ELF Crafting

Advance Anti-analysis techniques for the Linux Platform

INTEZER

CON

Whoami

- Security Researcher at Intezer Labs
- Malware Reverse Engineer and Threat Hunter
- CTF player with amn3s1a
- Radare2 Contributor
- Libelfmaster Contributor

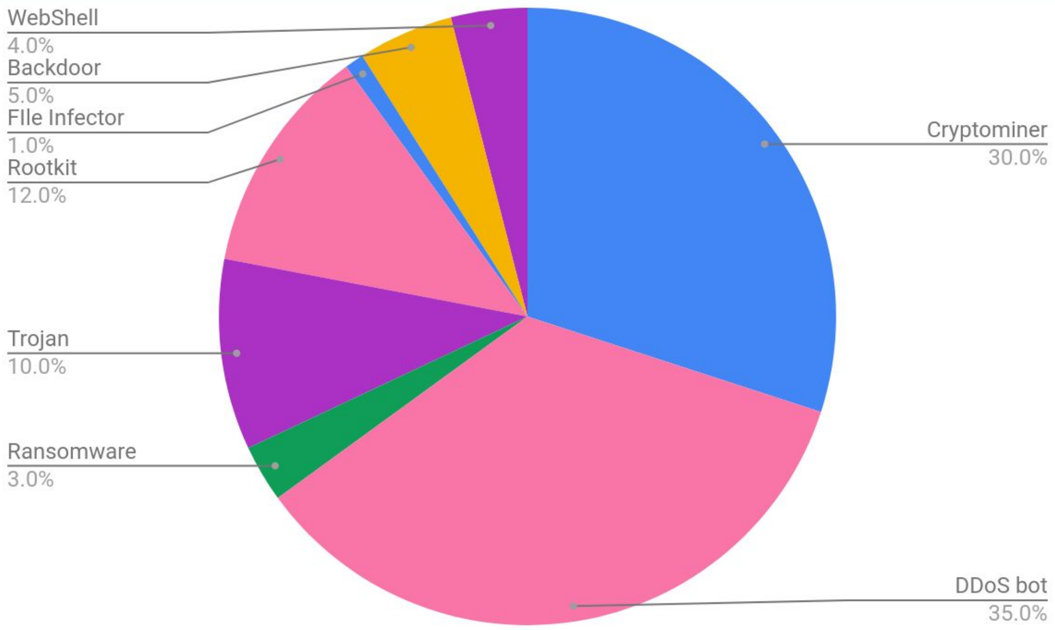
@ulexec

Outline

- Overview to the Linux Threat Landscape
- Advanced ELF crafting techniques
- Q&A

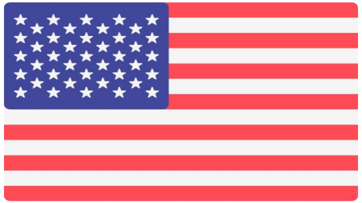
Overview of the Linux Threat Landscape

Linux Malware Distribution (2019)



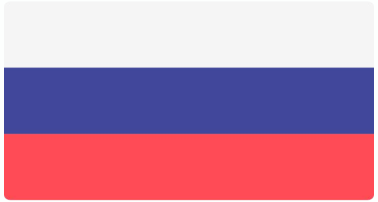
Intezer - 2019

APTs deploying Linux based Malware



Shadow Brokers
- jackpop
- NOPEN
- SUCTIONCHAR
- SECONDDATE

Vault7
- OutlawCountry
- BothanSpy
- Gyrfalcon



APT28
- Fysbis
- Zebrozy

APT29
- SeaDuke

Turla
- Penquin

Gamaredon
- EvilGnome

Winnti Umbrella
- HiddenWasp
- Azazel Based forks

Challenges on Linux Threat Detection

- Low Visibility
- Low Detection rate

Rise of awareness

- At Intezer we uncovered the following threats on 2019. Most of them were completely undetected before we reported them.

  - **Pacha Group** (Crypto-mining group)
  - **HiddenWasp** (Trojan linked to Winnti umbrella)
  - **QnapCrypt** (Ransomware targeting NAS servers)
  - **EvilGnome** (Trojan linked to Gamaredon Group)
  - **WatchBog Cython** (Crypto-mining botnet)

Rise of awareness

- In recent years we have seen a rise of researchers focused on Linux Malware Hunting promoting awareness on Linux based threats.

    - Cisco Talos
    - Netlab360
        - @liuya0904
        - @_rngm_
        - @zom3y3
        - @JiaYu_521
        - @huiwangeth
    - MalwareMustDie
        - @benkow_
        - @unixfreaxjp
    - Intezer
        - @polarply
        - @ulexec

What’s happening next

- Linux malware is doomed to increase in visibility.

- Linux threat detection performance will improve.

- Threat actors will invest more resources to make their implants more evasive and complex

This talk will present some techniques that may start appearing on the wild for defenders awareness as Linux malware evolves in complexity.

Advanced ELF Crafting Techniques

Advanced ELF Crafting

- ELF parsing struggles
  - A word about sections
  - Breaking naive parsers with 1 byte
  - Hiding dynamic entries

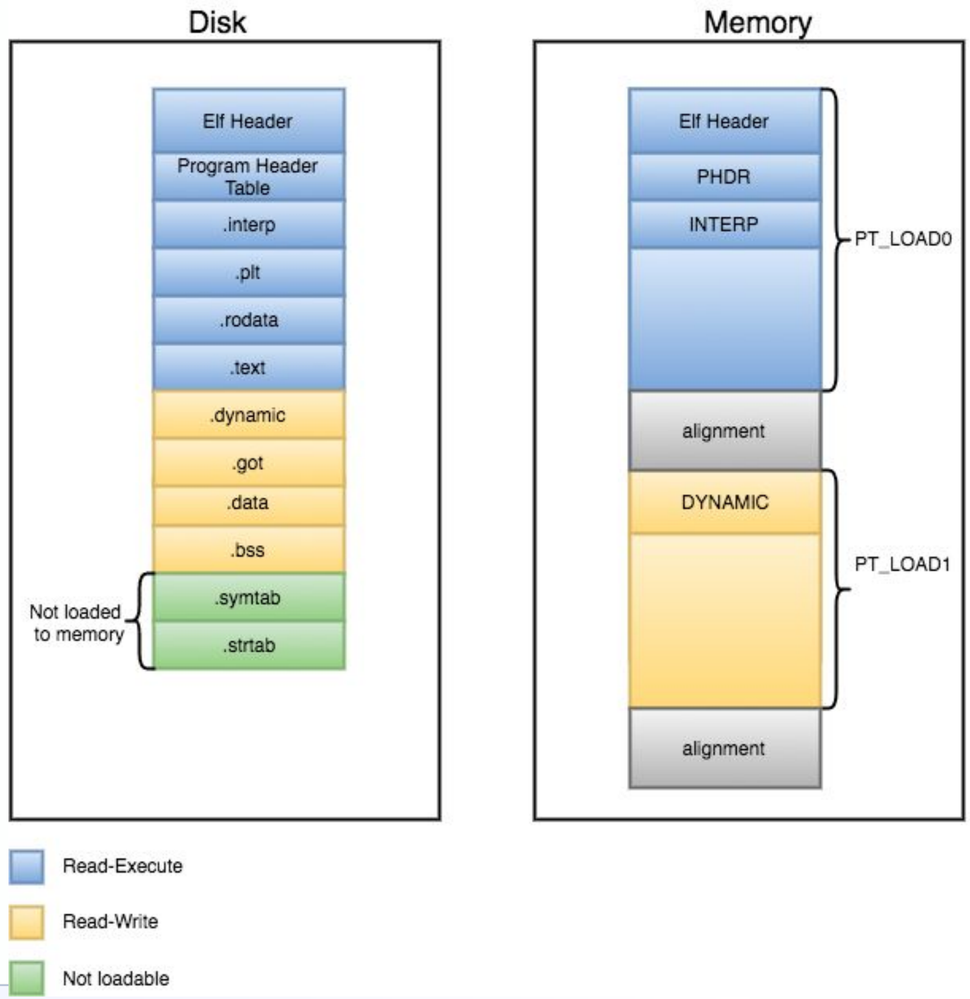
- Relocation Hijacking
  - EPO on PIE binaries
  - Hiding constructors in dynamic and static binaries

- Experimental Practices
  - Removing import strings from .dynstr string table

ELF Parsing Struggles

A word about sections

- Sections comprise all information needed for linking a target object file to build a runnable ELF executable.
- Sections are needed on link-time but they are not needed on run-time.
- Compiled ELF binaries will have a Section Header table (an array of *Elf*_Shdr* entries).
- Sections can hold different content ranging from code, strings, relocations or symbols.


Disk
Elf Header
Program Header Table
.interp
.plt
.rodata
.text
.dynamic
.got
.data
.bss
.symtab
.strtab
Not loaded to memory
Memory
Elf Header
PHDR
INTERP
PT_LOAD0
alignment
DYNAMIC
PT_LOAD1
alignment
Read-Execute
Read-Write
Not loadable

INTEZER

A word about sections

- Section information can be completely removed from the binary or modified.
- This can be abused by attackers to misguide analyst or parsers that heavily rely on section information
- Example use-cases:
  - Symbol scrambling
  - Section scrambling

Example outcome

```
ulexec  intezer  ~  R2CON2019  section_scramling  $  readelf -d ./ls
readelf: Error: Section 0 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Error: Section 3 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 16 for the rest of this dump)
readelf: Error: Section 4 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Error: Section 11 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 16 for the rest of this dump)
readelf: Error: Section 14 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Error: File contains multiple dynamic symbol tables
readelf: Error: File contains multiple dynamic symbol tables
readelf: Error: File contains multiple dynamic symbol tables
readelf: Error: Section 23 has invalid sh_entsize of 0000000000000008
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Error: Section 24 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 16 for the rest of this dump)
readelf: Error: File contains multiple dynamic symbol tables
readelf: Error: Section 26 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Warning: the .dynamic section is not contained within the dynamic segment

Dynamic section at offset 0x3e70 contains 1 entry:
  Tag        Type                         Name/Value
 0x90660021c15225ff (<unknown>: 90660021c15225ff)       0x90660021c16a25ff
ulexec  intezer  ~  R2CON2019  section_scramling  $
```

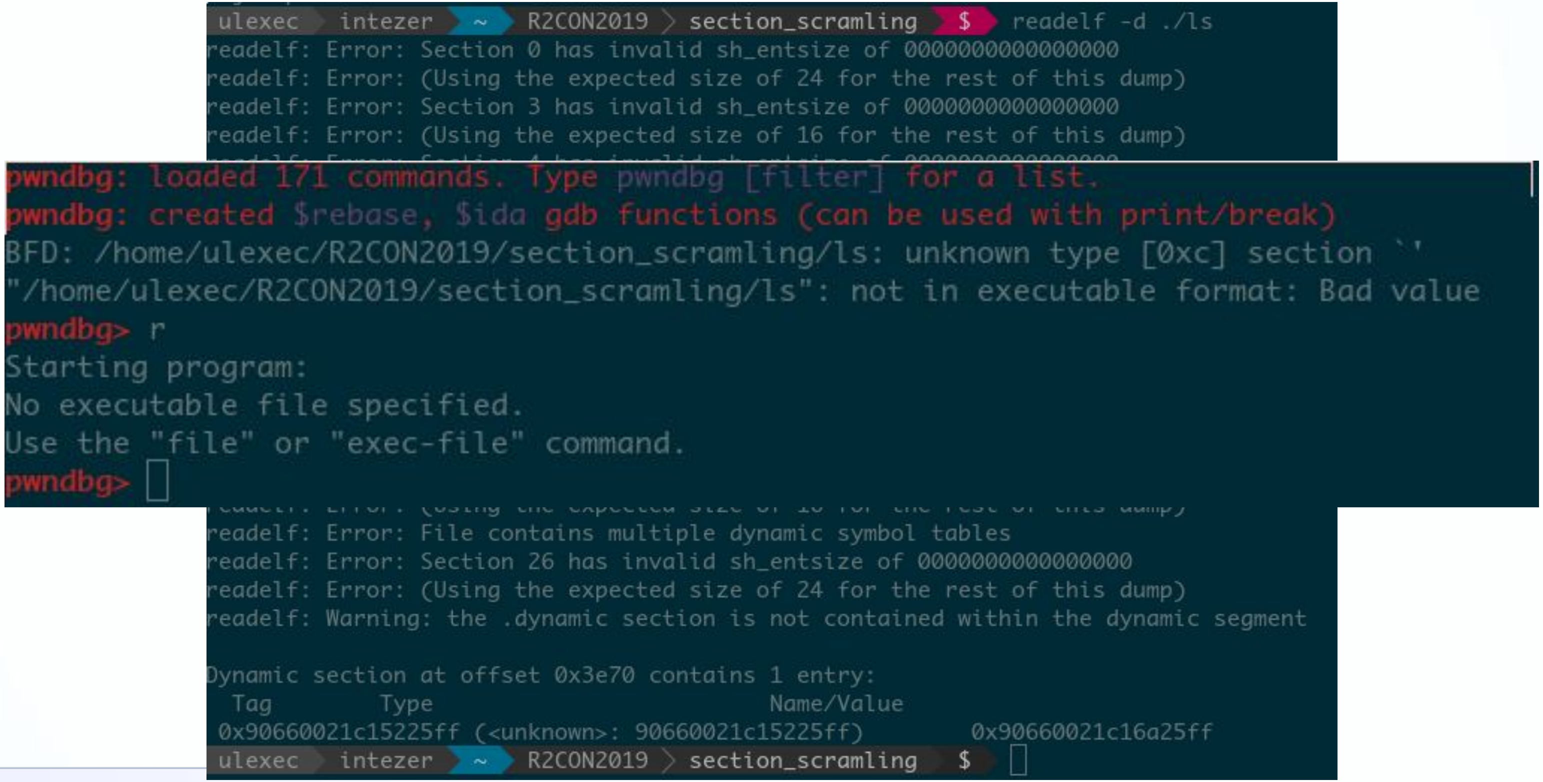
Example outcome

```
ulexec  intezer  ~  R2CON2019  section_scramling  $  readelf -d ./ls
readelf: Error: Section 0 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Error: Section 3 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 16 for the rest of this dump)
readelf: Error: Section 4 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Error: Section 11 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 16 for the rest of this dump)
readelf: Error: Section 23 has invalid sh_entsize of 0000000000000008
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Error: Section 24 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 16 for the rest of this dump)
readelf: Error: File contains multiple dynamic symbol tables
readelf: Error: Section 26 has invalid sh_entsize of 0000000000000000
readelf: Error: (Using the expected size of 24 for the rest of this dump)
readelf: Warning: the .dynamic section is not contained within the dynamic segment

Dynamic section at offset 0x3e70 contains 1 entry:
  Tag        Type                         Name/Value
 0x90660021c15225ff (<unknown>: 90660021c15225ff)        0x90660021c16a25ff
ulexec  intezer  ~  R2CON2019  section_scramling  $
```

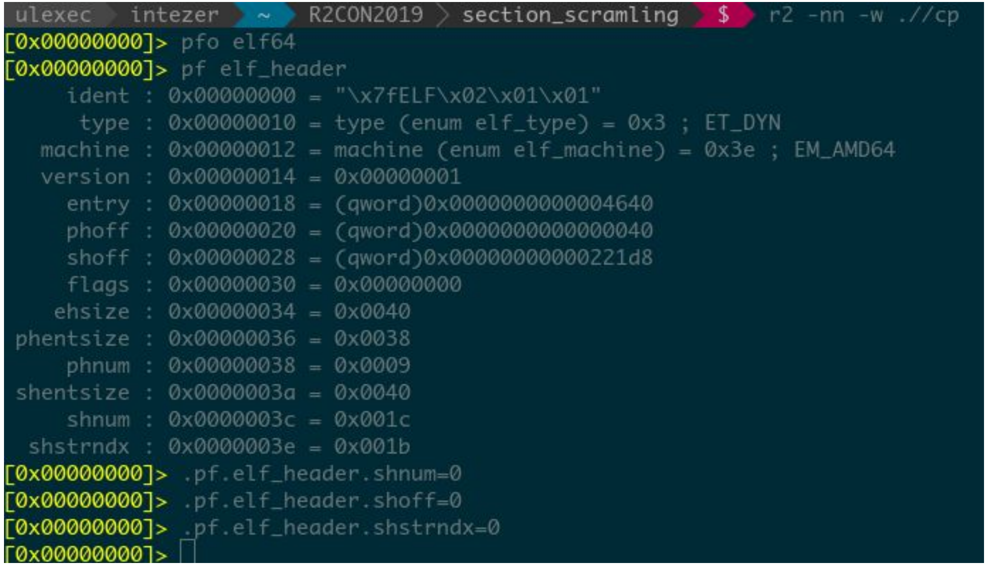
```
ulexec  intezer  ~  R2CON2019  section_scramling  $  objdump -D ./ls
objdump: ./ls: unknown type [0xc] section `'
objdump: ./ls: Bad value
```

Example outcome



Circumvention

- If sections are crafted is better to neglect them
- Sections can be neglected by zeroing out the following fields in the ELF header
    - shoff
    - shnum
    - shstrndx



```
ulexec  intezer  ~  R2CON2019  section_scramling  $  r2 -nn -w .//cp
[0x00000000]> pfo elf64
[0x00000000]> pf elf_header
     ident : 0x00000000 = "\x7fELF\x02\x01\x01"
      type : 0x00000010 = type (enum elf_type) = 0x3 ; ET_DYN
   machine : 0x00000012 = machine (enum elf_machine) = 0x3e ; EM_AMD64
   version : 0x00000014 = 0x00000001
     entry : 0x00000018 = (qword)0x0000000000004640
     phoff : 0x00000020 = (qword)0x0000000000000040
     shoff : 0x00000028 = (qword)0x00000000000221d8
     flags : 0x00000030 = 0x00000000
    ehsize : 0x00000034 = 0x0040
 phentsize : 0x00000036 = 0x0038
     phnum : 0x00000038 = 0x0009
 shentsize : 0x0000003a = 0x0040
     shnum : 0x0000003c = 0x001c
  shstrndx : 0x0000003e = 0x001b
[0x00000000]> .pf.elf_header.shnum=0
[0x00000000]> .pf.elf_header.shoff=0
[0x00000000]> .pf.elf_header.shstrndx=0
[0x00000000]>
```

Circumvention

- Import resolution can be done via .dynstr/.dynsym from PT_DYNAMIC segment entries DT_STRTAB and DT_SYMTAB accordingly

```
Dynamic section at offset 0x1fa38 contains 28 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libselinux.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x3758
 0x000000000000000d (FINI)               0x1636c
 0x0000000000000019 (INIT_ARRAY)         0x21eff0
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x21eff8
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x298
 0x0000000000000005 (STRTAB)             0x1180
 0x0000000000000006 (SYMTAB)             0x388
 0x000000000000000a (STRSZ)              1666 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x21fc38
```

- Export resolution can be done parsing entries from PT_GNU_EH_FRAME segment
  - https://github.com/intezer/scripts/blob/master/eh_frame_parser.py

1 byte parser breaker

- ELF section manipulation can provoke parsing problems.
- ELF fields can be crafted to incorrectly parse a given ELF image.
- Sometimes not much effort is needed to achieve this outcome.

1 Byte Parser Breaker

```
ulexec  intezer  ~  R2CON2019  parser_breaker  $  r2 -w -nn ./ls
[0x00000000]> pfo elf64
[0x00000000]> pf elf_header
     ident : 0x00000000 = "\x7fELF\x02\x01\x01"
      type : 0x00000010 = type (enum elf_type) = 0x3 ; ET_DYN
   machine : 0x00000012 = machine (enum elf_machine) = 0x3e ; EM_AMD64
   version : 0x00000014 = 0x00000001
     entry : 0x00000018 = (qword)0x0000000000005850
     phoff : 0x00000020 = (qword)0x0000000000000040
     shoff : 0x00000028 = (qword)0x00000000000203a0
     flags : 0x00000030 = 0x00000000
    ehsize : 0x00000034 = 0x0040
 phentsize : 0x00000036 = 0x0038
     phnum : 0x00000038 = 0x0009
 shentsize : 0x0000003a = 0x0040
     shnum : 0x0000003c = 0x001c
  shstrndx : 0x0000003e = 0x001b
```

1 Byte Parser Breaker

```
ulexec  intezer  ~  R2CON2019  parser_breaker  $  r2 -w -nn ./ls
[0x00000000]> pfo elf64
[0x00000000]> pf elf_header
     ident : 0x00000000 = "\x7fELF\x02\x01\x01"
      type : 0x00000010 = type (enum elf_type) = 0x3 ; ET_DYN
   machine : 0x00000012 = machine (enum elf_machine) = 0x3e ; EM_AMD64
   version : 0x00000014 = 0x00000001
     entry : 0x00000018 = (qword)0x0000000000005850
     phoff : 0x00000020 = (qword)0x0000000000000040
     shoff : 0x00000028 = (qword)0x00000000000203a0
     flags : 0x00000030 = 0x00000000
    ehsize : 0x00000034 = 0x0040
 phentsize : 0x00000036 = 0x0038
     phnum : 0x00000038 = 0x0009
 shentsize : 0x0000003a = 0x0040
     shnum : 0x0000003c = 0x001c
  shstrndx : 0x0000003e = 0x001b
```

1 Byte Parser Breaker

Array of 16 bytes containing identification flags about the file, which serve to decode and interpret the file's contents. Examples of these identification flags include:

- EI_MAG0-3: ELF magic
- EI_CLASS: File class.
- **EI_DATA: File's data encoding.**
- EI_VERSION: File's version.
- EI_OSABI: OS/ABI identification.
- EI_ABIVERSION: ABI version
- EI_PAD: Start of padding bytes.
- EI_NIDENT: Size of ei_ident.

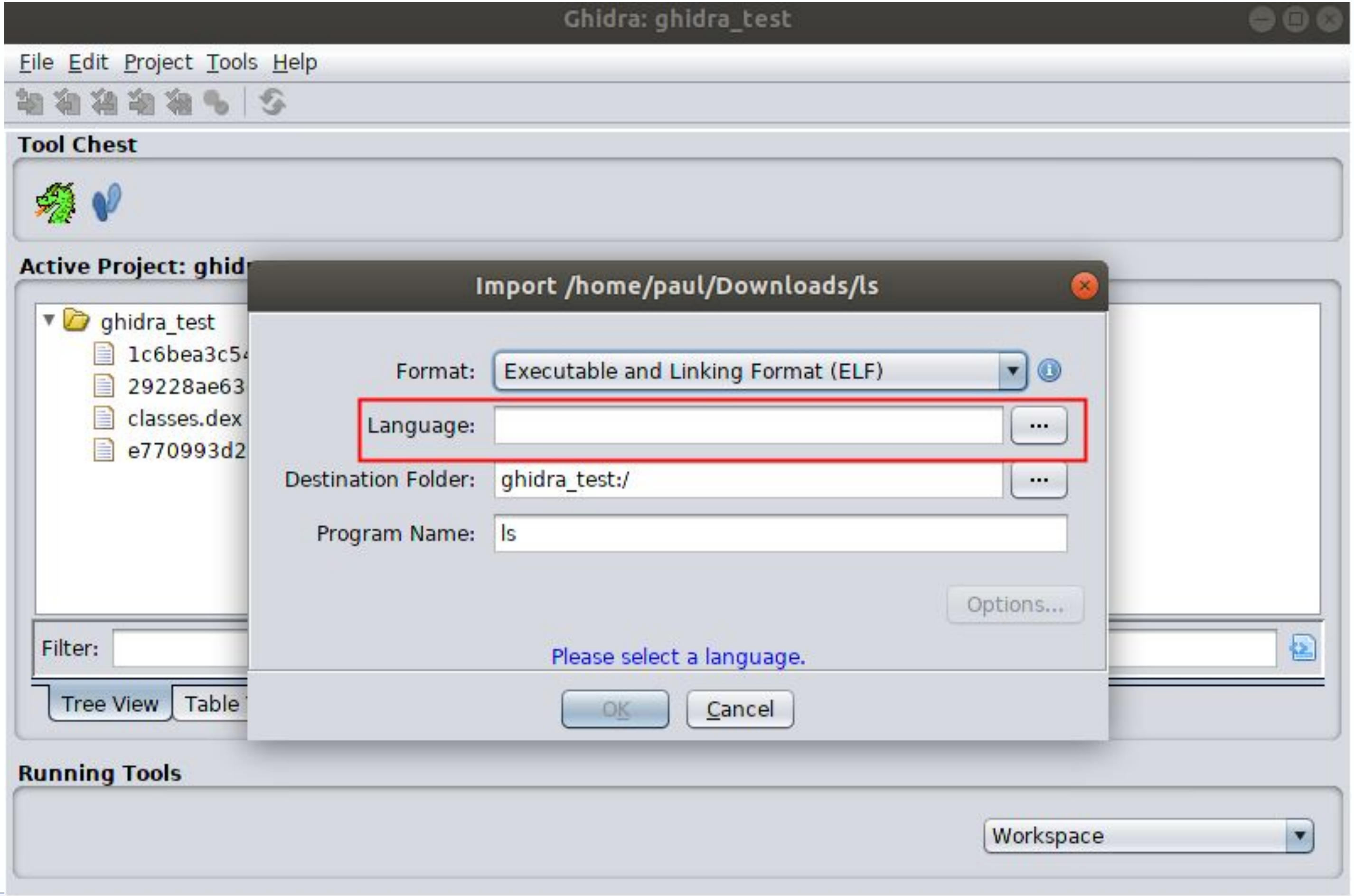
1 Byte Parser Breaker

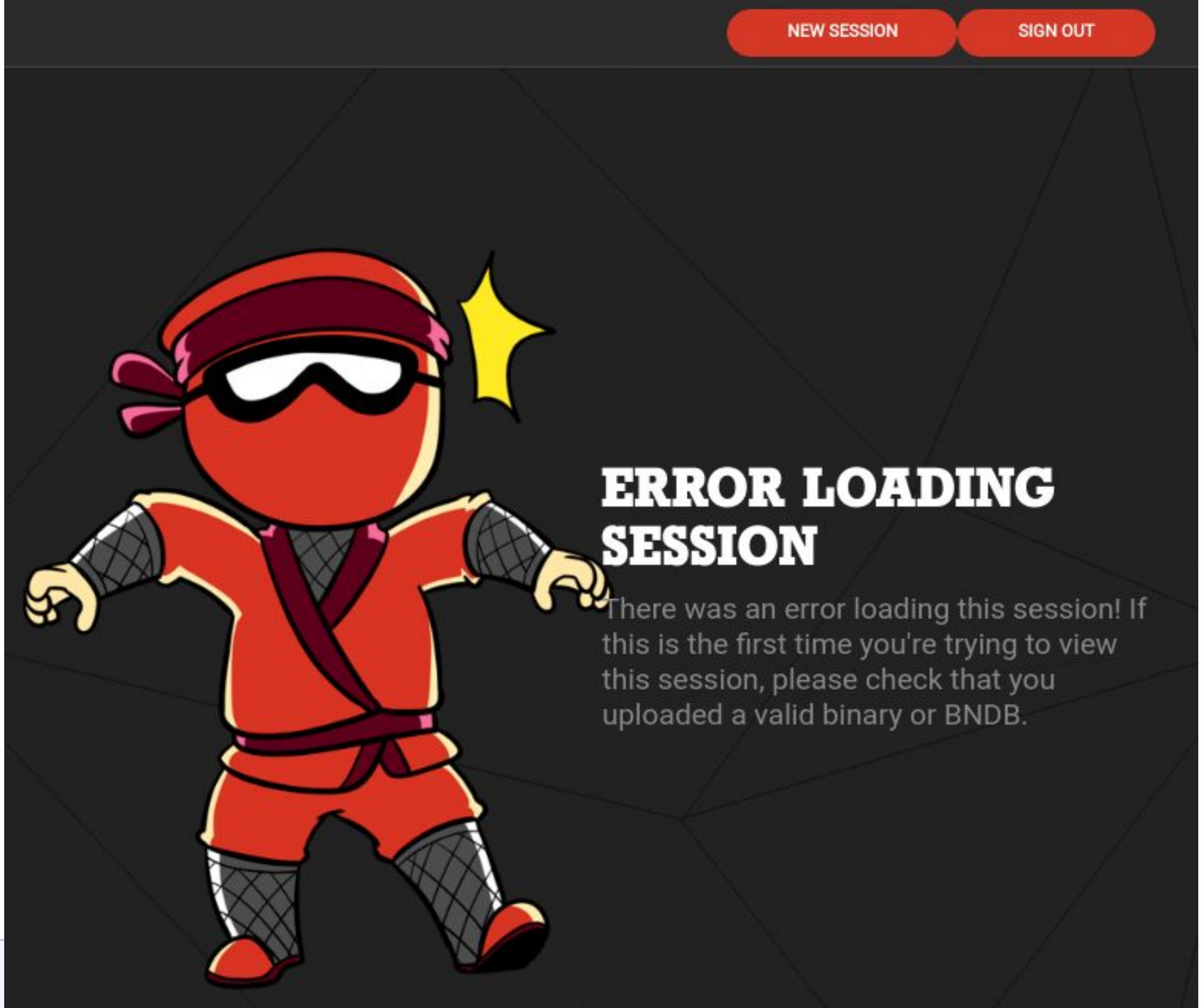
```
[0x00000000]> pf elf_header
     ident :
                struct<elf_ident>
          magic : 0x00000000 = "\x7fELF"
          class : 0x00000004 = class (enum elf_class) = 0x2 ; ELFCLASS64
           data : 0x00000005 = data (enum elf_data) = 0x2 ; ELFDATA2MSB
        version : 0x00000006 = version (enum elf_hdr_version) = 0x1 ; EV_CURRENT
      type : 0x00000010 = type (enum elf_type) = 0x3 ; ET_DYN
   machine : 0x00000012 = machine (enum elf_machine) = 0x3e ; EM_AMD64
   version : 0x00000014 = version (enum elf_obj_version) = 0x1 ; EV_CURRENT
     entry : 0x00000018 = 0x00005850
     phoff : 0x0000001c = 0x00000000
     shoff : 0x00000020 = 0x00000040
     flags : 0x00000024 = 0x00000000
    ehsize : 0x00000028 = 0x03a0
 phentsize : 0x0000002a = 0x0002
     phnum : 0x0000002c = 0x0000
 shentsize : 0x0000002e = 0x0000
     shnum : 0x00000030 = 0x0000
  shstrndx : 0x00000032 = 0x0000
```

1 Byt

```
 ulexec  intezer  ~  R2CON2019  parser_breaker  $  readelf -h ./ls
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:               0x5058000000000000
  Start of program headers:          4611686018427387904 (bytes into file)
  Start of section headers:          -6916682403687694336 (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:         7168
  Section header string table index: 6912
readelf: Warning: The e_shentsize field in the ELF header is larger than the size of an ELF section
 header
readelf: Error: Reading 117440512 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 33030144 bytes extends past end of file for program headers
 ulexec  intezer  ~  R2CON2019  parser_breaker  $  readelf -l ./ls
readelf: Warning: The e_shentsize field in the ELF header is larger than the size of an ELF section
 header
readelf: Error: Reading 117440512 bytes extends past end of file for section headers

Elf file type is <unknown>: 300
Entry point 0x5058000000000000
There are 2304 program headers, starting at offset 4611686018427387904
readelf: Warning: The e_phentsize field in the ELF header is larger than the size of an ELF program
 header
readelf: Error: Reading 33030144 bytes extends past end of file for program headers
 ulexec  intezer  ~  R2CON2019  parser_breaker  $
```


Ghidra: ghidra_test
File Edit Project Tools Help
Tool Chest
Active Project: ghid
ghidra_test
1c6bea3c5
29228ae63
classes.dex
e770993d2
Filter:
Tree View
Table
Import /home/paul/Downloads/ls
Format:
Executable and Linking Format (ELF)
Language:
...
Destination Folder:
ghidra_test:/
...
Program Name:
ls
Options...
Please select a language.
OK
Cancel
Running Tools
Workspace


NEW SESSION
SIGN OUT
ERROR LOADING SESSION
There was an error loading this session! If this is the first time you're trying to view this session, please check that you uploaded a valid binary or BNDB.

```
ulexec  intezer  ~  R2CON2019  parser_breaker  $  r2 ./ls
[0x5058000000000000]> pd 10
            ;-- entry0:
            ;-- rip:
            0x5058000000000000      ff              invalid
            0x5058000000000001      ff              invalid
            0x5058000000000002      ff              invalid
            0x5058000000000003      ff              invalid
            0x5058000000000004      ff              invalid
            0x5058000000000005      ff              invalid
            0x5058000000000006      ff              invalid
            0x5058000000000007      ff              invalid
            0x5058000000000008      ff              invalid
            0x5058000000000009      ff              invalid
[0x5058000000000000]> ii
[Imports]
Num  Vaddr       Bind      Type Name

[0x5058000000000000]> S
|ERROR| Invalid command 'S' (0x53)
[0x5058000000000000]> iS
[Sections]
Nm Paddr       Size Vaddr      Memsz Perms Name
00 0x00000000 133792 0x00010000 133792 -rwx uphdr
```

1 Byte Parser Breaker

- Most static parsers attempt to find the endianness of the file before the machine type.
- The e_machine field will be interpreted according to the endianness field.
- The only disassembler that I’ve seen that handled this anomaly gracefully was IDA.
- This technique could impact **dynamic analysis automation** and **sandbox performance** of the subject ELF file.

1 Byte Parser Breaker

The reason this technique can be enforced without interacting with the kernel's ELF loader is because the endianness is set on compile time in the linux kernel as the CPU endianness.

arch > x86 > include > asm > C elf.h > ELF_DATA

```
typedef struct user_i387_struct elf_fpregset_t;

#ifdef __i386__

typedef struct user_fxsr_struct elf_fpxregset_t;

#define R_386_NONE  0
#define R_386_32    1
#define R_386_PC32  2
#define R_386_GOT32 3
#define R_386_PLT32 4
#define R_386_COPY  5
#define R_386_GLOB_DAT  6
#define R_386_JMP_SLOT  7
#define R_386_RELATIVE  8
#define R_386_GOTOFF    9
#define R_386_GOTPC 10
#define R_386_NUM   11

/*
 * These are used to set parameters in the core dumps.
 */
#define ELF_CLASS   ELFCLASS32
#define ELF_DATA    ELFDATA2LSB
#define ELF_ARCH    EM_386
```

Circumvention

If an ELF file seems to have an unknown architecture and readelf's output seems highly broken,

Restore endianness byte accordingly and attempt further parsing heuristics.

```
ulexec  intezer  ~  R2CON2019  parser_breaker  $  r2 -w -nn ./ls
[0x00000000]> wx 0x01 @+5
[0x00000000]> pfo elf32
[0x00000000]> pf elf_header
     ident :
                      struct<elf_ident>
            magic : 0x00000000 = "\x7fELF"
            class : 0x00000004 = class (enum elf_class) = 0x2 ; ELFCLASS64
             data : 0x00000005 = data (enum elf_data) = 0x1 ; ELFDATA2LSB
          version : 0x00000006 = version (enum elf_hdr_version) = 0x1 ; EV_CURRENT
      type : 0x00000010 = type (enum elf_type) = 0x3 ; ET_DYN
   machine : 0x00000012 = machine (enum elf_machine) = 0x3e ; EM_AMD64
   version : 0x00000014 = version (enum elf_obj_version) = 0x1 ; EV_CURRENT
     entry : 0x00000018 = 0x00005850
     phoff : 0x0000001c = 0x00000000
     shoff : 0x00000020 = 0x00000040
     flags : 0x00000024 = 0x00000000
    ehsize : 0x00000028 = 0x03a0
 phentsize : 0x0000002a = 0x0002
     phnum : 0x0000002c = 0x0000
 shentsize : 0x0000002e = 0x0000
     shnum : 0x00000030 = 0x0000
  shstrndx : 0x00000032 = 0x0000
```

Hiding dynamic entries

- Dynamically compiled ELF files will always contain a segment of type PT_DYNAMIC
- This segment contains all needed information needed for the dynamic linking \ to take place.
- Dynamically linked binaries with a defectuous dynamic segment will crash

Hiding dynamic entries

- Android packer scene is much more mature in regards to evasion techniques than standalone ELF malware is.
- Most ELFs found in Android applications tend to be shared objects.
- Successfully analysis of shared objects heavily relies on visibility of dynamic linking artifacts. (init_array, exports, ... etc)
- The following technique implements a way to hide dynamic entries from ELF shared objects from the PT_DYNAMIC segment

Detecting anomalies

```
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  r2 libSDKRelativeJNI.so
[0x00000000]> aa
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00000000]> is
[Symbols]
Num Paddr       Vaddr       Bind      Type Size Name

[0x00000000]> !readelf -d ./libSDKRelativeJNI.so
readelf: Error: no .dynamic section in the dynamic segment

Dynamic section at offset 0x41464 contains 11 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     0x8328
 0x00000001 (NEEDED)                     0x8337
 0x00000001 (NEEDED)                     0x8347
 0x00000001 (NEEDED)                     0x8351
 0x00000001 (NEEDED)                     0x835e
 0x00000001 (NEEDED)                     0x8366
 0x00000001 (NEEDED)                     0x8373
 0x00000001 (NEEDED)                     0x837b
 0x00000001 (NEEDED)                     0x8383
 0x0000000e (SONAME)                     0x838c
 0x00000000 (NULL)                       0x0
[0x00000000]>
```

No symbols

Section parsing error

Weird looking Dynamic entries

Trying to circumvent anomalies

```
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  r2 -w -nn libSDKRelativeJNI_.so
[0x00000000]> pfo elf32
[0x00000000]> pf elf_header
     ident :
                struct<elf_ident>
         magic : 0x00000000 = "\x7fELF"
         class : 0x00000004 = class (enum elf_class) = 0x1 ; ELFCLASS32
          data : 0x00000005 = data (enum elf_data) = 0x1 ; ELFDATA2LSB
       version : 0x00000006 = version (enum elf_hdr_version) = 0x1 ; EV_CURRENT
      type : 0x00000010 = type (enum elf_type) = 0x3 ; ET_DYN
   machine : 0x00000012 = machine (enum elf_machine) = 0x28 ; EM_ARM
   version : 0x00000014 = version (enum elf_obj_version) = 0x1 ; EV_CURRENT
     entry : 0x00000018 = 0x00000000
     phoff : 0x0000001c = 0x00000034
     shoff : 0x00000020 = 0x000414e4
     flags : 0x00000024 = 0x05000000
    ehsize : 0x00000028 = 0x0034
 phentsize : 0x0000002a = 0x0020
     phnum : 0x0000002c = 0x0004
 shentsize : 0x0000002e = 0x0028
     shnum : 0x00000030 = 0x0004
  shstrndx : 0x00000032 = 0x0001
[0x00000000]> .pf.elf_header.shnum=0
[0x00000000]> .pf.elf_header.shoff=0
[0x00000000]> .pf.elf_header.shstrndx=0
```

More anomalies

```
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  readelf -h ./libSDKRelativeJNI_.so

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:                              DYN (Shared object file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          0 (bytes into file)
  Flags:                             0x5000000, Version5 EABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         4
  Size of section headers:           40 (bytes)
  Number of section headers:         0
  Section header string table index: 0
readelf: Error: the dynamic segment offset + size exceeds the size of the file
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  readelf -d ./libSDKRelativeJNI_.so

readelf: Error: the dynamic segment offset + size exceeds the size of the file

There is no dynamic section in this file.
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  python -c 'print "/x00" * 100' >>
./libSDKRelativeJNI_.so
```

No profit

```
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  readelf -d ./libSDKRelativeJNI_.so


Dynamic section at offset 0x41464 contains 11 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     0x8328
 0x00000001 (NEEDED)                     0x8337
 0x00000001 (NEEDED)                     0x8347
 0x00000001 (NEEDED)                     0x8351
 0x00000001 (NEEDED)                     0x835e
 0x00000001 (NEEDED)                     0x8366
 0x00000001 (NEEDED)                     0x8373
 0x00000001 (NEEDED)                     0x837b
 0x00000001 (NEEDED)                     0x8383
 0x0000000e (SONAME)                     0x838c
 0x00000000 (NULL)                       0x0
```

Focusing on Program Headers

```
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  readelf -l ./libSDKRelativeJNI_.so

Elf file type is DYN (Shared object file)
Entry point 0x0
There are 4 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x00000000 0x00000000 0x3db60 0x7e91c R E 0x1000
  LOAD           0x03e688 0x00080688 0x00080688 0x02dcc 0x08c00 RW  0x1000
  DYNAMIC        0x041464 0x00082ac8 0x00082ac8 0x00120 0x00120 RW  0x4
  EXIDX          0x07504c 0x0007504c 0x0007504c 0x03968 0x03968 R   0x4
```

Focusing on Program Headers

```
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  readelf -l ./libSDKRelativeJNI_.so


Elf file type is DYN (Shared object file)
Entry point 0x0
There are 4 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x00000000 0x00000000 0x3db60 0x7e91c R E 0x1000
  LOAD           0x03e688 0x00080688 0x00080688 0x02dcc 0x08c00 RW  0x1000
  DYNAMIC        0x041464 0x00082ac8 0x00082ac8 0x00120 0x00120 RW  0x4
  EXIDX          0x07504c 0x0007504c 0x0007504c 0x03968 0x03968 R   0x4
```

Inconsistencies between memory and disk layouts

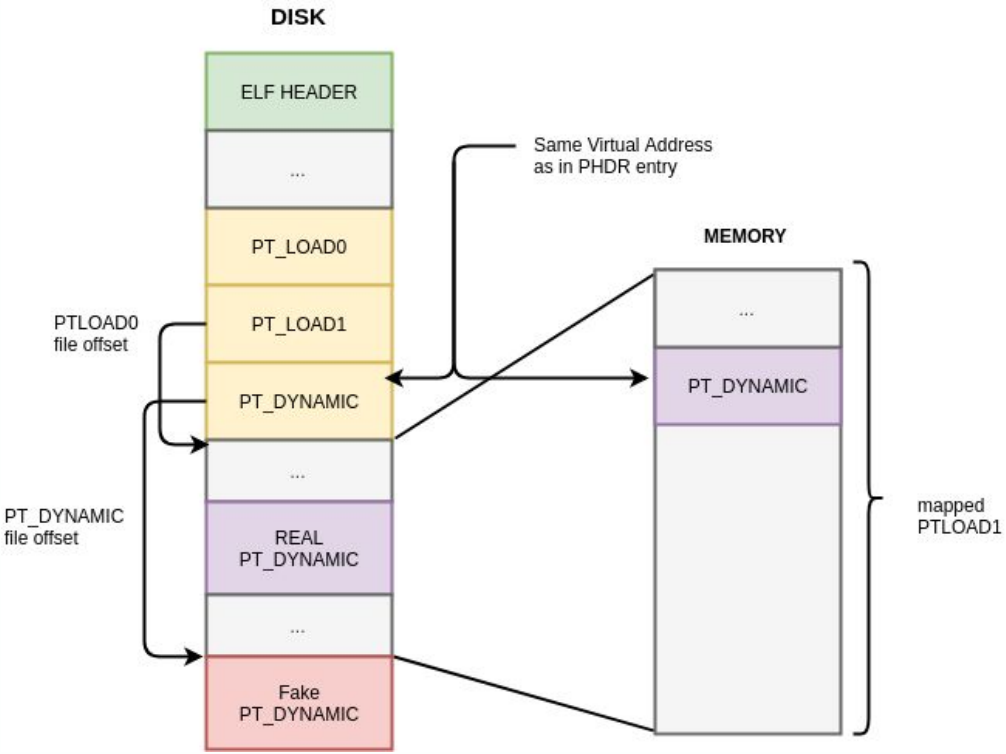
```
[0x00082ac8 [Xadvc] 95% 1848 ./libSDKRelativeJNI_.so]> xc @ segment.DYNAMIC
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF  comment
0x00082ac8  0300 0000 982e 0800 0200 0000 b802 0000  ................  ; segment.DYNAMIC
0x00082ad8  1700 0000 1c11 0100 1400 0000 1100 0000  ................
0x00082ae8  1100 0000 fcd5 0000 1200 0000 203b 0000  ............ ;..
0x00082af8  1300 0000 0800 0000 faff ff6f 6007 0000  ...........o`...
0x00082b08  0600 0000 4801 0000 0b00 0000 1000 0000  ....H...........
0x00082b18  0500 0000 983b 0000 0a00 0000 a183 0000  .....;..........
0x00082b28  0400 0000 3cbf 0000 0100 0000 2883 0000  ....<.......(...
0x00082b38  0100 0000 3783 0000 0100 0000 4783 0000  ....7.......G...
0x00082b48  0100 0000 5183 0000 0100 0000 5e83 0000  ....Q.......^...
0x00082b58  0100 0000 6683 0000 0100 0000 7383 0000  ....f.......s...
0x00082b68  0100 0000 7b83 0000 0100 0000 8383 0000  ....{...........
```

Memory

```
[0x00041464 [Xadvc] 0% 1848 ./libSDKRelativeJNI_.so]> xc @ elf_phdr+267312 # 0x41464
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF  comment
0x00041464  0100 0000 2883 0000 0100 0000 3783 0000  ....(.......7...
0x00041474  0100 0000 4783 0000 0100 0000 5183 0000  ....G.......Q...
0x00041484  0100 0000 5e83 0000 0100 0000 6683 0000  ....^.......f...
0x00041494  0100 0000 7383 0000 0100 0000 7b83 0000  ....s.......{...
0x000414a4  0100 0000 8383 0000 0e00 0000 8c83 0000  ................
0x000414b4  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000414c4  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000414d4  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000414e4  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x000414f4  0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Disk

Dynamic segment obfuscation technique



Circumvention

- This technique can be detected checking if the following heuristic does not match:

  dynamic segment offset = data segment offset
  + (vaddr dynamic segment - vaddr data segment)

Circumvention

```
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  $  r2 -nn -w libSDKRelativeJNI_.so
[0x00000000]> !readelf -l ./libSDKRelativeJNI_.so

Elf file type is DYN (Shared object file)
Entry point 0x0
There are 4 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x00000000 0x00000000 0x3db60 0x7e91c R E 0x1000
  LOAD           0x03e688 0x00080688 0x00080688 0x02dcc 0x08c00 RW  0x1000
  DYNAMIC        0x041464 0x00082ac8 0x00082ac8 0x00120 0x00120 RW  0x4
  EXIDX          0x07504c 0x0007504c 0x0007504c 0x03968 0x03968 R   0x4
[0x00000000]> ?v 0x03e688 + (0x00082ac8 - 0x00080688)
0x40ac8
[0x00000000]> pfo elf32
[0x00000000]> pf.elf_header.phoff
     phoff : 0x0000001c = 0x00000034
[0x00000000]> pf.elf_header.phentsize
 phentsize : 0x0000002a = 0x0020
[0x00000000]> s 0x34 + (0x20 * 2)
[0x00000074]> pf.elf_phdr
   type : 0x00000074 = type (enum elf_p_type) = 0x2 ; PT_DYNAMIC
 offset : 0x00000078 = 0x00041464
  vaddr : 0x0000007c = 0x00082ac8
  paddr : 0x00000080 = 0x00082ac8
 filesz : 0x00000084 = 0x00000120
  memsz : 0x00000088 = 0x00000120
  flags : 0x0000008c = flags (enum elf_p_flags) = 0x6 ; PF_Read_Write
  align : 0x00000090 = 0x00000004
[0x00000074]> .pf.elf_phdr.offset=0x40ac8
```

Profit

```
Dynamic section at offset 0x40ac8 contains 32 entries:
  Tag        Type                         Name/Value
 0x00000003 (PLTGOT)                     0x82e98
 0x00000002 (PLTRELSZ)                   696 (bytes)
 0x00000017 (JMPREL)                     0x1111c
 0x00000014 (PLTREL)                     REL
 0x00000011 (REL)                        0xd5fc
 0x00000012 (RELSZ)                      15136 (bytes)
 0x00000013 (RELENT)                     8 (bytes)
 0x6ffffffa (RELCOUNT)                   1888
 0x00000006 (SYMTAB)                     0x148
 0x0000000b (SYMENT)                     16 (bytes)
 0x00000005 (STRTAB)                     0x3b98
 0x0000000a (STRSZ)                      33697 (bytes)
 0x00000004 (HASH)                       0xbf3c
 0x00000001 (NEEDED)                     Shared library: [libdjivideo.so]
 0x00000001 (NEEDED)                     Shared library: [libGLESv1_CM.so]
 0x00000001 (NEEDED)                     Shared library: [liblog.so]
 0x00000001 (NEEDED)                     Shared library: [libGLESv2.so]
 0x00000001 (NEEDED)                     Shared library: [libz.so]
 0x00000001 (NEEDED)                     Shared library: [libstdc++.so]
 0x00000001 (NEEDED)                     Shared library: [libm.so]
 0x00000001 (NEEDED)                     Shared library: [libc.so]
 0x00000001 (NEEDED)                     Shared library: [libdl.so]
 0x0000000e (SONAME)                     Library soname: [libSDKRelativeJNI.so]
 0x0000001a (FINI_ARRAY)                 0x80de0
 0x0000001c (FINI_ARRAYSZ)               8 (bytes)
 0x00000019 (INIT_ARRAY)                 0x80de8
 0x0000001b (INIT_ARRAYSZ)               32 (bytes)
 0x00000010 (SYMBOLIC)                   0x0
 0x0000001e (FLAGS)                      SYMBOLIC BIND_NOW
 0x6ffffffb (FLAGS_1)                    Flags: NOW
 0x0000000c (INIT)                       0x113d8
 0x00000000 (NULL)                       0x0
```

Profit

```
readelf: Error: no .dynamic section in the dynamic segment

Dynamic section at offset 0x40ac8 contains 32 entries:
  Tag        Type                         Name/Value
 0x00000003 (PLTGOT)                     0x82e98
 0x00000002 (PLTRELSZ)                   696 (bytes)
 0x00000017 (JMPREL)                     0x1111c
 0x00000014 (PLTREL)                     REL
 0x00000011 (REL)                        0xd5fc
 0x00000012 (RELSZ)                      15136 (bytes)
 0x00000013 (RELENT)                     8 (bytes)
 0x6ffffffa (RELCOUNT)                   1888
 0x00000006 (SYMTAB)                     0x148
 0x0000000b (SYMENT)                     16 (bytes)
 0x00000005 (STRTAB)                     0x3b98
 0x0000000a (STRSZ)                      33697 (bytes)
```

```
ulexec  intezer  ~  R2CON2019  hidden_dynamic_entries  1  $  r2 libSDKRelativeJNI.so
[0x00000000]> is~?
95
```

```
 0x00000001 (NEEDED)                     Shared library: [libGLESv2.so]
 0x00000001 (NEEDED)                     Shared library: [libz.so]
 0x00000001 (NEEDED)                     Shared library: [libstdc++.so]
 0x00000001 (NEEDED)                     Shared library: [libm.so]
 0x00000001 (NEEDED)                     Shared library: [libc.so]
 0x00000001 (NEEDED)                     Shared library: [libdl.so]
 0x0000000e (SONAME)                     Library soname: [libSDKRelativeJNI.so]
 0x0000001a (FINI_ARRAY)                 0x80de0
 0x0000001c (FINI_ARRAYSZ)               8 (bytes)
 0x00000019 (INIT_ARRAY)                 0x80de8
 0x0000001b (INIT_ARRAYSZ)               32 (bytes)
 0x00000010 (SYMBOLIC)                   0x0
 0x0000001e (FLAGS)                      SYMBOLIC BIND_NOW
 0x6ffffffb (FLAGS_1)                    Flags: NOW
 0x0000000c (INIT)                       0x113d8
 0x00000000 (NULL)                       0x0
```

Relocation Hijacking

Relocation Hijacking

- ELF files contain a very sophisticated relocation system.
- Relocation information is held in entries located in specific relocation sections within an ELF object.
- These entries are implemented in the form of structures. There are two different Relocation entry structures: Elfxx_Rel and Elfxx_Rela:

Relocation Hijacking

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
} Elf32_Rel;

typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
        Elf32_Sword     r_addend;
} Elf32_Rela;
```

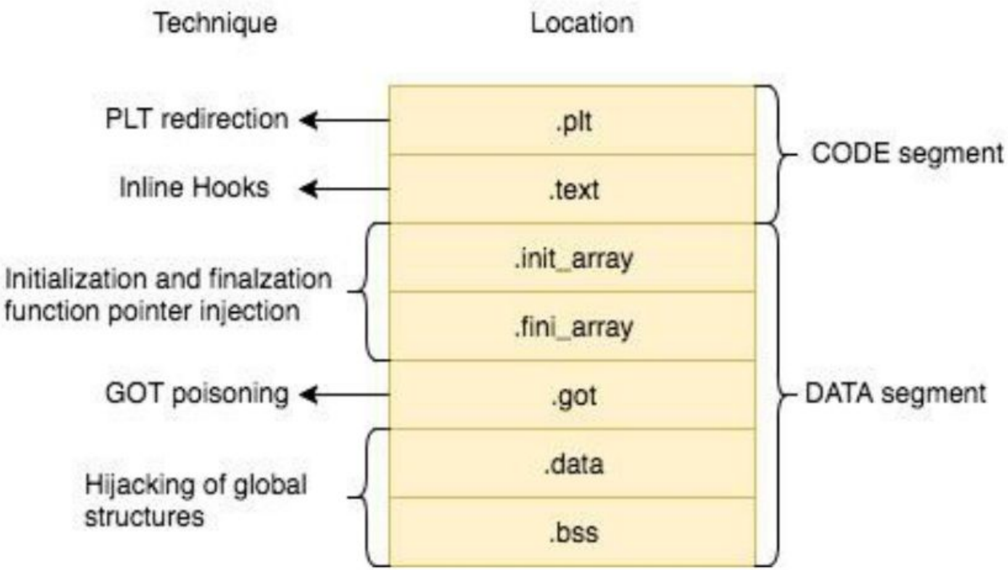
Relocation Hijacking

- Code injection techniques for the ELF file format exists to build custom tools or craft more complex implants.

- However there is still a challenge of pivoting control of execution into the injected payload.

- The are well known techniques abused in the past in order to do this.

Relocation Hijacking



Relocation Hijacking

- These techniques are known for a long time and easily detectable

- However there are ways to hijack relocations in order to pivot control flow in more stealthy ways.

- Two ideas:
    - EPO based on relocation tampering for PIE binaries
    - Hiding constructors in Dynamically and Statically linked binaries

Challenges on malicious relocations

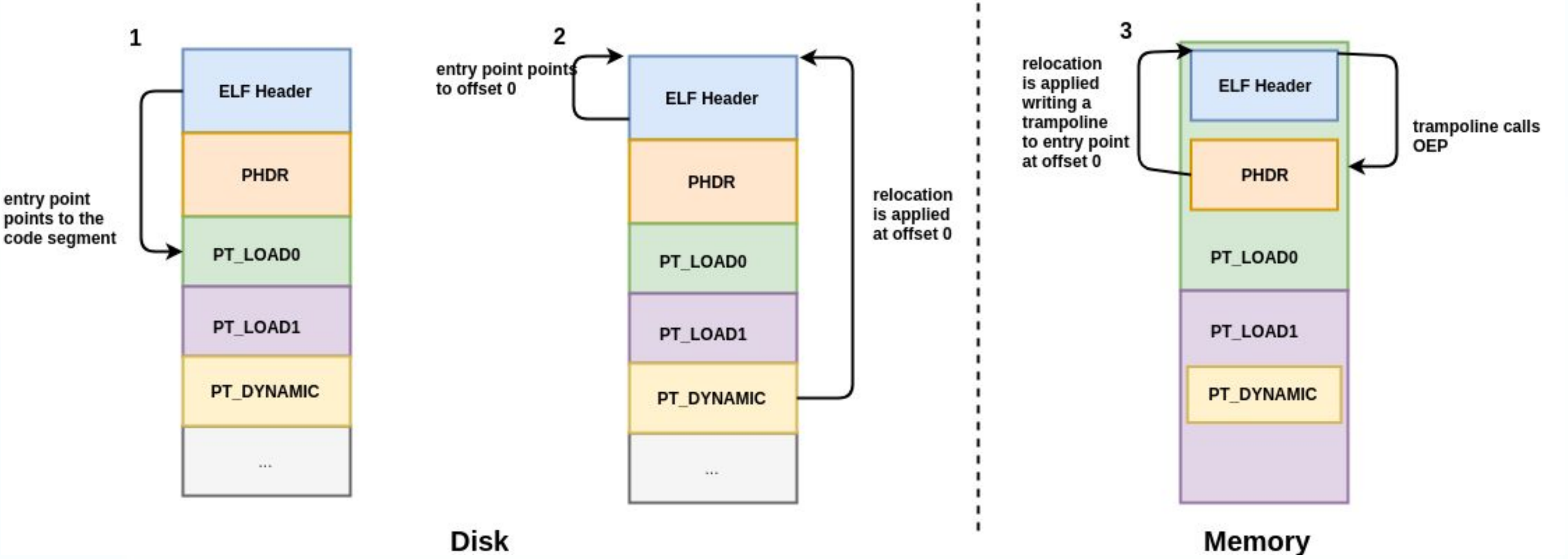
- Relocations essentially enforce hot-patching the binary on load-time.
- Relocation analysis is hard.
- Advanced heuristics have to be applied in order to recognise if a relocation has malicious intentions.
- Relocations are easy to tamper.

EPO on PIE executables

- PIE executables are ET_DYN binaries
- Image base will be chosen on load-time
- Virtual addresses in disk are relative to the chosen image base
- In disk the file’s image base is 0.

Demo

Overview



INTEZER

Overview

```c
for (i = 0; i < relasz; i++, rela++) {
        Elf64_Sym *sym = &dynsym[ELF64_R_SYM(rela->r_info)];
        if (!strcmp(&dynstr[sym->st_name], symbol_name)) {
                rela->r_offset = 0;
                rela->r_info = (rela->r_info & ~0xf) | R_X86_64_64;
                sym->st_value = 123456;
                rela->r_addend = (((reloc_value - sym->st_value)
                                + sym->st_value << 8)
                                | 0xe9);
                printf("[+] Rela entry for symbol: %s was modified\n", symbol_name);
                return true;
        }
}
```

Detection

- Check for relocations being applied at offset 0 on PIE binaries.
- This technique requires the text segment to be writable

```
00000021ffc8  000700000006 R_X86_64_GLOB_DAT 0000000000000000 free@GLIBC_2.2.5 + 0
00000021ffd0  000b00000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_deregisterTMClone + 0
00000021ffd8  003800000006 R_X86_64_GLOB_DAT 0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
000000000000  004400000001 R_X86_64_64       000000000001e240 __gmon_start__ + 5850e9
00000021ffe8  004e00000006 R_X86_64_GLOB_DAT 0000000000000000 malloc@GLIBC_2.2.5 + 0
00000021fff0  006e00000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_registerTMCloneTa + 0
00000021fff8  007200000006 R_X86_64_GLOB_DAT 0000000000000000 __cxa_finalize@GLIBC_2.2.5 + 0
000000220280  007700000005 R_X86_64_COPY     0000000000220280 __progname@GLIBC_2.2.5 + 0
000000220288  009300000005 R_X86_64_COPY     0000000000220288 stdout@GLIBC_2.2.5 + 0
```

Hiding Constructors in Dynamic and Static executables

Let’s imagine we are an attacker and we would like to execute a constructor before reaching our main payload.

Constructors are easy to spot if we analyse the dynamic segment in dynamic executables

```
Dynamic section at offset 0xdc0 contains 27 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x528
 0x000000000000000d (FINI)               0x7d4
 0x0000000000000019 (INIT_ARRAY)         0x200db0
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x200db8
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
```

Hiding Constructors in Dynamic and Static executables

- Would there be a way to hide used constructors from plain sight?
- Yes. We can we relocations

Technique Overview

Abusing IRELATIV relocations on static ELF binaries

- One could guess that hijacking Statically linked binaries would be as trivial as for dynamically linked binaries.

- However, this is not the case since statically linked executables only support one type of relocations, those being R_IRELATIVE.

What are IRELATIV relocations

From https://sites.google.com/site/x32abi/documents/ifunc.txt

```
This relocation is similar to R_*_RELATIVE except that the
value used in this relocation is the program address returned
by the function, which takes no arguments, at the address of
the result of the corresponding R_*_RELATIVE relocation as
specified in the processor-specific ABI.

The purpose of this relocation to avoid name lookup for locally
defined STT_GNU_IFUNC symbols at load-time.
```

/glibc/sysdeps/x86_64/dl-irel.h

```c
#define ELF_MACHINE_IRELA        1

static inline ElfW(Addr)
__attribute ((always_inline))
elf_ifunc_invoke (ElfW(Addr) addr)
{
  return ((ElfW(Addr) (*) (void)) (addr)) ();
}

static inline void
__attribute ((always_inline))
elf_irela (const ElfW(Rela) *reloc)
{
  ElfW(Addr) *const reloc_addr = (void *) reloc->r_offset;
  const unsigned long int r_type = ELFW(R_TYPE) (reloc->r_info);

  if (__glibc_likely (r_type == R_X86_64_IRELATIVE))
    {
      ElfW(Addr) value = elf_ifunc_invoke(reloc->r_addend);
      *reloc_addr = value;
    }
  else
    __libc_fatal ("Unexpected reloc type in static binary.\n");
}
```

```c
#define ELF_MACHINE_IRELA       1

static inline ElfW(Addr)
__attribute ((always_inline))
elf_ifunc_invoke (ElfW(Addr) addr)
{
  return ((ElfW(Addr) (*) (void)) (addr)) ();
}

static inline void
__attribute ((always_inline))
elf_irela (const ElfW(Rela) *reloc)
{
  ElfW(Addr) *const reloc_addr = (void *) reloc->r_offset;
  const unsigned long int r_type = ELFW(R_TYPE) (reloc->r_info);

  if (__glibc_likely (r_type == R_X86_64_IRELATIVE))
    {
      ElfW(Addr) value = elf_ifunc_invoke(reloc->r_addend);
      *reloc_addr = value;
    }
  else
    __libc_fatal ("Unexpected reloc type in static binary.\n");
}
```

```c
#define ELF_MACHINE_IRELA        1

static inline ElfW(Addr)
__attribute ((always_inline))
elf_ifunc_invoke (ElfW(Addr) addr)
{
  return ((ElfW(Addr) (*) (void)) (addr)) ();
}

static inline void
__attribute ((always_inline))
elf_irela (const ElfW(Rela) *reloc)
{
  ElfW(Addr) *const reloc_addr = (void *) reloc->r_offset;
  const unsigned long int r_type = ELFW(R_TYPE) (reloc->r_info);

  if (__glibc_likely (r_type == R_X86_64_IRELATIVE))
    {
      ElfW(Addr) value = elf_ifunc_invoke(reloc->r_addend);
      *reloc_addr = value;
    }
  else
    __libc_fatal ("Unexpected reloc type in static binary.\n");
}
```

How these ifuncs look like

```
[0x00423480]> ir~0x00423480
vaddr=0x006b9018 paddr=0x000b9018 type=SET_64 0x00423480 (ifunc)
[0x00423480]> pdf
            ;-- strcpy:
/ (fcn) sym.strcpy_ifunc 43
|   sym.strcpy_ifunc ();
|           0x00423480      f605a5892900.  test byte [0x006bbe2c], 0x10
|           0x00423487      488d05020602.  lea rax, sym.__strcpy_sse2_unaligned
|       ,=< 0x0042348e      7519           jne 0x4234a9
|       |   0x00423490      f6055a892900.  test byte [0x006bbdf1], 2
|       |   0x00423497      488d0572d401.  lea rax, sym.__strcpy_sse2
|       |   0x0042349e      488d152bd601.  lea rdx, sym.__strcpy_ssse3
|       |   0x004234a5      480f45c2       cmovne rax, rdx
\       `-> 0x004234a9      f3c3           ret
[0x00423480]>
```

An approach for IRELATIV Reloc hijacking

1 - Emulate all/specific ifuncs and populate it correspondent GOT entry with the function result.

2 - All IRELATIV relocations now are free to be tampered.

3 - Create a fake ifunc function that returns the address of some arbitrary malicious code.

4 - change an IRELATIV relocation entry so that the addend points to the address of the fake ifunc function.

5 - change the offset of the relocation to point to a known constructor/destructor data structure.

6 - ???

7 - profit.

Implementation with Radare2

```
import r2pipe
import sys

def main(file_path):
    r = r2pipe.open(file_path, flags=['-w'])
    relocs_value = r.cmd('ir~[6]').split()
    relocs_offset = r.cmd('ir~[1]').split()

    print("[+] Resolving all ifunc relocations");
    for value, offset in zip(relocs_value, relocs_offset):
        r.cmd("s %s" % value)
        r.cmd("aei")
        r.cmd("aeip")
        r.cmd("aesuo ret")
        resolved_ifunc = r.cmd("aer rax").strip()
        print("[+] Writing %s at %s" % (resolved_ifunc, offset))
        r.cmd("wv8 %s @%s" % (resolved_ifunc, offset))
        print r.cmd("pxq 8 @ %s" % offset);
        r.cmd("aei-")

    dummy_pointer = r.cmd("is~__fini_array_start~[2]").strip()
    payload_address = r.cmd("is~fake_ireloc~[2]").strip()

    print("[+] Creating new IRELATIV relocation (%s, %s)" % (dummy_pointer, payload_address))
    r.cmd("s section..rela.plt")
    r.cmd('wv8 %s @ $$ ' % dummy_pointer);
    r.cmd('wv8 %s @ $$+0x10' % payload_address);

if __name__ == '__main__':
    main(sys.argv[1])
```

Demo

Detection on dynamic executables

Before

```
Relocation section '.rela.dyn' at offset 0x438 contains 8 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000200db0  000000000008 R_X86_64_RELATIVE                    680
000000200db8  000000000008 R_X86_64_RELATIVE                    640
000000201008  000000000008 R_X86_64_RELATIVE                    201008
000000200fd8  000100000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_deregisterTMClone + 0
000000200fe0  000300000006 R_X86_64_GLOB_DAT 0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
000000200fe8  000400000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
000000200ff0  000600000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_registerTMCloneTa + 0
000000200ff8  000700000006 R_X86_64_GLOB_DAT 0000000000000000 __cxa_finalize@GLIBC_2.2.5 + 0

Relocation section '.rela.plt' at offset 0x4f8 contains 2 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000200fc8  000200000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
000000200fd0  000500000007 R_X86_64_JUMP_SLO 0000000000000000 mprotect@GLIBC_2.2.5 + 0
```

Detection on dynamic executables

After

```
Relocation section '.rela.dyn' at offset 0x438 contains 8 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000200db0  000000000008 R_X86_64_RELATIVE                    6ac
000000200db8  000000000008 R_X86_64_RELATIVE                    640
000000201008  000000000008 R_X86_64_RELATIVE                    201008
000000200fd8  000100000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_deregisterTMClone + 0
000000200fe0  000300000006 R_X86_64_GLOB_DAT 0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
000000200fe8  000400000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
000000200ff0  000600000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_registerTMCloneTa + 0
000000200ff8  000700000006 R_X86_64_GLOB_DAT 0000000000000000 __cxa_finalize@GLIBC_2.2.5 + 0

Relocation section '.rela.plt' at offset 0x4f8 contains 2 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000200fc8  000200000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
000000200fd0  000500000007 R_X86_64_JUMP_SLO 0000000000000000 mprotect@GLIBC_2.2.5 + 0
```

Detection on dynamic executables

After

```
Relocation section '.rela.dyn' at offset 0x438 contains 8 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000200db0  000000000008 R_X86_64_RELATIVE                    6ac
000000200db8  000000000008 R_X86_64_RELATIVE                    640
000000201008  000000000008 R_X86_64_RELATIVE                    201008
000000200fd8  000100000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_deregisterTMClone + 0
000000200fe0  000300000006 R_X86_64_GLOB_DAT 0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
000000200fe8  000400000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
000000200ff0  000600000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_registerTMCloneTa + 0
000000200ff8  000700000006 R_X86_64_GLOB_DAT 0000000000000000 __cxa_finalize@GLIBC_2.2.5 + 0

Relocation section '.rela.plt' at offset 0x4f8 contains 2 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000200fc8  000200000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
000000200fd0  000500000007 R_X86_64_JUMP_SLO 0000000000000000 mprotect@GLIBC_2.2.5 + 0
```

Detection on static executables

- Check where IRELATIV relocations points to.
- They all should point to STT_IFUNC type symbols.

Problem: these can be spotted thanks to the symbol table (STT_IFUNC symbols) however string/symbol tables can completely removed/instrumented in statically linked executables.

Experimental Practices

Removing .dynstr string table

- .Dynstr string table is the string table of .dynsym symbol table
- These two artifacts are needed on runtime by the Dynamic Linker
- Usually other string tables can be stripped but not .dynstr
- All dynamically linked executables have them

The following technique will explain an approach on how this string table can be removed to leverage an anti-analysis technique

How imports get resolved

- Imports get resolved thanks to the PLT mechanism implemented in most dynamic executable ELF files.

- There is a way for a dynamic executable to not contain a PLT, although is not common.

How imports get resolved

```
            ;-- section..plt:
            0x00003770      ff35cac42100   push qword [0x0021fc40]
            0x00003776      ff25ccc42100   jmp qword [0x0021fc48]
            0x0000377c      0f1f4000       nop dword [rax]
            ;-- imp.__ctype_toupper_loc:
            0x00003780      ff25cac42100   jmp qword reloc.__ctype_toupper_loc
            0x00003786      6800000000     push 0
            0x0000378b      e9e0ffffff     jmp section..plt
            ;-- imp.__uflow:
            0x00003790      ff25c2c42100   jmp qword reloc.__uflow
            0x00003796      6801000000     push 1
            0x0000379b      e9d0ffffff     jmp section..plt
            ;-- imp.getenv:
            0x000037a0      ff25bac42100   jmp qword reloc.getenv
            0x000037a6      6802000000     push 2
            0x000037ab      e9c0ffffff     jmp section..plt
            ;-- imp.sigprocmask:
            0x000037b0      ff25b2c42100   jmp qword reloc.sigprocmask
            0x000037b6      6803000000     push 3
            0x000037bb      e9b0ffffff     jmp section..plt
[0x00003770]>
```

How imports get resolved

First 3 GOT entries have reserved values.

- GOT[0] - Virtual address of the image’s DYNAMIC segment
- GOT[1] - Virtual address of link_map
- GOT[2] - Virtual address of _dl_runtime_resolve

```
;-- section..plt:
0x00003770      ff35cac42100      push qword [0x0021fc40]
0x00003776      ff25ccc42100      jmp qword [0x0021fc48]
0x0000377c      0f1f4000          nop dword [rax]
```

GOT[1] - **link_map**

GOT[2] - **_dl_runtime_resolve**

```
;-- imp.__ctype_toupper_loc:
0x00003780      ff25cac42100      jmp qword reloc.__ctype_toupper_loc
0x00003786      6800000000        push 0
0x0000378b      e9e0ffffff        jmp section..plt
```

Relocation index in .rela.plt

How imports get resolved

```
      result = _dl_lookup_symbol_x (strtab + sym->st_name, l, &sym, l->l_scope,
                                    version, ELF_RTYPE_CLASS_PLT, flags, NULL);

      /* We are done with the global scope.  */
      if (!RTLD_SINGLE_THREAD_P)
        THREAD_GSCOPE_RESET_FLAG ();

#ifdef RTLD_FINALIZE_FOREIGN_CALL
      RTLD_FINALIZE_FOREIGN_CALL;
#endif

      /* Currently result contains the base load address (or link map)
         of the object that defines sym.  Now add in the symbol
         offset.  */
      value = DL_FIXUP_MAKE_VALUE (result,
                                   SYMBOL_ADDRESS (result, sym, false));
```

How imports get resolved

```
	result = _dl_lookup_symbol_x (strtab + sym->st_name, l, &sym, l->l_scope,
				      version, ELF_RTYPE_CLASS_PLT, flags, NULL);

	/* We are done with the global scope.  */
	if (!RTLD_SINGLE_THREAD_P)
	  THREAD_GSCOPE_RESET_FLAG ();

#ifdef RTLD_FINALIZE_FOREIGN_CALL
	RTLD_FINALIZE_FOREIGN_CALL;
#endif

	/* Currently result contains the base load address (or link map)
	   of the object that defines sym.  Now add in the symbol
	   offset.  */
	value = DL_FIXUP_MAKE_VALUE (result,
				     SYMBOL_ADDRESS (result, sym, false));
```

The Challenge

- Is there a way to resolve symbols without the need to rely on their names? Yes
- How? Creating a DT_HASH/DT_GNU_HASH resolver

```
Dynamic section at offset 0x27e68 contains 19 entries:
  Tag        Type                         Name/Value
 0x000000000000000e (SONAME)             Library soname: [ld-linux-x86-64.so.2]
 0x0000000000000004 (HASH)               0x1f0
 0x000000006ffffef5 (GNU_HASH)           0x2c8
 0x0000000000000005 (STRTAB)             0x6f0
 0x0000000000000006 (SYMTAB)             0x3c0
```

The approach (experimental WIP)

1. Hash all dynamic symbol strings for subject binary and remove .dynstr.

2. Save all hashes in a hash-table.

3. Make sure the hashes are placed in the hash table in the same order as the relocation indexes per dynamic symbol.

4. Inject a custom hash resolver into target binary along with the hashing table.

5. Inject constructor that replaces the value of GOT[2] for our resolver on runtime.

6. Profit.

Demo

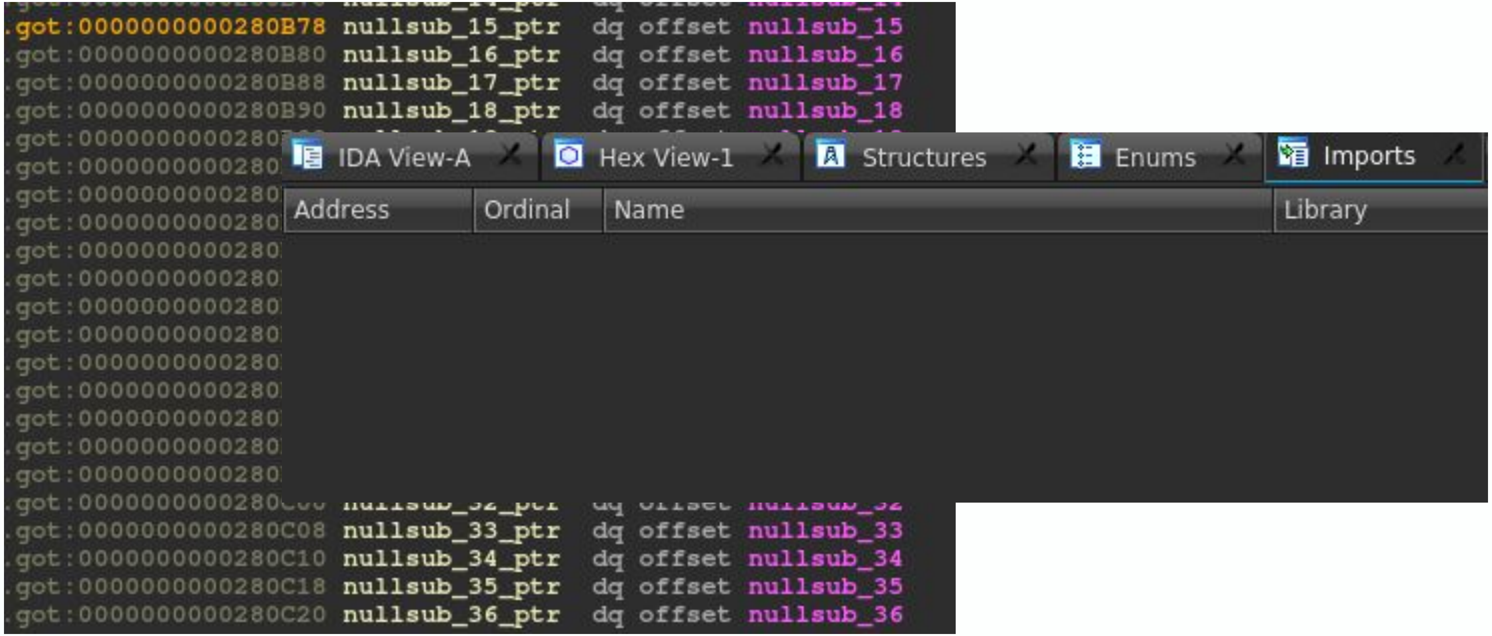
Outcome

```
Symbol table '.dynsym' contains 546 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
     4: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
     5: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
     6: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
     7: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
     8: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
     9: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    10: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    11: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    12: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    13: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    14: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    15: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    16: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
    17: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND
```

Outcome

```
.got:0000000000280B78 nullsub_15_ptr  dq offset nullsub_15
.got:0000000000280B80 nullsub_16_ptr  dq offset nullsub_16
.got:0000000000280B88 nullsub_17_ptr  dq offset nullsub_17
.got:0000000000280B90 nullsub_18_ptr  dq offset nullsub_18
.got:0000000000280B98 nullsub_19_ptr  dq offset nullsub_19
.got:0000000000280BA0 nullsub_20_ptr  dq offset nullsub_20
.got:0000000000280BA8 nullsub_21_ptr  dq offset nullsub_21
.got:0000000000280BB0 nullsub_22_ptr  dq offset nullsub_22
.got:0000000000280BB8 nullsub_23_ptr  dq offset nullsub_23
.got:0000000000280BC0 nullsub_24_ptr  dq offset nullsub_24
.got:0000000000280BC8 nullsub_25_ptr  dq offset nullsub_25
.got:0000000000280BD0 nullsub_26_ptr  dq offset nullsub_26
.got:0000000000280BD8 nullsub_27_ptr  dq offset nullsub_27
.got:0000000000280BE0 nullsub_28_ptr  dq offset nullsub_28
.got:0000000000280BE8 nullsub_29_ptr  dq offset nullsub_29
.got:0000000000280BF0 nullsub_30_ptr  dq offset nullsub_30
.got:0000000000280BF8 nullsub_31_ptr  dq offset nullsub_31
.got:0000000000280C00 nullsub_32_ptr  dq offset nullsub_32
.got:0000000000280C08 nullsub_33_ptr  dq offset nullsub_33
.got:0000000000280C10 nullsub_34_ptr  dq offset nullsub_34
.got:0000000000280C18 nullsub_35_ptr  dq offset nullsub_35
.got:0000000000280C20 nullsub_36_ptr  dq offset nullsub_36
```

Outcome


.got:0000000000280B78 nullsub_15_ptr dq offset nullsub_15
.got:0000000000280B80 nullsub_16_ptr dq offset nullsub_16
.got:0000000000280B88 nullsub_17_ptr dq offset nullsub_17
.got:0000000000280B90 nullsub_18_ptr dq offset nullsub_18
IDA View-A
Hex View-1
Structures
Enums
Imports
Address
Ordinal
Name
Library
.got:0000000000280C08 nullsub_33_ptr dq offset nullsub_33
.got:0000000000280C10 nullsub_34_ptr dq offset nullsub_34
.got:0000000000280C18 nullsub_35_ptr dq offset nullsub_35
.got:0000000000280C20 nullsub_36_ptr dq offset nullsub_36

Outcome

```
[0x0000d090]> ii
[Imports]
Num  Vaddr       Bind      Type  Name
   1 0x00000000  GLOBAL    FUNC
   2 0x00000000  GLOBAL    FUNC
   3 0x00000000  GLOBAL    FUNC
   4 0x00000000  GLOBAL    FUNC
   5 0x00000000  GLOBAL    FUNC
   6 0x00000000  GLOBAL    FUNC
   7 0x00000000  GLOBAL    FUNC
   8 0x00000000  GLOBAL    FUNC
   9 0x00000000  GLOBAL    FUNC
  10 0x00000000  GLOBAL    FUNC
  11 0x00000000  GLOBAL    FUNC
  12 0x00000000  GLOBAL    FUNC
  13 0x00000000  GLOBAL    FUNC
  14 0x00000000  GLOBAL    FUNC
  15 0x00000000  GLOBAL    FUNC
  16 0x00000000  GLOBAL    FUNC
  17 0x00000000  GLOBAL    FUNC
```

```
0x0000d09f  4883ec28         sub rsp, 0x28
0x0000d0a3  4c8b26           mov r12, qword [rsi]
0x0000d0a6  be2f000000       mov esi, 0x2f
0x0000d0ab  64488b042528.    mov rax, qword fs:[0x28]
0x0000d0b4  4889442418       mov qword [rsp + 0x18], rax
0x0000d0b9  31c0             xor eax, eax
0x0000d0bb  4c89e7           mov rdi, r12
0x0000d0be  e85ddbffff       call 0xac20
0x0000d0c3  488d5001         lea rdx, [rax + 1]
0x0000d0c7  4885c0           test rax, rax
0x0000d0ca  4c0f45e2         cmovne r12, rdx
0x0000d0ce  4531ff           xor r15d, r15d
0x0000d0d1  83fb01           cmp ebx, 1
```

Conclusion

- The ELF file format is very flexible.
- Parsing anomalies are easy.
- Code injection may start becoming more common as more complex implants arise.
- Relocations can be used to hide conventional control flow hijacking methodologies.
- ELF files can be easily abused for anti-analysis purposes in ways today are not being used.
- Expecting an increase in Linux malware complexity in coming years.

Questions?