Adventures in Binary Golf

Who Am I?

yuu

@netspooky on twitter/github

n0.lol

thugcrowd

Talk Outline

File Format Hacking

Various Case Studies

(ab)Use Cases

Approaches to ELF and PE files

Further Reading

A Brief Overview of File Format Hacking

Extensive research has been done into File Format Hacking

Smallest Possible Files

- mems/small: https://github.com/mems/small

Polyglots, Chimeras et al.

- PoC||GTFO 7.6 - Abusing File Formats
- https://www.alchemistowl.org/pocorgtfo/pocorgtfo07.pdf
- https://github.com/ViGrey/gb-nes-pdf-html-zip

ThugCrowd: badge.gif (2018)

https://thugcrowd.com/chal/badge.gif

Part of a challenge to win a custom Defcon badge. See “spiderman frozen elsa” (2019) for another example of a polyglot used in a ThugCrowd challenge [vtt/jpg].

Created with a hex editor.

Triple Polyglot (with other fun stuff thrown in)

- Relevant files: GIF, Gameboy ROM, Zip Archive

Binwalk Output:

```
$ binwalk badge.gif

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Gameboy ROM, name: "[^0^][~_^]!!", [ROM ONLY], ROM: 256Kbit
21520         0x5410          ELF, 64-bit LSB executable, AMD x86-64, version 1 (SYSV)
31804         0x7C3C          Zip archive data, encrypted at least v2.0 to extract, compressed size: 806, uncompressed size: 8142, name: nice
32730         0x7FDA          End of Zip archive
```

Badge.gif Internals

```
00000000: 4749 4638 3961 0100 0100 00ff 002c 0000  GIF89a.......,..
00000010: 0000 0100 0100 0002 003b ffff ffff ffff  .........;......
00000020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000030: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000040: e521 0cc5 c367 00ff e521 1cc5 c367 00ff  .!...g...!...g..
00000050: e521 2cc5 c367 00ff e521 3cc5 c367 00ff  .!,..g...!<..g..
00000060: e521 4cc5 c367 00f5 c5d5 2ab6 280b e53a  .!L..g....*.(..:
00000070: 6e67 cd7e 00e1 2318 f1d1 c1f1 e1d9 e9ff  ng.~..#.........
00000080: 2020 2323 2323 2020 2020 2323 2323 2020      ####    ####
00000090: 2023 2020 2020 2320 2023 2020 2020 2320   #    #  #    #
000000a0: 2320 2023 2023 2023 2320 2320 2320 2023  #  # # ## # #  #
000000b0: 2320 2023 2023 2023 2320 2320 2320 2023  #  # # ## # #  #
000000c0: 2320 2320 2020 2023 2320 2020 2023 2023  # #    ##    # #
000000d0: 2320 2023 2323 2023 2320 2323 2320 2023  #  ### ## ###  #
000000e0: 2023 2020 2020 2320 2023 2020 2020 2320   #    #  #    #
000000f0: 2020 2323 2323 2020 2020 2323 2323 2020     ####    ####
00000100: 00c3 5001 ceed 6666 cc0d 000b 0373 0083  ..P...ff.....s..
00000110: 000c 000d 0008 111f 8889 000e dccc 6ee6  ..............n.
00000120: dddd d999 bbbb 6763 6e0e eccc dddc 999f  ......gcn.......
00000130: bbb9 333e 5b5e 305e 5d5b 7e5f 5e5d 2121  ..3>[^0^][~_^]!!
00000140: 0000 0000 0000 0000 0000 0000 014d 4433  .............MD3
00000150: f357 af31 00e0 21ff df0e 2006 0032 0520  .W.1..!... ..2.
00000160: fc0d 20f9 21ff fe06 0032 0520 fc21 ffff  .. .!....2. .!..
00000170: 0680 3205 20fc 7aea 03c5 cde1 12af e042  ..2. .z........B
00000180: e043 e041 e04a 3e07 e04b 0180 ff21 f812  .C.A.J>..K...!..
00000190: 060a 2ae2 0c05 20fa 01b9 12cd 7012 0102  ..*... .....p...
000001a0: 13cd 8212 3ee4 e047 e048 3e1b e049 3ec0  ....>..G.H>..I>.
000001b0: e040 afe0 0f3e 09e0 ffaf e026 e002 3e66  .@...>.....&..>f
000001c0: e001 3e80 e002 afcd f929 fbcd cc13 d306  ..>......).....
000001d0: 0100 7618 fdff ffff ffff ffff ffff ffff  ..v.............
000001e0: c9ff ffff c34e 23ff c374 19ff ffff ffff  .....N#..t......
000001f0: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000200: 2103 1ee5 cd57 29e8 0221 fec4 7ec6 f84f  !....W)..!..~..O
00000210: 21ff c47e f533 79f5 333e 03f5 33cd bc19  !..~.3y.3>..3...
00000220: e803 2104 20e5 cd57 29e8 0221 ffc4 7ef5  ..!. ..W)..!..~.
00000230: 3321 fec4 7ef5 333e 04f5 33cd bc19 e803  3!..~.3>..3.....
00000240: c921 043c e5cd 5729 e802 21fe c47e c6f8  .!.<..W)..!..~..
```

GIF 26 bytes

Random Ascii Art

Start of Gameboy Header

```
00007f70: f73f b514 b60f 4f82 0d3e 8e92 cbe3 ff5f  .?....O..>....._
00007f80: 438a e669 504b 0102 3f00 1400 0100 0800  C..iPK..?.......
00007f90: 1796 ff4c b899 c2aa 2603 0000 ce1f 0000  ...L....&.......
00007fa0: 0400 2400 0000 0000 0000 2000 0000 0000  ..$....... .....
00007fb0: 0000 6e69 6365 0a00 2000 0000 0000 0100  ..nice.. .......
00007fc0: 1800 befe 12a3 2029 d401 114b d082 1e29  ...... )...K...)
00007fd0: d401 f554 c982 1e29 d401 504b 0506 0000  ...T...)..PK....
00007fe0: 0000 0100 0100 5600 0000 4803 0000 0000  ......V...H.....
```

Zip File - 108 Bytes

What is Binary Golf?

Binary Golf is the practice of crafting the smallest possible binary that still performs a given function.

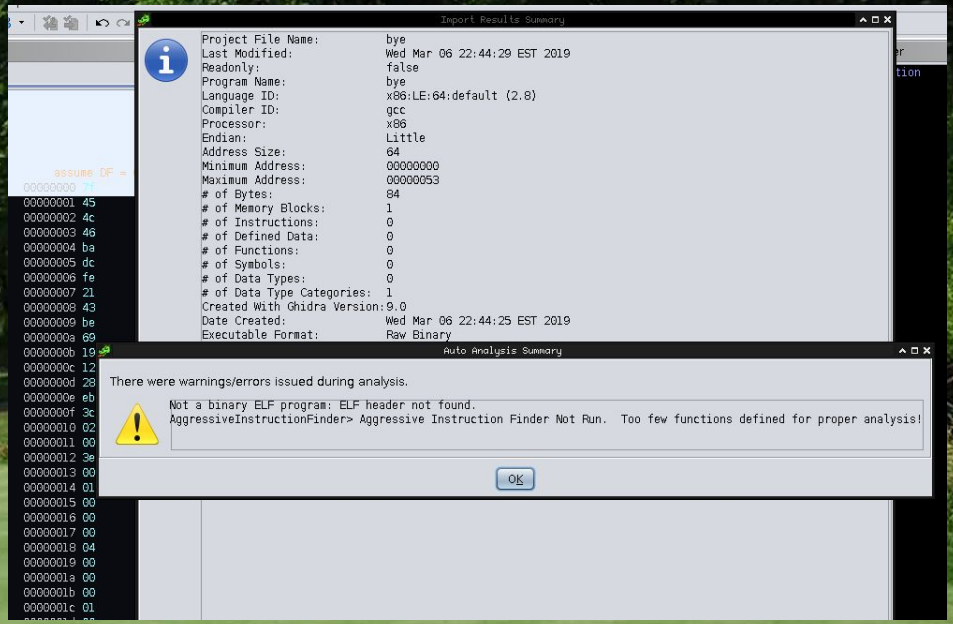
Can be created with or without a compiler, generally created without one.

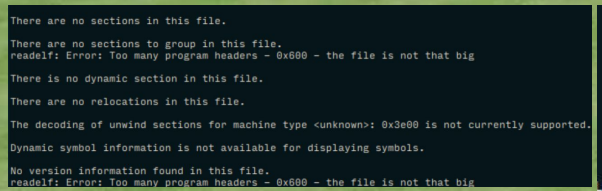
Tools include:

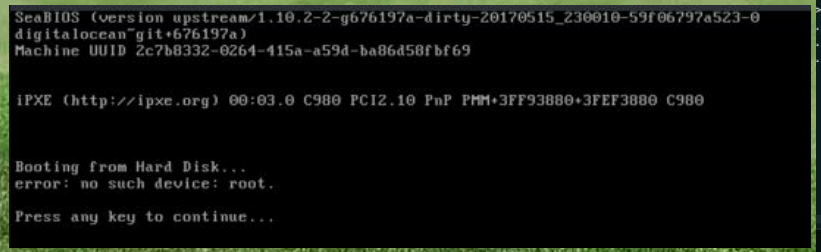
- nasm (solid), or any other assembler you like
- hex editor (for fixing mistakes)
- gcc (good luck, it's possible tho, shoutout Anonymous_)

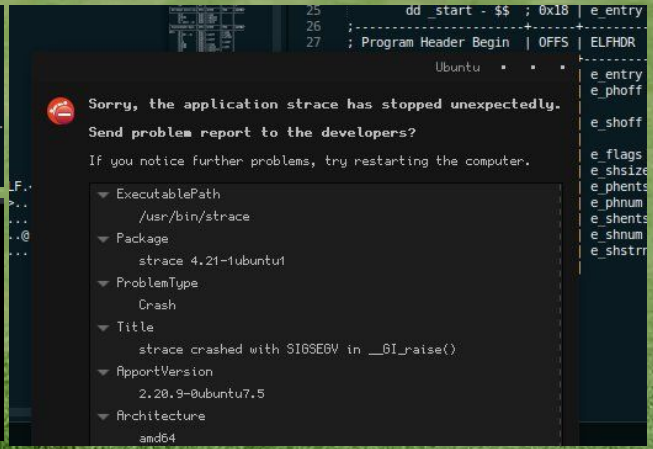
(ab)Use Cases

- Anti-Debug/Anti-Forensics
- Exploit Prototyping
- AV / Detection bypass
- File upload filter bypass
- Fuzzing
- Malware loaders
- Fun!









Approaches to Binary Golf

- Examining binary file structure
- Analyzing the specification/RFC/dev notes
- Analyzing open source parsers
- (Slowly?) removing things you don't need
- Fuzzing suspected areas of interest
- Binary Diffing different files to better understand the format

ELF64 (Linux)

ELF files have a lot of extra stuff in them that aren't needed. These include:

- Debug Symbols
- Unnecessary Sections and Headers
- Padding
- Other info needed by parsing tools

All you really need (for a standard ELF binary) is:

- ELF Header
- Program Header
- Code to execute

Note: Shared Objects and Kernel Modules require some additional parts!

Tiny ELF Files

Prior art (32 bit): https://www.muppetlabs.com/~breadbox/software/tiny/teensy.html

- Doesn't run on x86_64 Linux
- The syscall interface is different now

64 bit Tiny ELF examples: https://github.com/netspooky/golfclub

Updated for modern Linux systems

See ELF Binary Mangling Parts 1-3

https://medium.com/@dmxinajeansuit/elf-binary-mangling-part-1-concepts-e00cb1352301

https://medium.com/@dmxinajeansuit/elf-binary-mangling-pt-2-golfin-7e5c82bb482c

https://medium.com/@dmxinajeansuit/elf-binary-mangling-part-3-weaponization-6e11971108b3

Golfing with ELF

Strip all unnecessary sections and headers.

Overlay program headers with ELF header.

Store code and other data in unused sections of the headers.

Reuse values from the headers for other things.

Short jumps to locations within the header.

Load binary at 0x100000000 so that e_entry and p_type match.

```
; 84 byte LINUX_REBOOT_CMD_POWER_OFF Binary Golf
BITS 64
  org 0x100000000
;---------------------+------+------------+-------------------------------------+----------------------------+-----------+
; CODE LISTING        | OFFS | ASSEMBLY   | CODE COMMENT                        | ELF HEADER STRUCT          | PHDR      |
;---------------------+------+------------+-------------------------------------+----------------------------+-----------+
  db 0x7F, "ELF"      ; 0x0  |   7f454c46 | PROTIP: Can use magic as a constant ;) | ELF Magic               |           |
_start:               ;------|------------|-------------------------------------|----------------------------|-----------|
  mov edx, 0x4321fedc ; 0x04 | badcfe2143 | Moving magic values...              | ei_class,ei_data,ei_version |          |
  mov esi, 0x28121969 ; 0x09 | be69191228 | into their respective places        | unused                     |           |
  jmp short reeb      ; 0x0E |       eb3c | Short jump down to @x4c             | unused                     |           |
  dw 2                ; 0x10 |       0200 |                                     | e_type                     |           |
  dw 0x3e             ; 0x12 |       3e00 |                                     | e_machine                  |           |
  dd 1                ; 0x14 |   01000000 |                                     | e_version                  |           |
  dd _start - $$      ; 0x18 |   04000000 |                                     | e_entry                    |           |
phdr:                 ;------|------------|-------------------------------------|----------------------------|-----------|
  dd 1                ; 0x1C |   01000000 |                                     | e_entry                    | p_type    |
  dd phdr - $$        ; 0x20 |   1c000000 |                                     | e_phoff                    | p_flags   |
  dd 0                ; 0x24 |   00000000 |                                     | e_phoff                    | p_offset  |
  dd 0                ; 0x28 |   00000000 |                                     | e_shoff                    | p_offset  |
  dq $$               ; 0x2C |   00000000 |                                     | e_shoff                    | p_vaddr   |
                      ; 0x30 |   01000000 |                                     | e_flags                    | p_vaddr   |
  dw 0x40             ; 0x34 |       4000 |                                     | e_shsize                   | p_addr    |
  dw 0x38             ; 0x36 |       3800 |                                     | e_phentsize                | p_addr    |
  dw 1                ; 0x38 |       0100 |                                     | e_phnum                    | p_addr    |
  dw 2                ; 0x3A |       0200 |                                     | e_shentsize                | p_addr    |
cya:                  ;------|------------|-------------------------------------|----------------------------|-----------|
  mov al, 0xa9        ; 0x3C |       b0a9 | Load syscall                        | e_shnum                    | p_filesz  |
  syscall             ; 0x3E |       0f05 | Execute syscall                     | e_shstrndx                 | p_filesz  |
  dd 0                ; 0x40 |   00000000 | Filler, should try to keep as all 0's |                          | p_filesz  |
  mov al, 0xa9        ; 0x44 |       b0a9 | Load syscall                        |                            | p_memsz   |
  syscall             ; 0x46 |       0f05 | Execute syscall                     |                            | p_memsz   |
  dd  0               ; 0x48 |   00000000 | Filler, should try to keep as all 0's |                          | p_memsz   |
reeb:                 ;------|------------|-------------------------------------|----------------------------|-----------|
  mov edi, 0xfee1dead ; 0x4C | bfaddee1fe | Load magic "LINUX_REBOOT_CMD_POWER_OFF" |                        | p_align   |
  jmp short cya       ; 0x51 |       ebe9 | Short jmp back to e_shnum/p_filesz @0x3C |                       | p_align   |
  nop                 ; 0x53 |         90 | Filler, could use this byte for code. |                          | p_align   |
;---------------------+------+------------+-------------------------------------+----------------------------+-----------+
```

https://github.com/netspooky/golfclub/blob/master/linux/bye.asm

Case Study: golf.so

A recent challenge in a CTF to create a shared object under 1024 bytes that pops a shell:

https://teamrocketist.github.io/2020/04/20/Misc-PCTF2020-golf-so/

Shared Objects have much more strict checking. Techniques of putting code at 0x04:0x0F and other locations do not work here.

```
ERROR: ld.so: object './golf.2.so' from LD_PRELOAD cannot be preloaded (wrong ELF class: ELFCLASS32): ignored.
ERROR: ld.so: object './golf.2.so' from LD_PRELOAD cannot be preloaded (ELF file data encoding not little-endian): ignored.
ERROR: ld.so: object './golf.2.so' from LD_PRELOAD cannot be preloaded (ELF file OS ABI invalid): ignored.
ERROR: ld.so: object './golf.2.so' from LD_PRELOAD cannot be preloaded (ELF file ABI version invalid): ignored.
ERROR: ld.so: object './golf.2.so' from LD_PRELOAD cannot be preloaded (nonzero padding in e_ident): ignored.
```

Invoke using LD_PRELOAD

golf.so.2

Determined the needed sections:

- DT_STRTAB
- DT_INIT
- DT_SYMTAB

Overlayed with program headers

Used nasm to create the final binary

Resulting file size: 185 bytes

```
BITS 64
org     0
ehdr:
  db    0x7f, "ELF", 2, 1, 1, 0    ; e_ident
  db    0, 0, 0, 0,  0, 0, 0, 0
  dw    3                          ; e_type    = ET_DYN
  dw    62                         ; e_machine = EM_X86_64
  dd    1                          ; e_version = EV_CURRENT
  dq    _start                     ; e_entry   = _start
  dq    phdr - $$                  ; e_phoff
  dd    phdr - $$                  ; e_shoff (chaged to phdr instead of shdr)
  dq    0                          ; e_flags
  dw    ehdrsize                   ; e_ehsize
  dw    phdrsize                   ; e_phentsize
  dw    2                          ; e_phnum
ehdrsize equ  $ - ehdr

phdr:
  dd    1                          ; p_type   = PT_LOAD
  dd    7                          ; p_flags  = rwx
  dq    0                          ; p_offset
  dq    $$                         ; p_vaddr
  dq    $$                         ; p_paddr
  dq    0x68732f6e69622f                ; p_filesz
  dq    0xDEADBEEF                 ; p_memsz
  dq    0x1000                     ; p_align
phdrsize equ  $ - phdr
  dd    2                          ; p_type  = PT_DYNAMIC
  dd    7                          ; p_flags = rwx
dynsection:
; DT_STRTAB
  dq    0x5                        ; p_offset (OVERLAPPED)
  dq    dynsection                 ; p_vaddr
; DT_INIT
  dq    0x0c
  dq    _start
; DT_SYMTAB
  dq    0x06
  dq    _start
global _start
_start:
lea rdi,[rax-0x50]
push 59
pop rax
push 0
push rdi
mov rsi,rsp
;cdq ; this may be needed locally but in the website accepts anyway without this (1 byte save)
syscall
```

Case Study: PE

Quirks with PEs differ drastically between Windows versions

TinyPE is already a well known thing

PE files must be >=268 bytes on Windows 7/10

Overlay technique for headers is used

REFS:

- https://github.com/rcx/tinyPE
- https://github.com/corkami/pics/tree/master/binary/pe101
- http://www.phreedom.org/research/tinype/

Exploring Code Caves

Unused header sections means that values can be placed in these locations and *hopefully* not interfere with binary loading/execution.

Short jumps allow us to easily hop around the header in these small sections.

This process can be manual or fuzzer guided.

Identified Caves:

Range	Len
0x0C:0x18	12
0x1E:0x2C	14
0x44:0x4C	8
0x4E:0x54	6
0x5C:0x60	4
0x70:0x78	8

DOS Header

#	Sz	Desc
MA	2	e_magic
MB	2	e_cblp **
MC	2	e_cp **
MD	2	e_crlc **
ME	2	e_cparhdr **
MF	2	e_minalloc **
MG	2	e_maxalloc **
MH	2	e_ss **
MI	2	e_sp **
MJ	2	e_csum **
MK	2	e_ip **
ML	2	e_cs **
MM	2	e_lsarlc **
MN	2	e_ovno **
MO	8	e_res **
MP	2	e_oemid **
MQ	2	e_oeminfo **
MR	20	e_res2 **
MS	4	e_lfanew PE Sig Addr

Anything marked with a * means that it is unused. Some of these might have some expected value ranges to respect, so keep that in mind when playing with them!

PE Header

#	Sz	Desc
PA	4	PE Signature
PB	2	Machine (Intel 386)
PC	2	NumberOfSections
PD	4	TimeDateStamp **
PE	4	PointerToSymbolTable **
PF	4	NumberOfSymbols **
PG	2	SizeOfOptionalHeader
PH	2	Characteristics (no relocs, executable, 32 bit)

Optional Header

#	Sz	Desc
OA	2	Magic (PE32)
OB	1	MajorLinkerVersion **
OC	1	MinorLinkerVersion **
OD	4	SizeOfCode **
OE	4	SizeOfInitializedData **
OF	4	SizeOfUninitializedData **
OG	4	AddressOfEntryPoint
OH	4	BaseOfCode **
OI	4	BaseOfData **
OJ	4	ImageBase
OK	4	SectionAlignment
OL	4	FileAlignment
OM	2	MajorOperatingSystemVersion **
ON	2	MinorOperatingSystemVersion **
OO	2	MajorImageVersion **
OP	2	MinorImageVersion **
OQ	2	MajorSubsystemVersion
OR	2	MinorSubsystemVersion **
OS	4	Win32VersionValue **
OT	4	SizeOfImage
OU	4	SizeOfHeaders
OV	4	CheckSum ** *
OW	2	Subsystem (Win32 GUI)
OX	2	DllCharacteristics **
OY	4	SizeOfStackReserve **
OZ	4	SizeOfStackCommit
O1	4	SizeOfHeapReserve
O2	4	SizeOfHeapCommit **
O3	4	LoaderFlags **
O4	4	NumberOfRvaAndSizes **

Launching calc.exe

Used PEB -> WinExec technique

Shellcode was a bit longer than could fit.

Determined places in the header that code could go.

Performed short jumps around the header before landing in the code section: jump0-jump6

https://n0.lol/a/pemangle.html

```
$ xxd tiny268.exe
00000000: 4d5a 0001 5045 0000 4c01 0000 31f6 83ec  MZ..PE..L...1...
00000010: 1856 6a63 9090 eb06 6000 0301 0b01 6668  .Vjc....`.....fh
00000020: 7865 6857 696e 4589 65fc eb22 7c00 0000  xehWinE.e.."|...
00000030: 0000 0000 0000 0000 0000 4000 0400 0000  ..........@.....
00000040: 0400 0000 8b5b 0c8b 5b14 eb10 0500 648b  .....[..[.....d.
00000050: 5e30 ebf0 8000 0000 7c00 0000 8b1b eb10  ^0......|.......
00000060: 0200 0004 0000 1000 0010 0000 0000 1000  ................
00000070: 8b1b 8b5b 10eb 07c3 0000 0000 eb8e 895d  ...[...........]
00000080: f88b 433c 01d8 8b40 7801 d88b 4824 01d9  ..C<...@x...H$..
00000090: 894d f48b 7820 01df 897d f08b 501c 01da  .M..x ...}..P...
000000a0: 8955 ec8b 5814 31c0 8b55 f88b 7df0 8b75  .U..X.1..U..}..u
000000b0: fc31 c9fc 8b3c 8701 d766 83c1 08f3 a674  .1...<...f.....t
000000c0: 0a40 39d8 72e5 83c4 26eb ac8b 4df4 89d3  .@9.r...&...M...
000000d0: 8b55 ec66 8b04 418b 0482 01d8 31d2 5268  .U.f..A.....1.Rh
000000e0: 2e65 7865 6863 616c 6368 6d33 325c 6879  .exehcalchm32\hy
000000f0: 7374 6568 7773 5c53 6869 6e64 6f68 433a  stehws\ShindohC:
00000100: 5c57 89e6 6a0a 56ff d083 c446            \W..j.V....F
```

```
;--- Start of MZ Header ---------------------------------------------------------
mzhdr:
  dw "MZ"       ; 0x00 ; [MA] e_magic
  dw 0x100      ; 0x02 ; [MB] e_cblp This value will bypass TinyPE detections!
;--- Start of PE Header ---------------------------------------------------------
pesig:
  dd "PE"       ; 0x04 ; [MC] e_cp [MD] e_crlc [PA] PE Signature
pehdr:
  dw 0x014C     ; 0x08 ; [ME] e_cparhdr [PB] Machine (Intel 386)
  dw 0          ; 0x0A ; [MF] e_minalloc [PC] NumberOfSections (0 haha)
jump0: ; WinExec Setup Part 1 ----------------------------------------------------
  xor  esi,esi  ; 0x0C ; 31f6   ; Clear ESI [MG] e_maxalloc [PD] TimeDateStamp
  sub  esp,0x18 ; 0x0E ; 83ec18 ; Make room for our bullshit [MH] e_ss [MI] e_sp
  push esi      ; 0x11 ; 56     ; Null   [PE] PointerToSymbolTable
  push 0x63     ; 0x12 ; 6a63   ; "c"    [MJ] e_csum
  nop           ; 0x14 ; 90     ; spacer [MK] e_ip [PF] NumberOfSymbols
  nop           ; 0x15 ; 90     ; spacer
  jmp jump1     ; 0x16 ; eb06   ; [ML] e_cs
  dw 0x60       ; 0x18 ; [MM] e_lsarlc [PG] SizeOfOptionalHeader
  dw 0x103      ; 0x1A ; [MN] e_ovno [PH] Characteristics
;--- Start of Optional Header ---------------------------------------------------
  dw 0x10B      ; 0x1C ; [MO] e_res [OA] Magic (PE32)
jump1: ; WinExec Setup Part 2 ----------------------------------------------------
  push word 0x6578 ;0x1E; 66687865   ; "ex" [OB] MajorLinkerVersion
                                     ; [OC] MinorLinkerVersion [OD] SizeOfCode
  push 0x456e6957  ;0x22; 6857696e45 ; "EniW"
                                     ; [MP] e_oemid [MQ] e_oeminfo [OE] SizeOfInitializedData
  mov dword [ebp-4], esp ;0x27; 8965fc; Save our stack pointer addr for later
                                     ; [MR] e_res2 [OF] SizeOfUninitializedData
  jmp jump2     ; 0x2A ; eb22
  dd 0x7C       ; 0x2C ; [OG] AddressOfEntryPoint (Could make a label pointer)
  dd 0          ; 0x30 ; [OH] BaseOfCode
  dd 0          ; 0x34 ; [OI] BaseOfData
  dd 0x400000   ; 0x38 ; [OJ] ImageBase
  dd 4          ; 0x3C ; [MS] e_ifanew [OK] SectionAlignment
  dd 4          ; 0x40 ; [OL] FileAlignment
jump3: ; PEB Parse Part 2 --------------------------------------------------------
  mov  ebx, [ebx+0xc]  ; 0x44 ; 8b5b0c ; Get addr of PEB_LDR_DATA
                                       ; [OM] MajorOperatingSystemVersion
                                       ; [ON] MinorOperatingSystemVersion
  mov  ebx, [ebx+0x14] ; 0x47 ; 8b5b14 ; InMemoryOrderModuleList first entry
                                       ; [OO] MajorImageVersion
  jmp  jump4           ; 0x4A ; eb10
                                       ; [OP] MinorImageVersion
  dw 5          ; 0x4C ; [OQ] MajorSubsystemVersion
jump2: ; PEB Parser Part 1 -------------------------------------------------------
  mov  ebx, [fs:0x30+esi] ; 0x4E ; 648b5e30 ; Get PEB addr, FS holds TEB address
                                       ; [OR] MinorSubsystemVersion
                                       ; [OS] Win32VersionValue
  jmp jump3     ; 0x52 ; ebf0
  dd 0x80       ; 0x54 ; [OT] SizeOfImage
  dd 0x7C       ; 0x58 ; [OU] SizeOfHeaders
jump4: ; PEB Parser Part 3 -------------------------------------------------------
  mov  ebx, [ebx] ; 0x5C ; 8b1b ; Get address of ntdll.dll entry [OV] CheckSum
  jmp jump5     ; 0x5E ; eb10
  dw 2          ; 0x60 ; [OW] Subsystem (Win32 GUI)
  dw 0x400      ; 0x62 ; [OX] DllCharacteristics
  dd 0x100000   ; 0x64 ; [OY] SizeOfStackReserve
  dd 0x1000     ; 0x68 ; [OZ] SizeOfStackCommit
  dd 0x100000   ; 0x6C ; [O1] SizeOfHeapReserve
jump5: ; PEB Parser Part 4 -------------------------------------------------------
  mov  ebx, [ebx]      ; 0x70 ; 8b1b   ; Get address of kernel32.dll list entry
                                       ; [O2] SizeOfHeapCommit
  mov  ebx, [ebx+0x10] ; 0x72 ; 8b5b10 ; Get kernel32.dll base address
                                       ; [O3] LoaderFlags
  jmp jump6            ; 0x75 ; eb07
endy:
  ret                  ; 0x77 ; c3  ; Used to end the program
  dd 0          ; 0x78 ; [O4] NumberOfRvaAndSizes  ; Note - this is touchy
codesec:        ; 0x7C - Start of code -----------------------------------------
                ; MachineCode ; Description
  jmp jump0            ; eb8e       ; Jump back to header to begin execution
jump6:
;--- Grab kernel32.dll base addr ------------------------------------------------
; This piece of code grabs the Thread Environment Block structure's address from
```

Defeating Detection

Since TinyPE is already a known technique for obfuscation, there are detections for it on Virus Total, and other scanners, as well as Yara Rules.

Detections for TinyPE were defeated by changing one bit in e_cblp.

```
$ xxd tiny268.2.exe
00000000: 4d5a 0001 5045 0000 4c01 0000 31f6 83ec  MZ..PE..L...1...
00000010: 1856 6a63 9090 eb06 6000 0301 0b01 6668  .Vjc....`.....fh
00000020: 7865 6857 696e 4589 65fc eb22 7c00 0000  xehWinE.e.."|...
00000030: 0000 0000 0000 0000 0000 4000 0400 0000  ..........@.....
00000040: 0400 0000 8b5b 0c8b 5b14 eb10 0500 648b  .....[..[.....d.
00000050: 5e30 ebf0 8000 0000 7c00 0000 8b1b eb10  ^0......|.......
00000060: 0200 0004 0000 1000 0010 0000 0000 1000  ................
00000070: 8b1b 8b5b 10eb 07c3 0000 0000 eb8e 895d  ...[...........]
00000080: f88b 433c 01d8 8b40 7801 d88b 4824 01d9  ..C<...@x...H$..
00000090: 894d f48b 7820 01df 897d f08b 501c 01da  .M..x ...}..P...
000000a0: 8955 ec8b 5814 31c0 8b55 f88b 7df0 8b75  .U..X.1..U..}..u
000000b0: fc31 c9fc 8b3c 8701 d766 83c1 08f3 a674  .1...<...f.....t
000000c0: 0a40 39d8 72e5 83c4 26eb ac8b 4df4 89d3  .@9.r...&...M...
000000d0: 8b55 ec66 8b04 418b 0482 01d8 31d2 5268  .U.f..A.....1.Rh
000000e0: 2e65 7865 6863 616c 6368 6d33 325c 6879  .exehcalchm32\hy
000000f0: 7374 6568 7773 5c53 6869 6e64 6f68 433a  stehws\ShindohC:
00000100: 5c57 89e6 6a0a 56ff d083 c446            \W..j.V....F
```

Defeating Detection

Yara detection is bypassed.

New yara detection does catch this PE. (For now...)

See sshell's writeup on fuzzing VT detection engines.


```
$ clear
$ yara tinype.yar tiny268.exe
SUSP_TINY_PE tiny268.exe
$ yara tinype.yar tiny268.2.exe
$ yara tinype.2.yar tiny268.exe
netspooky_SUSP_TINY_PE tiny268.exe
$ yara tinype.2.yar tiny268.2.exe
netspooky_SUSP_TINY_PE tiny268.2.exe
$ xxd tiny268.exe
00000000: 4d5a 0000 5045 0000 4c01 0000 31f6 83ec  MZ..PE..L...1...
00000010: 1856 6a63 9090 eb06 6000 0301 0b01 6668  .Vjc....`.....fh
00000020: 7865 6857 696e 4589 65fc eb22 7c00 0000  xehWinE.e.."|...
00000030: 0000 0000 0000 0000 4000 0400 0000 0000  ........@.......
00000040: 0400 0000 8b5b 0c8b 5b14 eb10 0500 648b  .....[..[.....d.
00000050: 5e30 ebf0 8000 0000 7c00 0000 8b1b eb10  ^0......|.......
00000060: 0200 0004 0000 1000 0010 0000 0000 1000  ................
00000070: 8b1b 8b5b 10eb 07c3 0000 0000 eb8e 895d  ...[...........]
00000080: f88b 433c 01d8 8b40 7801 d88b 4824 01d9  ..C<...@x...H$..
00000090: 894d f48b 7820 01df 897d f08b 501c 01da  .M..x ...}..P...
000000a0: 8955 ec8b 5814 31c0 8b55 f88b 7df0 8b75  .U..X.1..U..}..u
000000b0: fc31 c9fc 8b3c 8701 d766 83c1 08f3 a674  .1...<...f.....t
000000c0: 0a40 39d8 72e5 83c4 26eb ac8b 4df4 89d3  .@9.r...&...M...
000000d0: 8b55 ec66 8b04 418b 0482 01d8 31d2 5268  .U.f..A.....1.Rh
000000e0: 2e65 7865 6863 616c 6368 6d33 325c 6879  .exehcalchm32\hy
000000f0: 7374 6568 7773 5c53 6869 6e64 6f68 433a  stehws\ShindohC:
00000100: 5c57 89e6 6a0a 56ff d083 c446            \W..j.V....F
$ yara tinype.2.yar tiny268.2.exe
netspooky_SUSP_TINY_PE tiny268.2.exe
$ cat tinype.yar
rule SUSP_TINY_PE {
   meta:
      description = "Detects Tiny PE file"
      author = "Florian Roth"
      reference = "https://webserver2.tecgraf.puc-rio.br/~ismael/Cursos/YC++/apostilas/win32_xcoff_pe/tyne-example/Tiny%20PE.htm"
      date = "2019-10-23"
      score = 80
   strings:
      $header = { 4D 5A 00 00 50 45 00 00 }
   condition:
      uint16(0) == 0x5a4d and uint16(4) == 0x4550 and filesize <= 20KB and $header at 0
}$ cat tinype.2.yar
rule netspooky_SUSP_TINY_PE {
   meta:
      description = "Detects Tiny PE file"
      author = "Florian Roth"
      reference = "https://webserver2.tecgraf.puc-rio.br/~ismael/Cursos/YC++/apostilas/win32_xcoff_pe/tyne-example/Tiny%20PE.htm"
      date = "2019-10-23"
      score = 80
   strings:
      $header = { 50 45 00 00 }
   condition:
      uint16(0) == 0x5a4d and uint16(4) == 0x4550 and filesize <= 20KB and $header at 4
}$
```

Lessons Learned

You don't need huge bloated software to run binary programs in 2020.

File parsers generally lazy and need only a few things to consider a file valid.

Most debuggers and binary parsers kinda suck at understanding minified binaries still.

Even with a tiny binary, you can still bypass detections.

Binary Golfing gives you complete control over every single byte in your file.

Other Resources

- Radare2 - Great for debugging weird stuff
- https://github.com/corkami
- http://fileformats.archiveteam.org/wiki/Main_Page
- curl -sL https://n0.lol/i2ao/intro