. Or maybe O_RDONLY?*

**Sentences in italic indicate real questions asked by myself or my fellow colleagues*

Examples of boring steps (3)

- Alright, I have the shellcode: now let's compile and test it
    - *mmm, how can I do that?*
- Let's run it in gdb
    - *Hey it crashed, WTF?*
    - *oh, self modifying shellcode in the non-writable code segment?*
        - no good :/
- Now let's run it against the target
    - *FUUUCK, if it contains byte "0x42" it gets corrupted.*
    - *Do you think that "inc %edx" will be a problem?*

ShellNoob to the rescue!

Disclaimer -- What ShellNoob is NOT

- It's NOT a replacement for Metasploit's shellcode generator
- It will NOT try to generate shellcode for you
- It will NOT be bug-free
    - But the goal is simple enough that coders more skilled than me will fix them soon!
        - Go and start now: https://github.com/reyammer/shellnoob :-)

What the hell is it then?

- A toolkit to help you write shellcode

- Design principles & goals
  - Extremely easy to deploy and use
  - Automate and make as easy as possible whatever it supposed to be easy
  - Trial & error should be cheap process
  - Portable & Flexible -- easy to extend
  - Easy to understand "what's going on"
    - To debug the tool
    - As a way to learn how to do it manually!

Easy to deploy & use

- ShellNoob is a single self-contained python script (~1K LOC)

- Deployment? Just `scp` it on the target device

- If you want, you can "install" it
  - ./shellnoob.py --install
    - `"cp shellnoob.py /usr/local/bin/snoob"`

- You are now ready to hack!

Conversion mode

- Usual task: convert the shellcode from one "format" to another one

- Input formats
  - --from-asm
  - --from-bin
  - --from-hex
  - --from-obj (an ELF)
  - --from-c
  - --from-shellstorm

Conversion mode

- Usual task: convert the shellcode from one "format" to another one

- Input formats
  - **--from-asm**
  - --from-bin
  - --from-hex
  - --from-obj (an ELF)
  - --from-c
  - --from-shellstorm

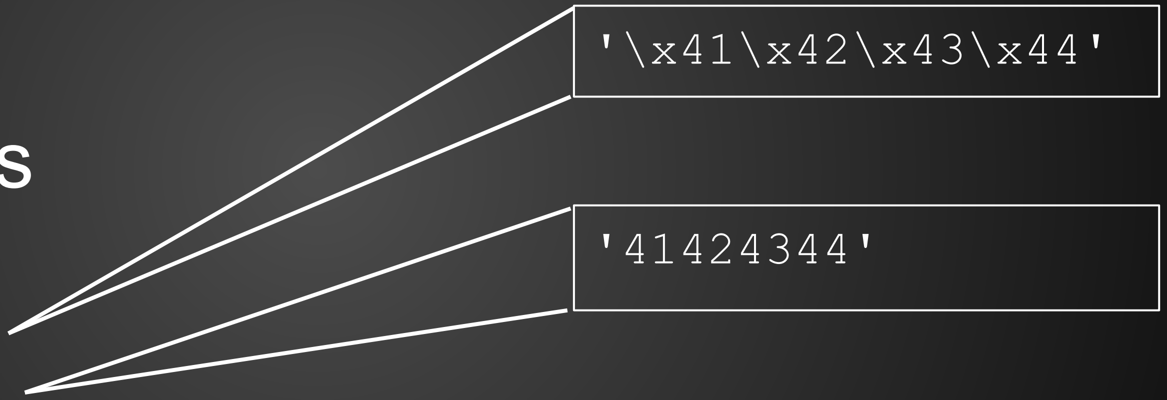
```
.section .text
    mov %ebx, %ecx
    mov %eax, %ebx
    xor %edx, %edx
    addb $0xff, %dl
    xor %eax, %eax
    movb $0x3, %al
    int $0x80
```

Support for both ATT & Intel syntax!

Conversion mode

- Usual task: convert the shellcode from one "format" to another one

- Input formats
  - --from-asm
  - **--from-bin**
  - **--from-hex**
  - --from-obj (an ELF)
  - --from-c
  - --from-shellstorm



```
'\x41\x42\x43\x44'
```

```
'41424344'
```

Conversion mode

- Usual task: convert the shellcode from one "format" to another one

- Input formats
  - --from-asm
  - --from-bin
  - --from-hex
  - --from-obj (an ELF)
  - **--from-c**
  - --from-shellstorm

```
char shellcode[] =
  "\x6a\x0b\x58\x99"
  "\x52\x66\x68\x2d"
  "\x70\x89\xe1\x52"
  "\x6a\x68\x68\x2f";
```

But be careful, it's just doing its best in guessing what's the shellcode

Conversion mode

- Usual task: convert the shellcode from one "format" to another one

- Input formats
  - --from-asm
  - --from-bin
  - --from-hex
  - --from-obj (an ELF)
  - --from-c
  - **--from-shellstorm <shellcode_id>**

That's it! ShellNoob will download and convert the shellcode from the DB

Conversion mode

- Usual task: convert the shellcode from one "format" to another one
- Output formats
  - --to-asm
  - --to-safeasm
  - --to-bin
  - --to-hex
  - --to-obj
  - --to-exe
  - --to-c, --to-completec
  - --to-python, --to-bash, --to-ruby

Conversion mode

- Usual task: convert the shellcode from one "format" to another one

- Output formats
  - **--to-asm**
  - --to-safeasm
  - --to-bin
  - --to-hex
  - --to-obj
  - --to-exe
  - --to-c, --to-completec
  - --to-python, --to-bash, --to-ruby

```
.section .text
  jmp 0x37          # .byte 0xeb,0x35
  pop %ebx          # .byte 0x5b
  mov %ebx,%eax     # .byte 0x89,0xd8
  add $0xb,%eax     # .byte 0x83,0xc0,0x0b
```

Conversion mode

- Usual task: convert the shellcode from one "format" to another one

- Output formats
  - **--to-asm**
  - --to-safeasm
  - --to-bin
  - --to-hex
  - --to-obj
  - --to-exe
  - --to-c, --to-completec
  - --to-python, --to-bash, --to-ruby

```
.section .text
  ...
  das                      # .ascii "/"
  je 0xac                  # .ascii "tm"
  jo 0x70                  # .ascii "p/"
  jae 0xa8                 # .ascii "se"
  arpl %si,0x65(%edx)      # .ascii "cre"
  je 0xa0                  # .ascii "tX"
```

Conversion mode

- Usual task: convert the shellcode from one "format" to another one

- Output formats
  - --to-asm
  - **--to-safeasm**
  - --to-bin
  - --to-hex
  - --to-obj
  - --to-exe
  - --to-c, --to-completec
  - --to-python, --to-bash, --to-ruby

```
.section .text
  .byte 0xeb,0x35
  .byte 0x5b
  .byte 0x89,0xd8
  .byte 0x83,0xc0,0x0b
```

"safe mode" -- 100% assemblable

Uber flexible CLI

- Some examples (all equivalent)

```
$ snoob --from-asm shell.asm --to-bin shell.bin
$ snoob shell.asm --to-bin shell.bin
$ snoob shell.asm --to-bin
$ snoob shell.asm --to-bin - > shell.bin
$ cat shell.asm | snoob --from-asm - --to-bin shell.bin
```

- Several switches

```
$ snoob -c shell.asm --to-exe  # prepend a breakpoint
$ snoob --intel shell.asm --to-exe  # Intel vs ATT syntax
$ snoob --64 shell.asm --to-exe  # 64bits vs 32bits mode
```

Syscalls and constants

- When writing shellcode, you need to directly call syscalls: you need to know the numbers!
  - `$ snoob --get-sysnum read`
  - `x86 ~> 3`
  - `x86_64 ~> 0`

- Similarly, you need to resolve the constants!
  - `$ snoob --get-const O_RDWR`
  - `O_RDWR ~> 2`
  - It can also be used to resolve the error numbers!
    - Example: EACCES ~> 13

Interactive mode

- Quick ways to check which bytes a specific instruction is assembled to (and viceversa)

- Assembly ~> opcode

  ```
  $ snoob -i --to-opcode
  >> mov %eax, %ebx
  mov %eax, %ebx ~> 89c3
  ```

- Opcode ~> assembly

  ```
  $ snoob -i --to-asm
  >> 89c3
  89c3 ~> mov %eax, %ebx
  ```

Trial & error should be "cheap"

- You are convinced your shellcode is right, but there is a bug. Debug it!

- "Special" output modes
  - **--to-strace**

```
$ snoob open-read-write-shell.asm --to-strace
[ Process PID=15085 runs in 32 bit mode. ]
open("/tmp/secret", O_RDONLY) = 3
read(3, "ThisIsMySecret", 255) = 14
write(1, "ThisIsMySecret", 14ThisIsMySecret) = 14
_exit(0) = ?
```

Trial & error should be "cheap"

- You are convinced your shellcode is right, but there is a bug. Debug it!

- "Special" output modes
  - --to-strace
  - **--to-gdb**

```
$ snoob open-read-write.asm --to-gdb
Reading symbols from /tmp/tmpzTg_T0...(no debugging
symbols found)...done.
(gdb) Breakpoint 1 at 0x8048054
(gdb)
```

A breakpoint is automatically set on the first instruction!

Easily portable & extendable

- The only dependencies
  - as, objdump, ld, objcopy, python, [strace, gdb]
- Built-in support for
  - Linux / i386 - x86_64 - ARM
  - FreeBSD / i386 - x86_64
- Possible extensions
  - Adding a new conversion mode is really easy
    - You just need to define a *_to_hex and/or hex_to_* functions!
    - All the plumbing is done automatically!
  - Adding support for a new OS/arch is simple as well
    - check the `{as,objdump,ld}_options_map` fields!

ShellNoob as a library

- ShellNoob is a huge Python object
    - all the settings go in the constructor
    - all the features are exported as methods
        - conversion functions (`asm_to_hex(asm)`, ...)
        - `do_resolve_syscall(syscall)`
        - ...

```
$ python
Python 2.7.4 (default, Apr 19 2013, 18:28:01)
[GCC 4.7.3] on linux2
>>> from shellnoob import ShellNoob
>>> sn = ShellNoob()
>>> sn.asm_to_hex('nop')
'90'
>>>
```

Additional plugins

- Following the usual mantra
    - *All the simple tasks should be automated and made as simple as possible*

- Few additional "plugins"

```
$ snoob --file-patch <exe_fp> <file_offset> <data>
$ snoob --vm-patch <exe_fp> <vm_address> <data>
$ snoob --fork-nopper <exe_fp>  # this nops out the fork()s
```

That's all folks!

Thanks!

- Links
  - Website: http://cs.ucsb.edu/~yanick
  - Email: yanick [at] cs.ucsb.edu
  - Twitter: @reyammer
  - ShellNoob: https://github.com/reyammer/shellnoob