SO That Looks Suspicious

Leveraging Process Memory & Kernel/Usermode Probes To Detect Shared Object Injection At Scale On Linux

Whoami

Daniel Jary (@JanielDary) – Security researcher

Previously:
- Senior security researcher @WithSecure/F-Secure.
- Security research & endpoint agent developer @UKGov.
- IR @Mandiant.
- Prev Speaker @BlackHatUSA, BlackHatAsia x2 …

Professional interests:
- OS internals.
- Reverse engineering.
- Tool & Sensor Dev.

- JanielDary

AGENDA

01 OVERVIEW
Shared Object Injection & the Linux threat landscape

02 ELF 101
ELF binary & process memory basics

03 DYNAMIC LINKER HIJACKING
Preloading and DT_NEEDED infections

04 REALTIME INJECTION & MONITORING
Reflective Shared Object Injection & __libc_dlopen_mode()

05 HIDE & SEEK
Hidden Shared Objects & Detection rules

06 KEY TAKEAWAYS

01

OVERVIEW

Shared Object Injection & The Linux Threat Landscape

Threat Landscape
(Trends)

> The level of innovation of Linux malware came close to that of Windows-based malware, highlighting just how prevalent Linux malware innovation has become, a trend that we are sure to see increasing in 2022 as well.

IBM X-Force Threat Intelligence Index 2022

> The importance of securing Linux® systems has risen in prominence as increasing amounts of malicious activity targeting Linux have appeared. Malware developers are increasingly developing Linux malware and creating Linux variants of existing malware families. These changes to the Linux threat landscape highlight the criticality of systems hardening and monitoring for malicious activity.

IBM X-Force Threat Intelligence Index 2024

cp<r> CHECK POINT RESEARCH

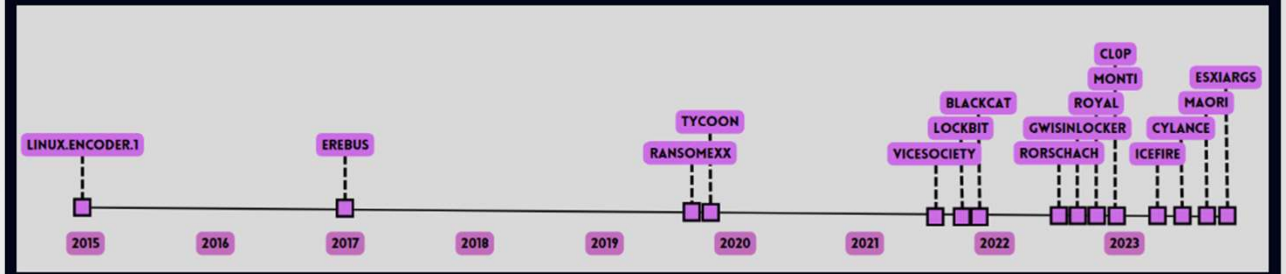


Figure 1 – Linux ransomware families.

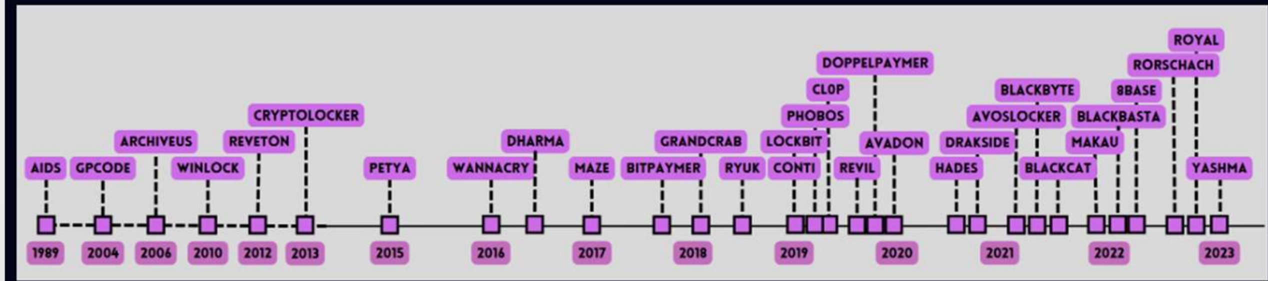
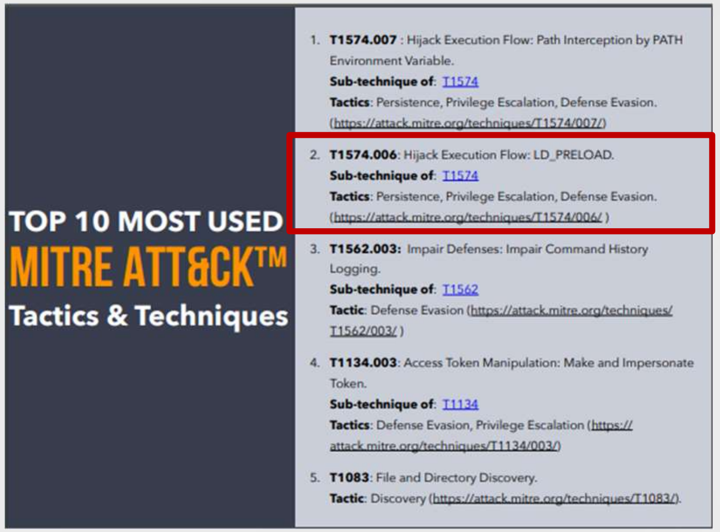


Figure 2 – Windows ransomware families.

https://research.checkpoint.com/2023/the-platform-matters-a-comparative-study-on-linux-and-windows-ransomware-attacks/

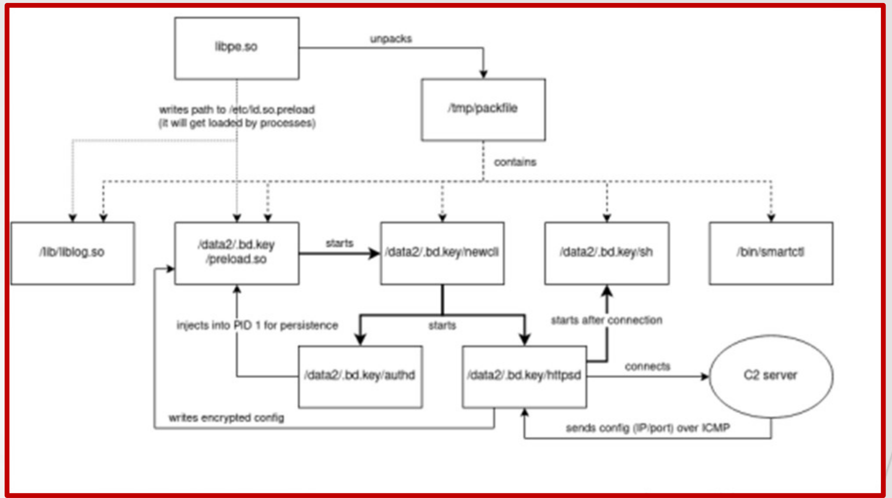
Threat Landscape



TOP 10 MOST USED
MITRE ATT&CK™
Tactics & Techniques

1. **T1574.007** : Hijack Execution Flow: Path Interception by PATH Environment Variable.
   Sub-technique of: T1574
   Tactics: Persistence, Privilege Escalation, Defense Evasion.
   (https://attack.mitre.org/techniques/T1574/007/)
2. **T1574.006**: Hijack Execution Flow: LD_PRELOAD.
   Sub-technique of: T1574
   Tactics: Persistence, Privilege Escalation, Defense Evasion.
   (https://attack.mitre.org/techniques/T1574/006/)
3. **T1562.003:** Impair Defenses: Impair Command History Logging.
   Sub-technique of: T1562
   Tactic: Defense Evasion (https://attack.mitre.org/techniques/T1562/003/)
4. **T1134.003**: Access Token Manipulation: Make and Impersonate Token.
   Sub-technique of: T1134
   Tactics: Defense Evasion, Privilege Escalation (https://attack.mitre.org/techniques/T1134/003/)
5. **T1083**: File and Directory Discovery.
   Tactic: Discovery (https://attack.mitre.org/techniques/T1083/).

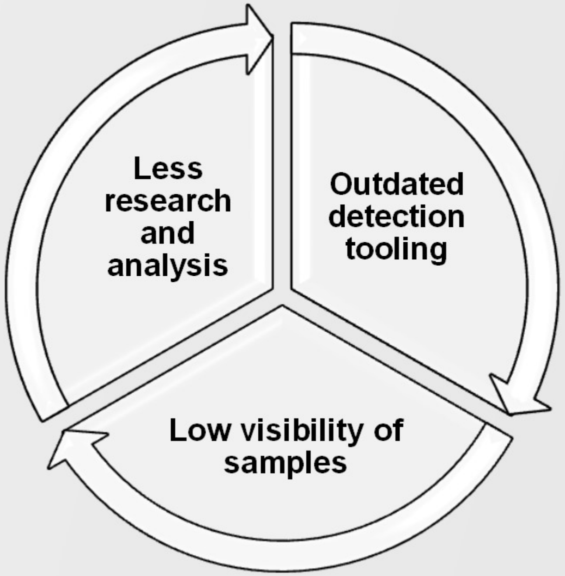
https://vfeed.io/wp-content/uploads/2021/02/Top-10-Most-Used-MITRE-ATTCK.pdf



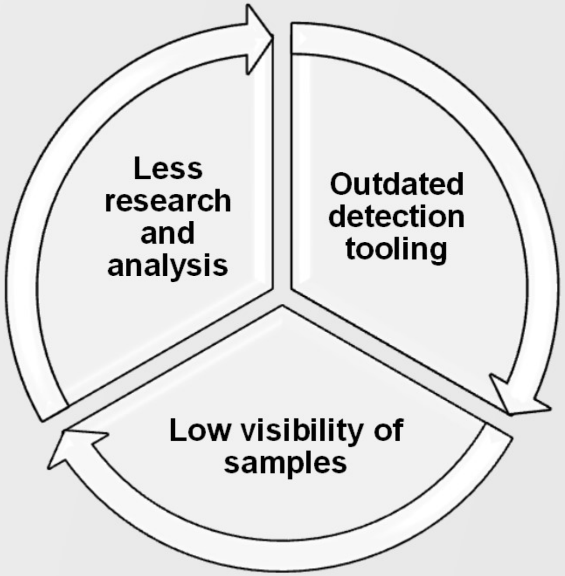
https://www.ncsc.nl/binaries/ncsc/documenten/publicaties/2024/februari/6/mivd-aivd-advisory-coathanger-tlp-clear/TLP-CLEAR+MIVD+AIVD+Advisory+COATHANGER.pdf

Threat Landscape

- Lack of detection maturity compared with Windows desktops.



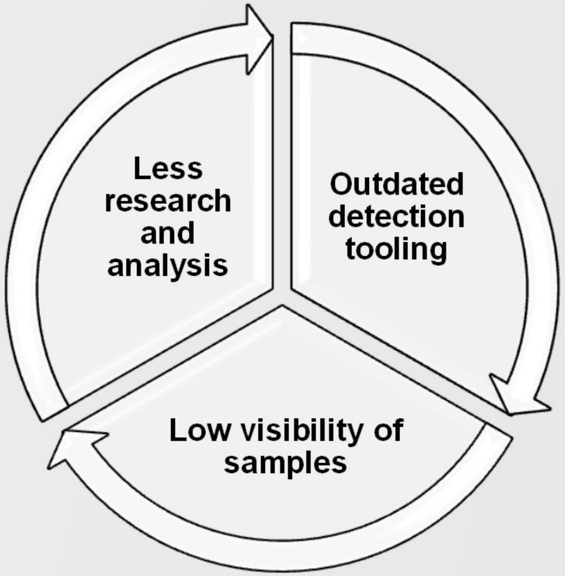
Threat Landscape



- Lack of detection maturity compared with Windows desktops.

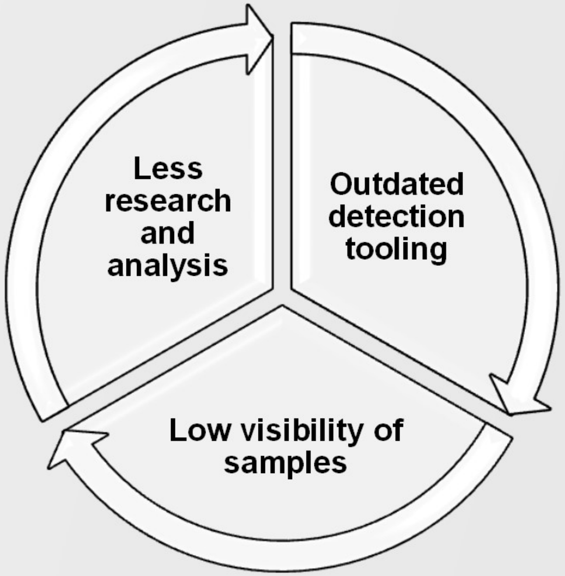
- Threat groups incorporating opensource code directly into their malware:
  - Winnti group (HiddenWasp / Azazel)
  - Rocke Group (Monero miner / Libprocess hider)

Threat Landscape



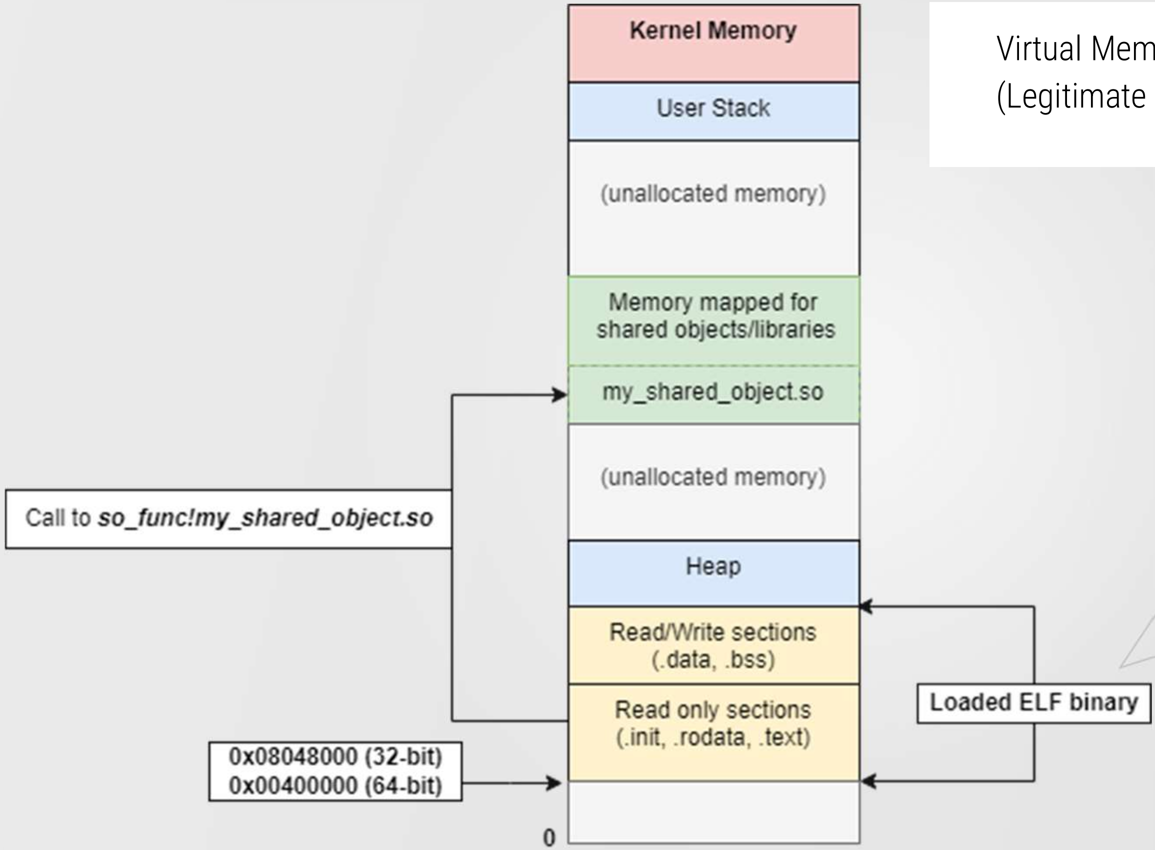
- Lack of detection maturity compared with Windows desktops.
- Threat groups incorporating opensource code directly into their malware:
  - Winnti group (HiddenWasp / Azazel)
  - Rocke Group (Monero miner / Libprocess hider)
- Post exploitation frameworks & state sponsored attackers using SO injection techniques:
  - Ninjasec/PupyRAT
  - COATHANGER (Chinese FortiGate RAT)

Threat Landscape



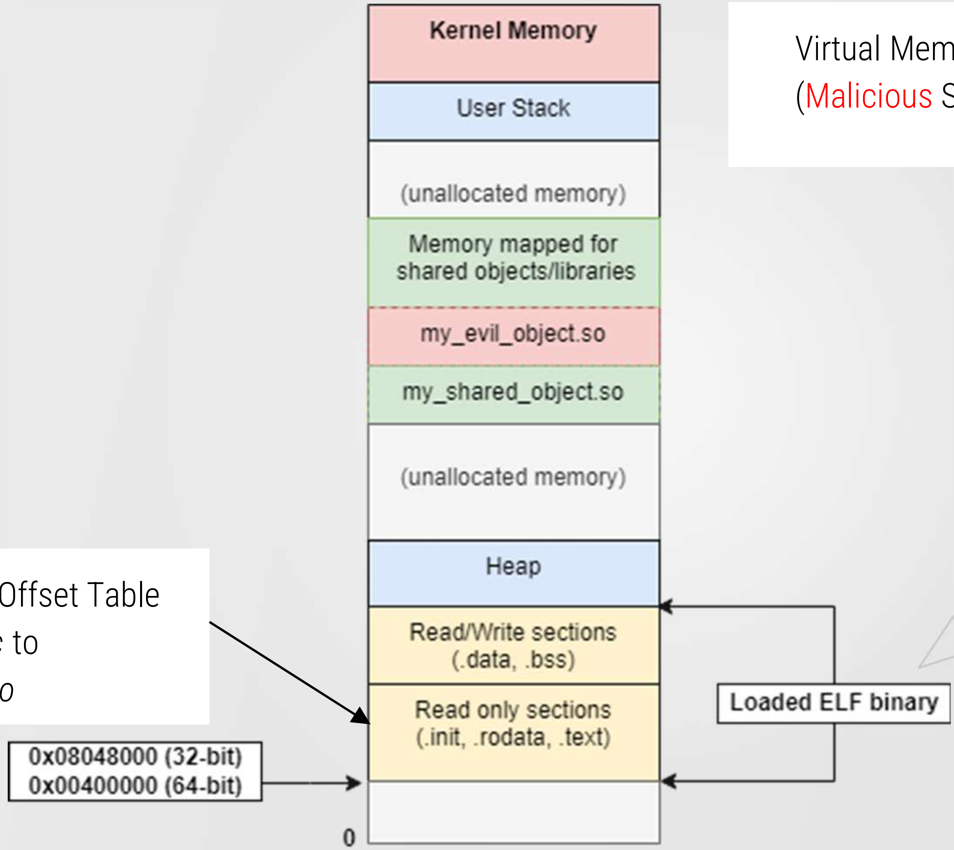
- Lack of detection maturity compared with Windows desktops.
- Threat groups incorporating opensource code directly into their malware:
  - Winnti group (HiddenWasp / Azazel)
  - Rocke Group (Monero miner / Libprocess hider)
- Post exploitation frameworks & state sponsored attackers using SO injection techniques:
  - Ninjasec/PupyRAT
- Open-source tooling & conferences presentations demonstrating Usermode memory injection techniques

What Is Shared Object (SO) Injection?



Virtual Memory Layout
(Legitimate SO usage)

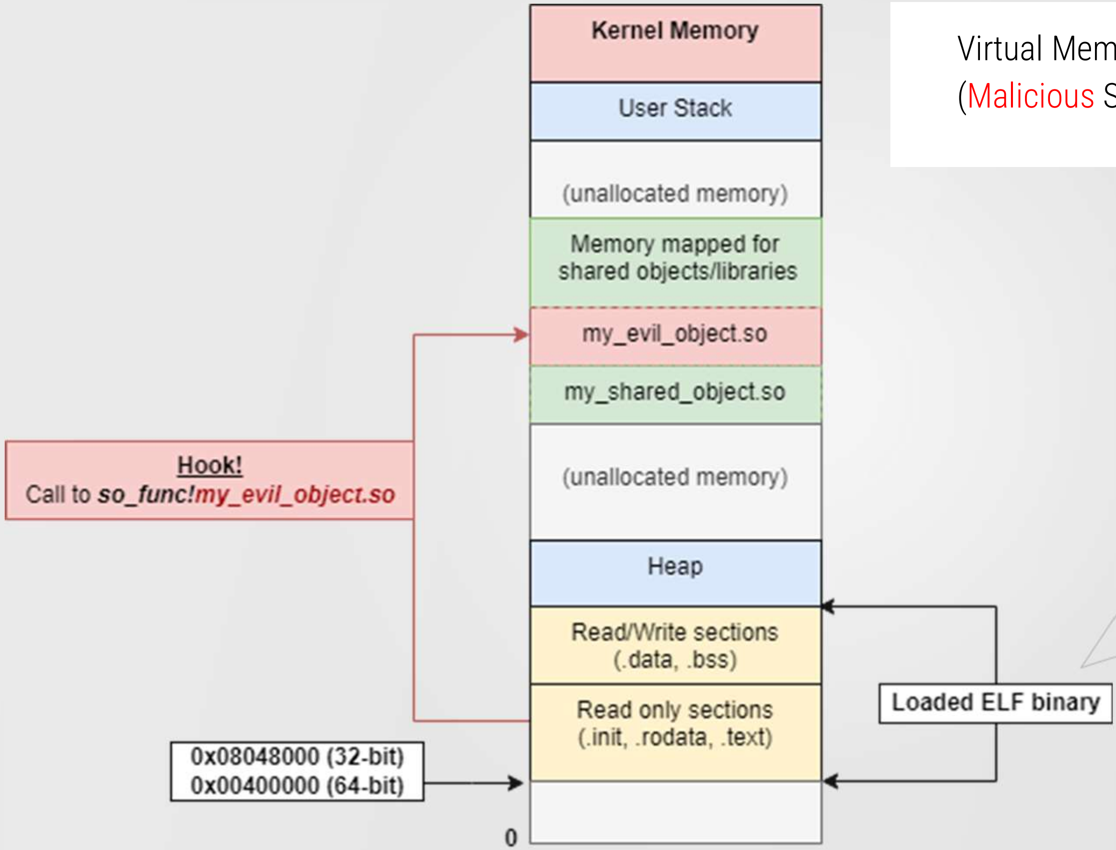
What Is Shared Object (SO) Injection?



Virtual Memory Layout
(Malicious SO injection)

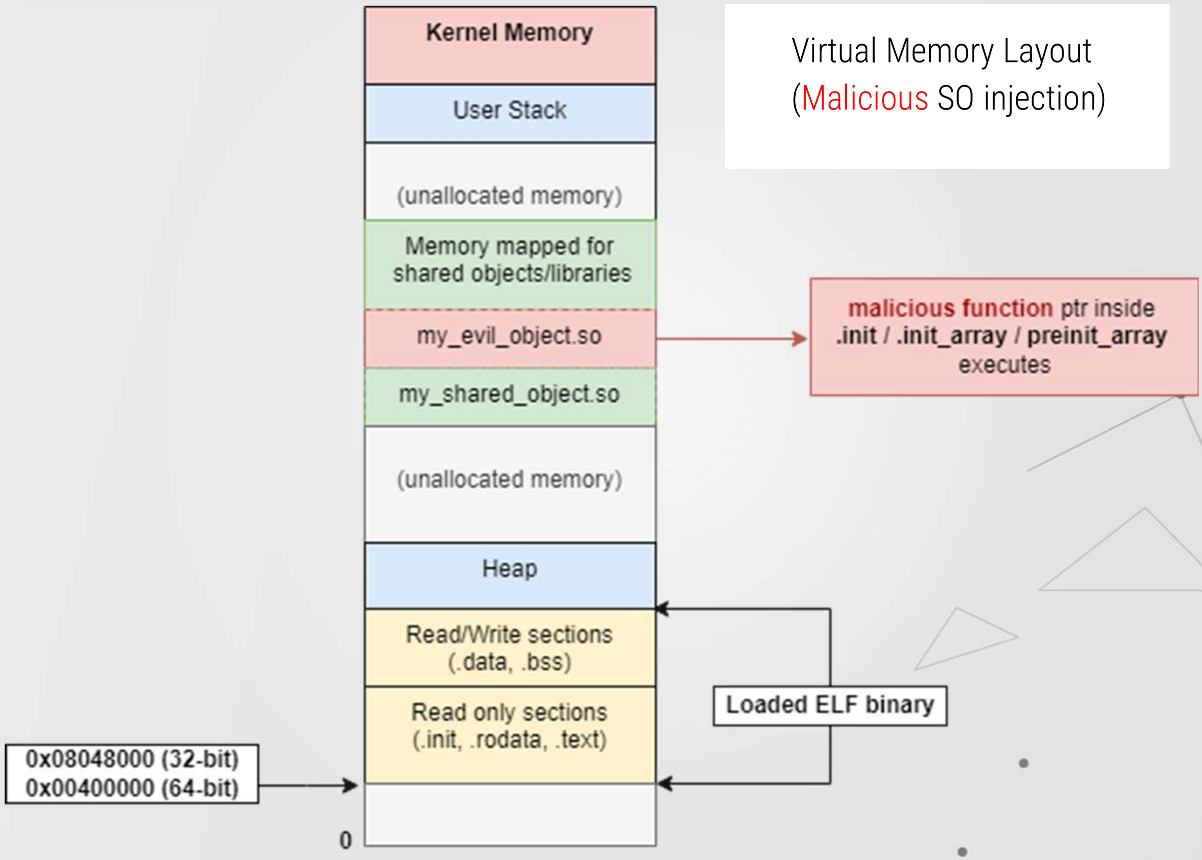
Patched Global Offset Table (.got) of *so_func* to *my_evil_object.so*

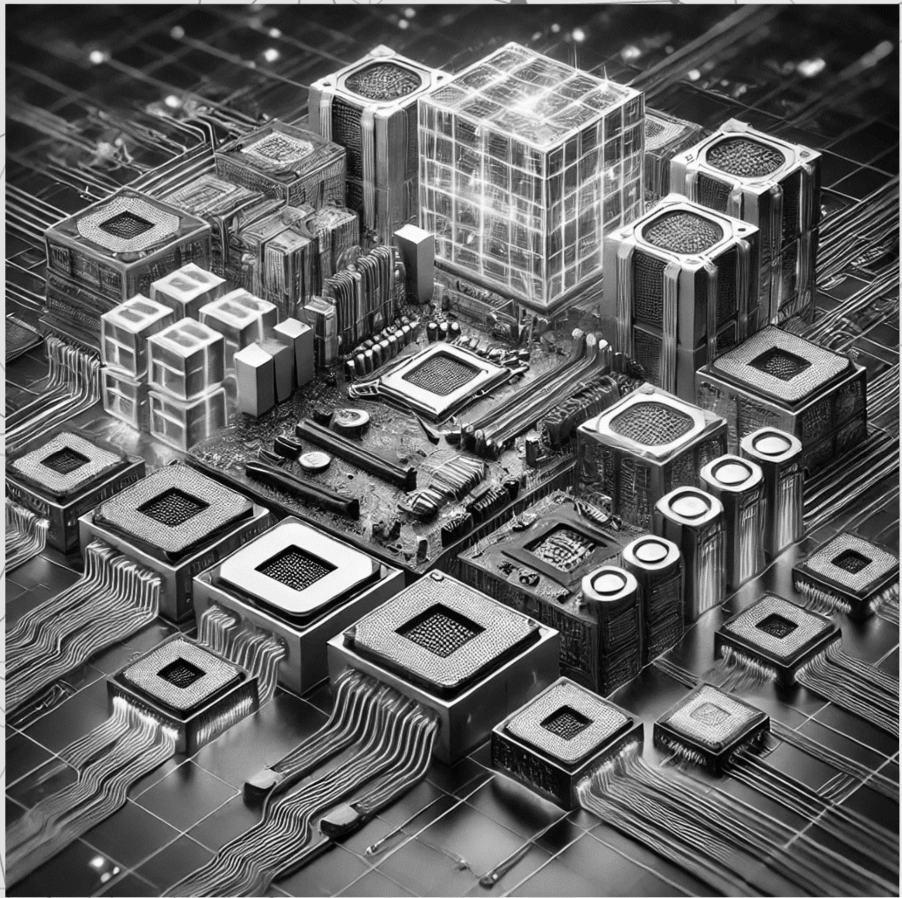
What Is Shared Object (SO) Injection?



Virtual Memory Layout
(Malicious SO injection)

What Is Shared Object (SO) Injection?





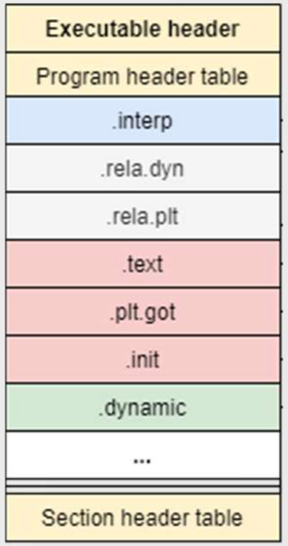
02

ELF 101

ELF Binary & Process Memory Basics

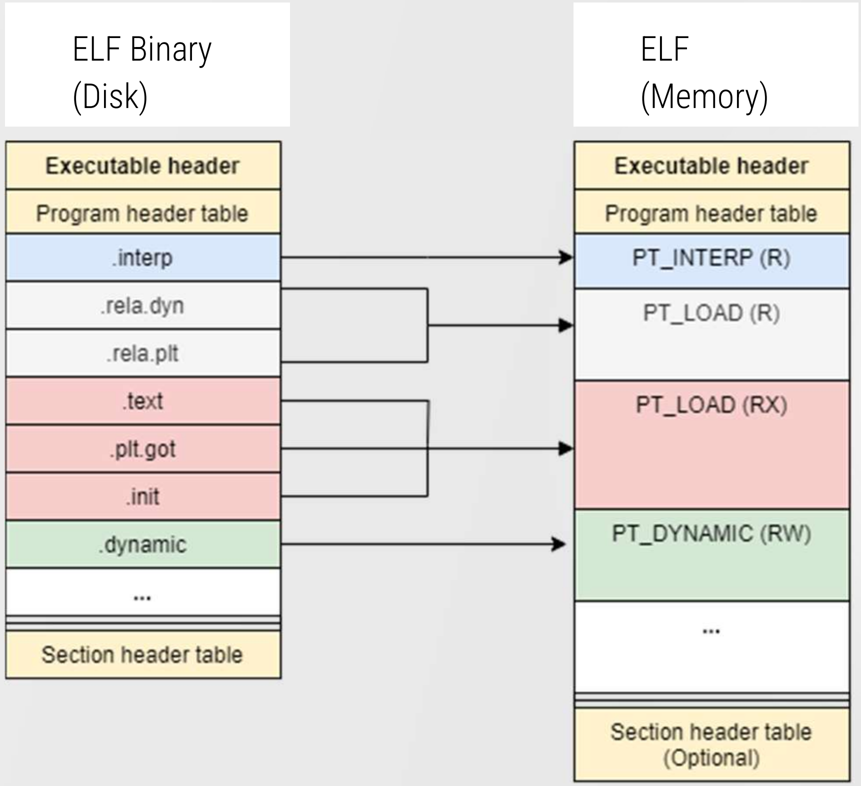
Binary Vs Memory Images

ELF Binary
(Disk)



- ELF Sections contain large degree of forensic value.
  - *Symbol Table, Relocation table, Constructors/Destructors, Program Data, Dynamic linking information*

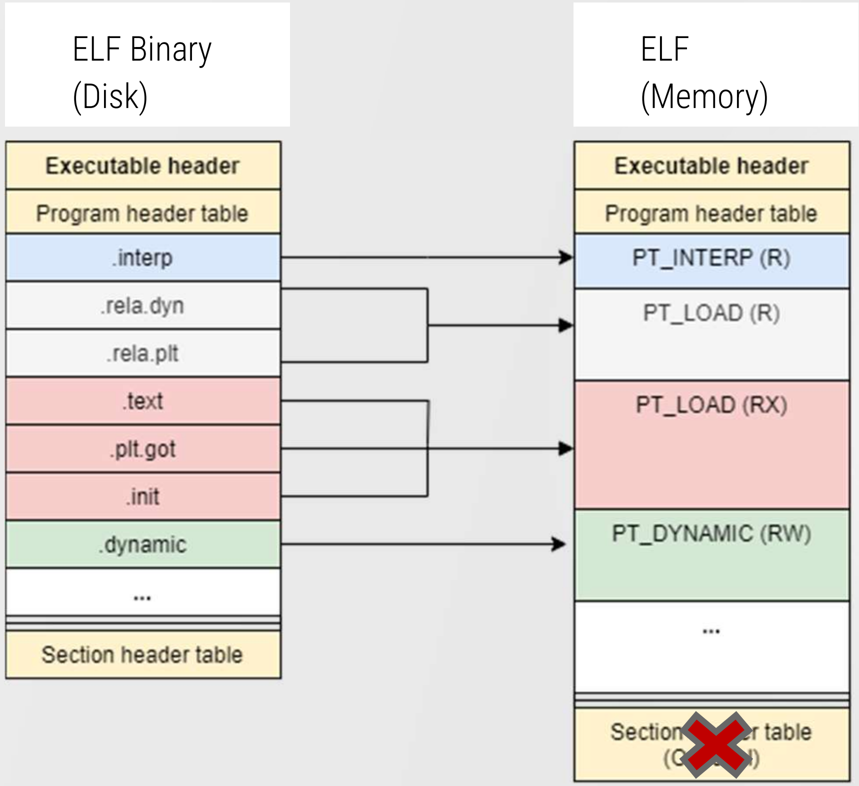
Binary Vs Memory Images



- ELF Sections contain large degree of forensic value.
  - *Symbol Table, Relocation table, Constructors/Destructors, Program Data, Dynamic linking information*

- Segments loose Section granularity!

Binary Vs Memory Images

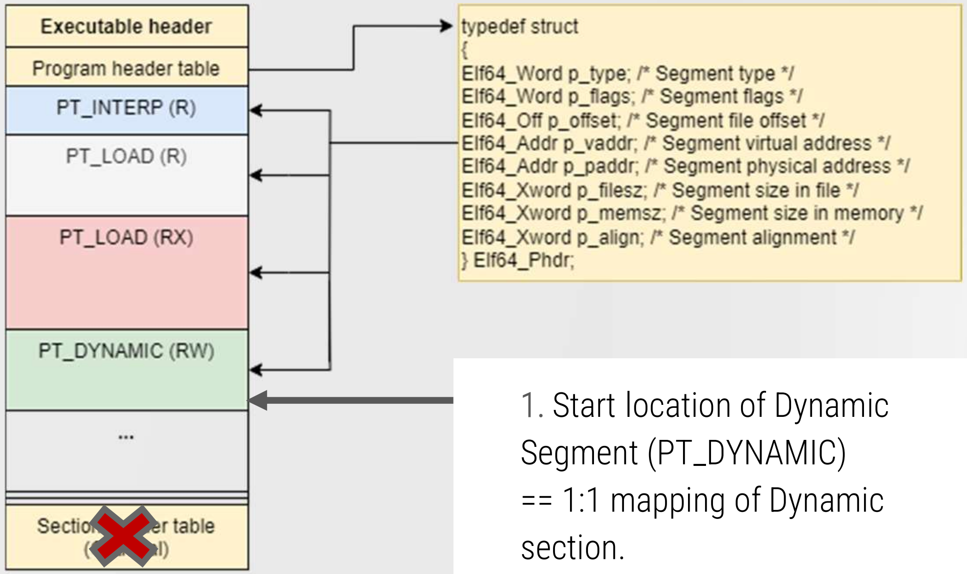


- ELF Sections contain large degree of forensic value.
  - *Symbol Table, Relocation table, Constructors/Destructors, Program Data, Dynamic linking information*

- Segments loose Section granularity!

- Section header table is Optional in mapped memory image. Not suitable for use in forensic tooling.

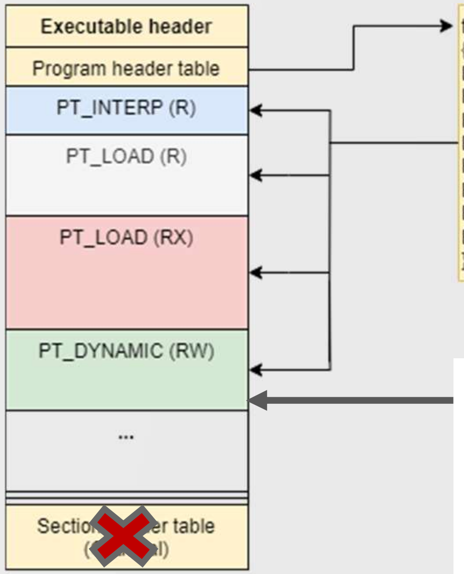
Rebuilding Elf Sections From Memory
(Using The Dynamic Section)



1. Start location of Dynamic Segment (PT_DYNAMIC) == 1:1 mapping of Dynamic section.

Rebuilding Elf Sections From Memory

(Using The Dynamic Section)




```
typedef struct
{
Elf64_Word p_type; /* Segment type */
Elf64_Word p_flags; /* Segment flags */
Elf64_Off p_offset; /* Segment file offset */
Elf64_Addr p_vaddr; /* Segment virtual address */
Elf64_Addr p_paddr; /* Segment physical address */
Elf64_Xword p_filesz; /* Segment size in file */
Elf64_Xword p_memsz; /* Segment size in memory */
Elf64_Xword p_align; /* Segment alignment */
} Elf64_Phdr;
```

1. Start location of Dynamic Segment (PT_DYNAMIC) == 1:1 mapping of Dynamic section.

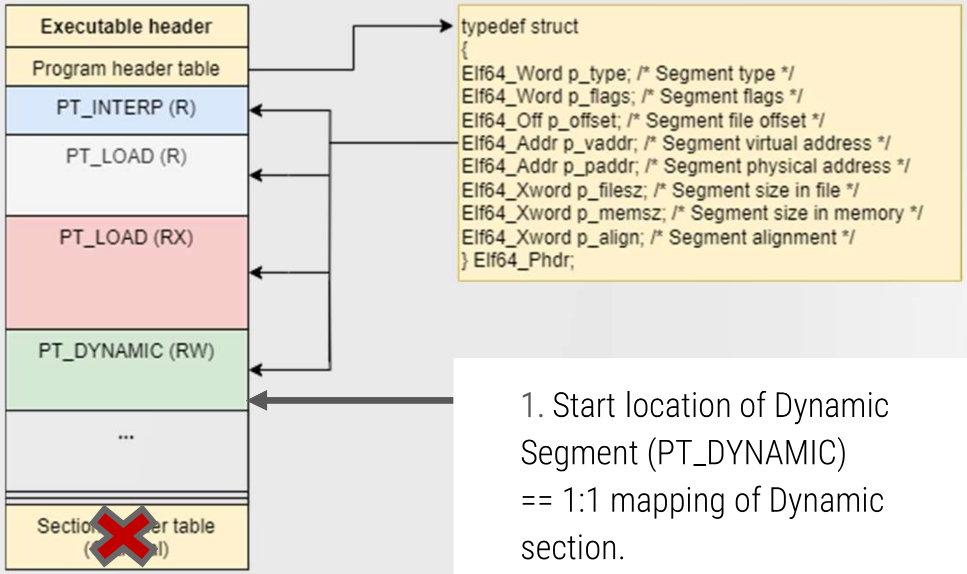
2. Contains pointers to ELF sections needed by the Dynamic Linker

Readelf output of Dynamic section

```
root@test-VirtualBox:~# readelf -d /bin/bash

Dynamic section at offset 0x106630 contains 29 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libtinfo.so.5]
 0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x2be18
 0x000000000000000d (FINI)               0xcf7c4
 0x0000000000000019 (INIT_ARRAY)         0x303d90
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x303d98
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x298
 0x0000000000000005 (STRTAB)             0x12a28
 0x0000000000000006 (SYMTAB)             0x4c40
 0x000000000000000a (STRSZ)              37574 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x306840
 0x0000000000000002 (PLTRELSZ)           5136 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x2aa08
 0x0000000000000007 (RELA)               0x1d040
 0x0000000000000008 (RELASZ)             55752 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000000000001e (FLAGS)              BIND_NOW
 0x000000006ffffffb (FLAGS_1)            Flags: NOW PIE
 0x000000006ffffffe (VERNEED)            0x1cf70
 0x000000006fffffff (VERNEEDNUM)         3
 0x000000006ffffff0 (VERSYM)             0x1bcee
 0x000000006ffffff9 (RELACOUNT)          2309
 0x0000000000000000 (NULL)               0x0
```

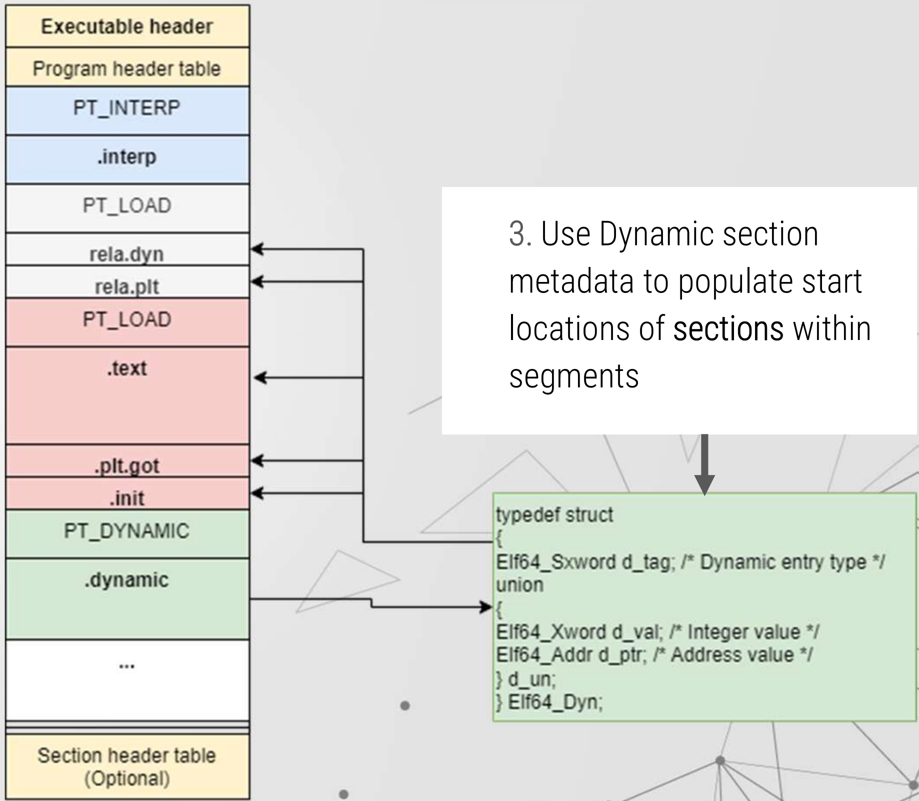
Rebuilding Elf Sections From Memory

(Using The Dynamic Section)



1. Start location of Dynamic Segment (PT_DYNAMIC) == 1:1 mapping of Dynamic section.

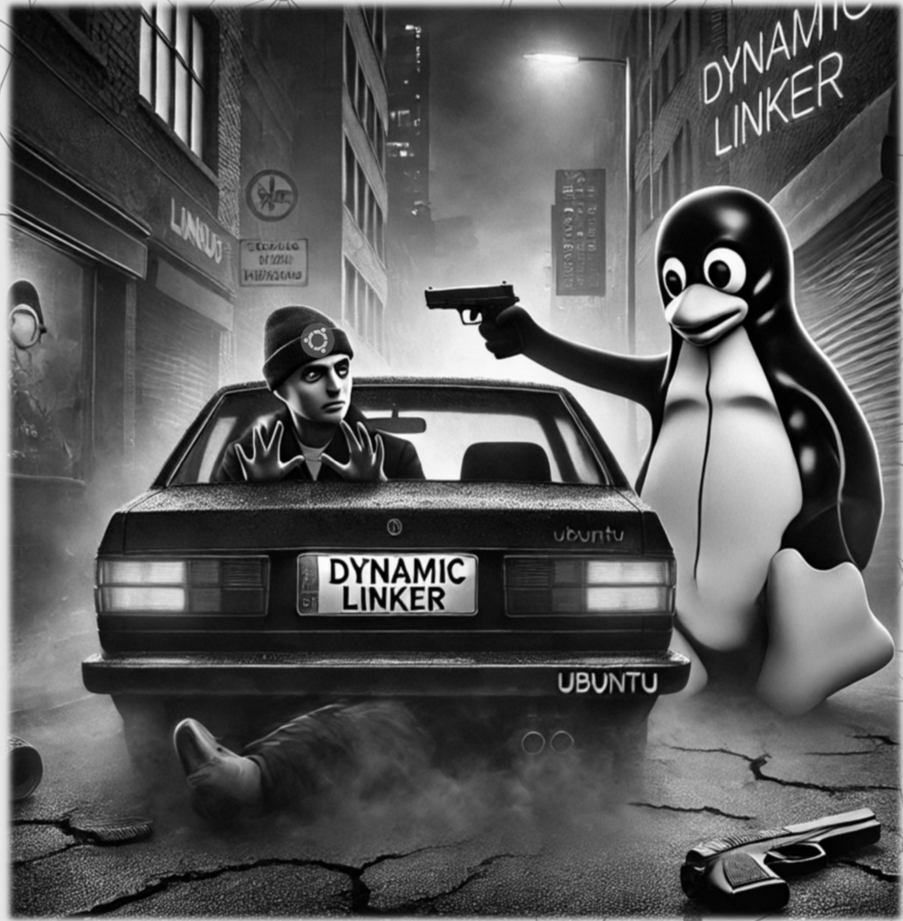
2. Contains pointers to ELF sections needed by the Dynamic Linker



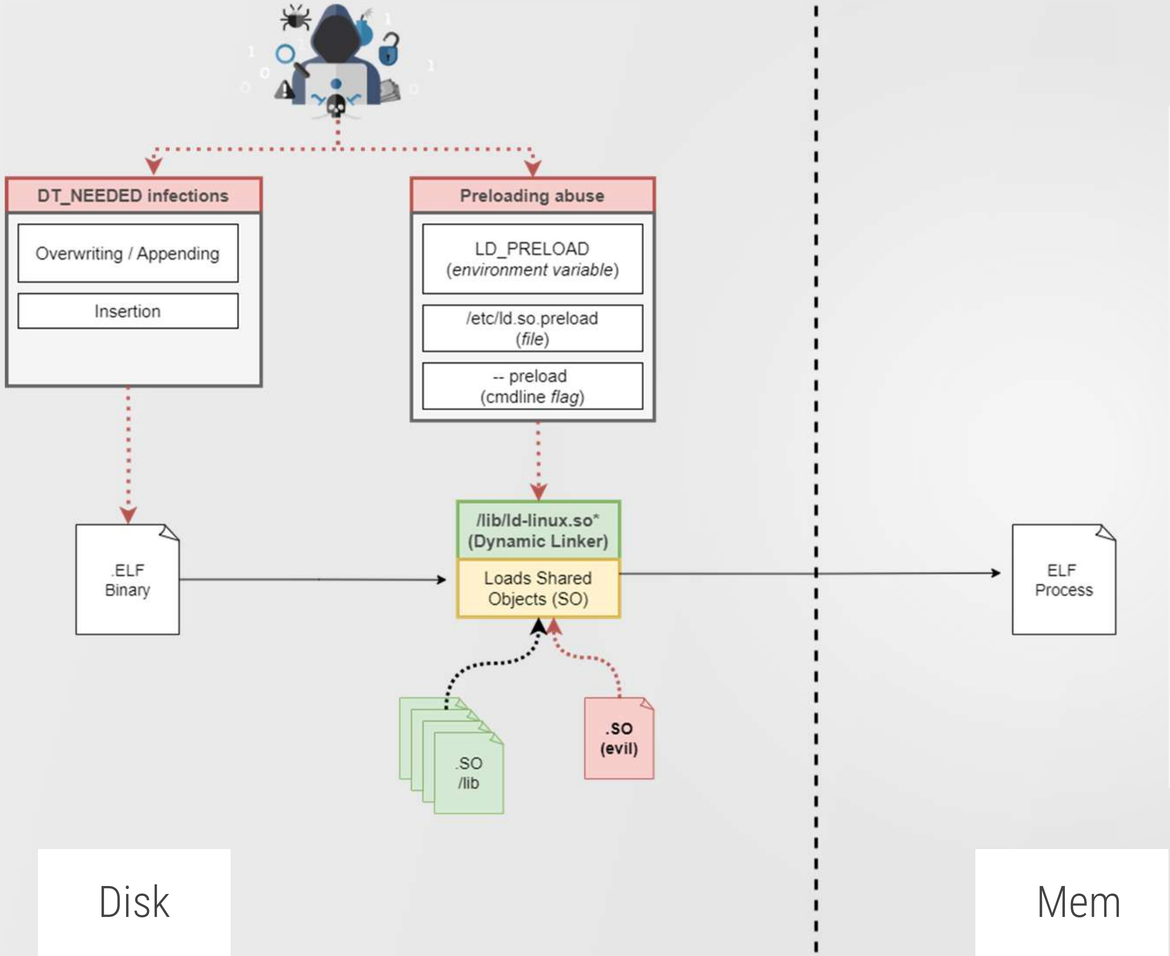
3. Use Dynamic section metadata to populate start locations of sections within segments

03
DYNAMIC LINKER HIJACKING

Preloading & DT_NEEDED Infections



Abusing The Dynamic Linker To Load Malicious SOs



- Two types:
  - DT_NEEDED infections
  - Preloading
- Setup before process execution
- Common among Usermode rootkits & malware
- Current detection strategies & limitations.
- Generating actionable telemetry.

DT_NEEDED Entries & The Dynamic Linker

- Dynamic Segment 1:1 mapping with the Dynamic section
- Present in all dynamic linked binaries
- Each entry Dynamic section is required by the dynamic linker to load a binary into memory:
  - DT_NEEDED - Dependencies to load.
  - DT_SYMTAB – Dynamic symbol table.
  - DT_FLAGS – How to process shared objects.
  - DT_PLTGOT – Pointer within Global Offset Table (GOT).
  - DT_RPATH/DT_RUNPATH (optional) – Pointer to a directory the dynamic linker should look to load libraries from.

```
root@test-VirtualBox:~# readelf -d /bin/bash

Dynamic section at offset 0x106630 contains 29 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libtinfo.so.5]
 0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x2be18
 0x000000000000000d (FINI)               0xcf7c4
 0x0000000000000019 (INIT_ARRAY)         0x303d90
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x303d98
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x298
 0x0000000000000005 (STRTAB)             0x12a28
 0x0000000000000006 (SYMTAB)             0x4c40
 0x000000000000000a (STRSZ)              37574 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x306840
 0x0000000000000002 (PLTRELSZ)           5136 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x2aa08
 0x0000000000000007 (RELA)               0x1d040
 0x0000000000000008 (RELASZ)             55752 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000000000001e (FLAGS)              BIND_NOW
 0x000000006ffffffb (FLAGS_1)            Flags: NOW PIE
 0x000000006ffffffe (VERNEED)            0x1cf70
 0x000000006fffffff (VERNEEDNUM)         3
 0x000000006ffffff0 (VERSYM)             0x1bcee
 0x000000006ffffff9 (RELACOUNT)          2309
 0x0000000000000000 (NULL)               0x0
```

DT_NEEDED Entries & The Dynamic Linker

- Dynamic Segment 1:1 mapping with the Dynamic section

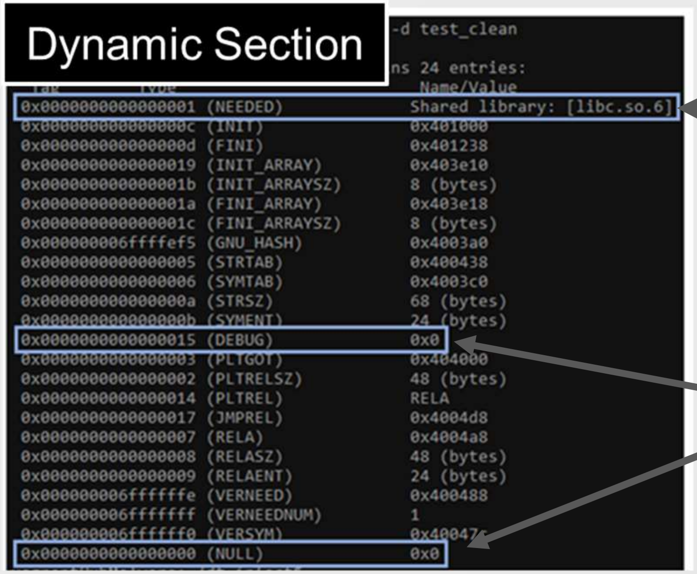
- Present in all dynamic linked binaries

- Each entry Dynamic section is required by the dynamic linker to load a binary into memory:
  - DT_NEEDED - Dependencies to load.
  - DT_SYMTAB – Dynamic symbol table.
  - DT_FLAGS – How to process shared objects.
  - DT_PLTGOT – Pointer within Global Offset Table (GOT).
  - DT_RPATH/DT_RUNPATH (optional) – Pointer to a directory the dynamic linker should look to load libraries from.

```
root@test-VirtualBox:~# readelf -d /bin/bash

Dynamic section at offset 0x106630 contains 29 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libtinfo.so.5]
 0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x2be18
 0x000000000000000d (FINI)               0xcf7c4
 0x0000000000000019 (INIT_ARRAY)         0x303d90
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x303d98
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x298
 0x0000000000000005 (STRTAB)             0x12a28
 0x0000000000000006 (SYMTAB)             0x4c40
 0x000000000000000a (STRSZ)              37574 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x306840
 0x0000000000000002 (PLTRELSZ)           5136 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x2aa08
 0x0000000000000007 (RELA)               0x1d040
 0x0000000000000008 (RELASZ)             55752 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000000000001e (FLAGS)              BIND_NOW
 0x000000006ffffffb (FLAGS_1)            Flags: NOW PIE
 0x000000006ffffffe (VERNEED)            0x1cf70
 0x000000006fffffff (VERNEEDNUM)         3
 0x000000006ffffff0 (VERSYM)             0x1bcee
 0x000000006ffffff9 (RELACOUNT)          2309
 0x0000000000000000 (NULL)               0x0
```

DT_NEEDED Infections

(Overwrites)

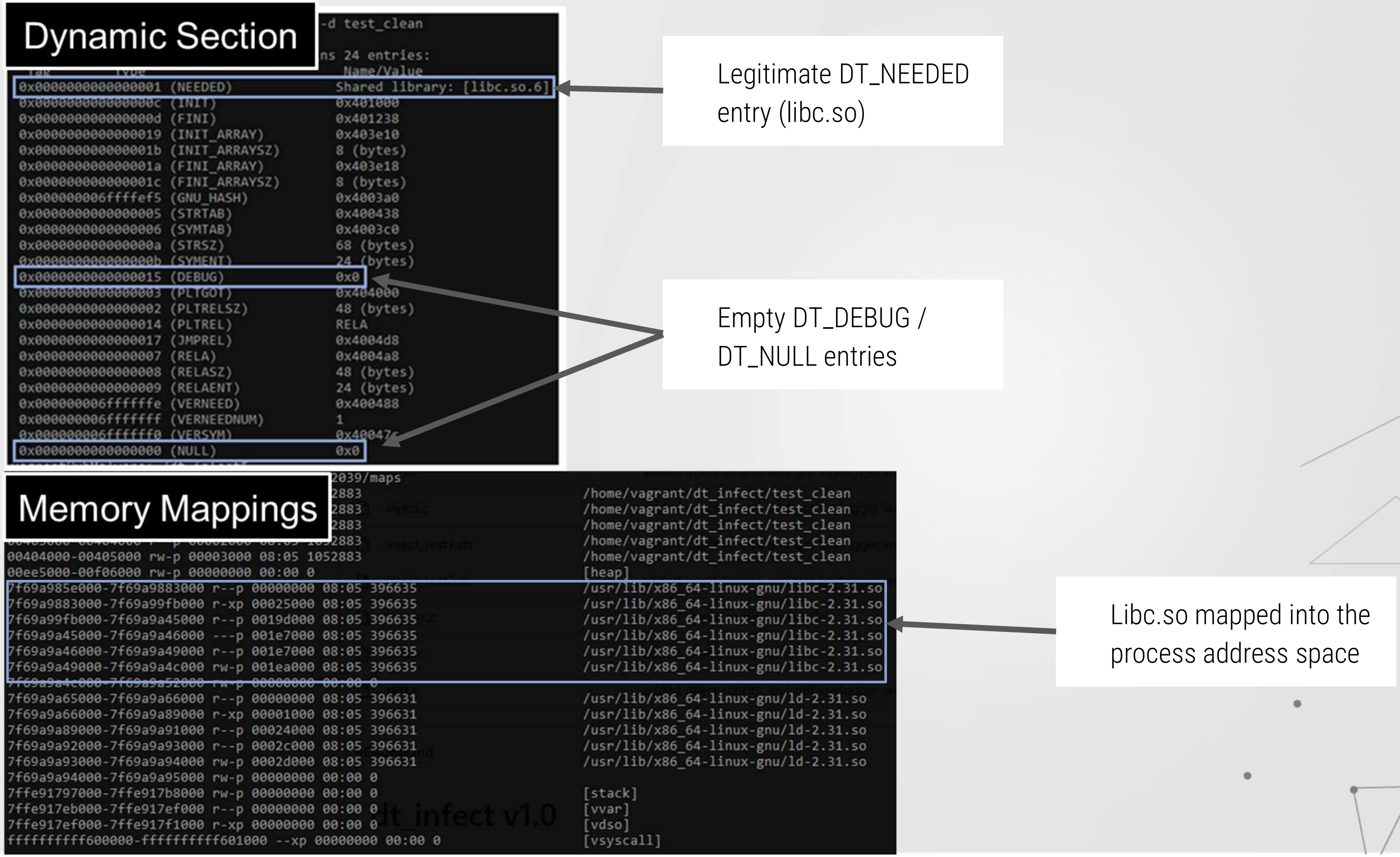


Legitimate DT_NEEDED entry (libc.so)

Empty DT_DEBUG / DT_NULL entries

DT_NEEDED Infections

(Overwrites)



DT_NEEDED Infections

(Overwrites)

Dynamic Section

```
-d test_clean

ns 24 entries:
                                        Name/Value
0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
0x000000000000000c (INIT)               0x401000
0x000000000000000d (FINI)               0x401238
0x0000000000000019 (INIT_ARRAY)         0x403e10
0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
0x000000000000001a (FINI_ARRAY)         0x403e18
0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
0x000000006ffffef5 (GNU_HASH)           0x4003a0
0x0000000000000005 (STRTAB)             0x400438
0x0000000000000006 (SYMTAB)             0x4003c0
0x000000000000000a (STRSZ)              68 (bytes)
0x000000000000000b (SYMENT)             24 (bytes)
0x0000000000000015 (DEBUG)              0x0
0x0000000000000003 (PLTGOT)             0x404000
0x0000000000000002 (PLTRELSZ)           48 (bytes)
0x0000000000000014 (PLTREL)             RELA
0x0000000000000017 (JMPREL)             0x4004d8
0x0000000000000007 (RELA)               0x4004a8
0x0000000000000008 (RELASZ)             48 (bytes)
0x0000000000000009 (RELAENT)            24 (bytes)
0x000000006ffffffe (VERNEED)            0x400488
0x000000006fffffff (VERNEEDNUM)         1
0x000000006ffffff0 (VERSYM)             0x40047c
0x0000000000000000 (NULL)               0x0
```

Overwrite DT_DEBUG with DT_NEEDED (libevil.so)

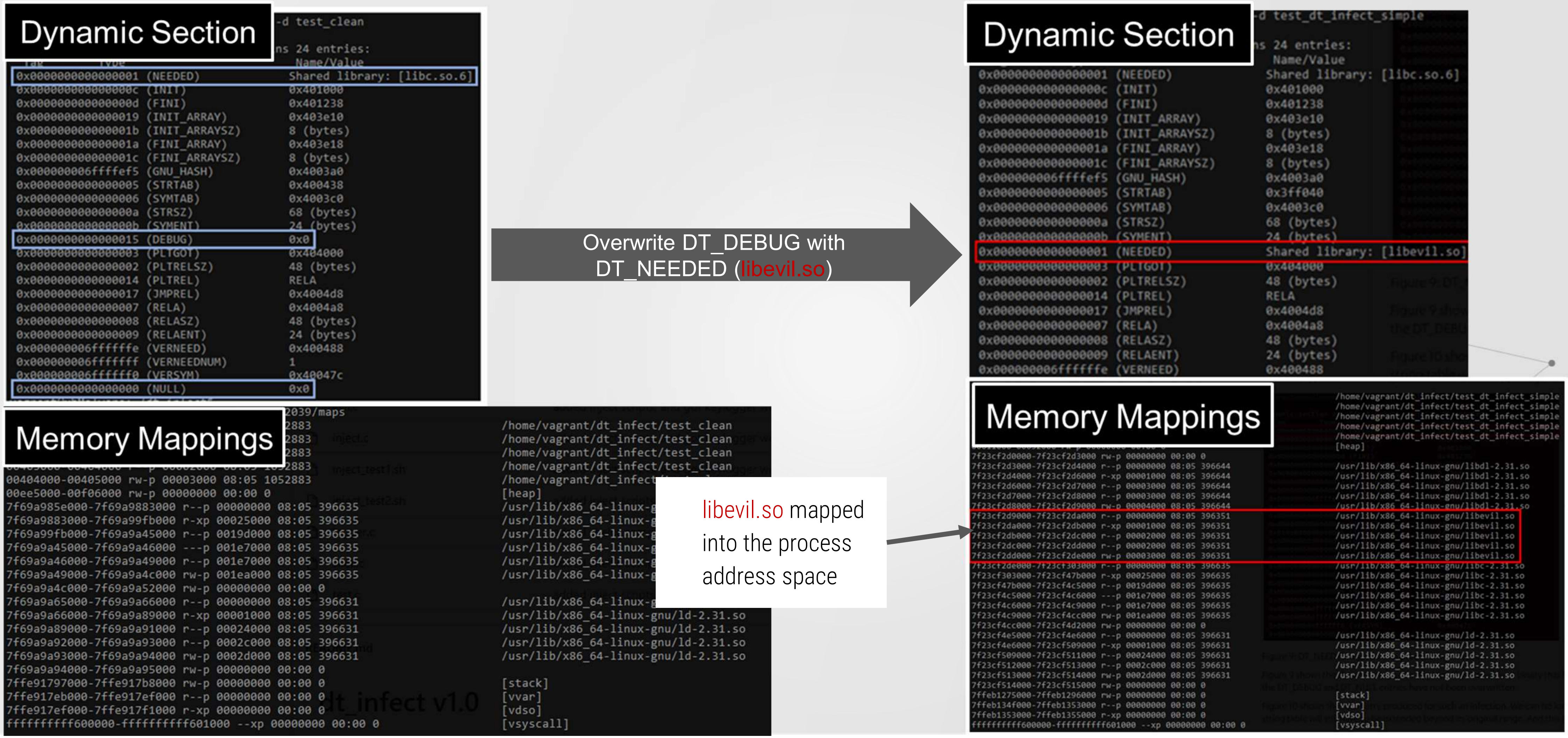
Dynamic Section

```
-d test_dt_infect_simple

ns 24 entries:
                                        Name/Value
0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
0x000000000000000c (INIT)               0x401000
0x000000000000000d (FINI)               0x401238
0x0000000000000019 (INIT_ARRAY)         0x403e10
0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
0x000000000000001a (FINI_ARRAY)         0x403e18
0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
0x000000006ffffef5 (GNU_HASH)           0x4003a0
0x0000000000000005 (STRTAB)             0x3ff040
0x0000000000000006 (SYMTAB)             0x4003c0
0x000000000000000a (STRSZ)              68 (bytes)
0x000000000000000b (SYMENT)             24 (bytes)
0x0000000000000001 (NEEDED)             Shared library: [libevil.so]
0x0000000000000003 (PLTGOT)             0x404000
0x0000000000000002 (PLTRELSZ)           48 (bytes)
0x0000000000000014 (PLTREL)             RELA
0x0000000000000017 (JMPREL)             0x4004d8
0x0000000000000007 (RELA)               0x4004a8
0x0000000000000008 (RELASZ)             48 (bytes)
0x0000000000000009 (RELAENT)            24 (bytes)
0x000000006ffffffe (VERNEED)            0x400488
```

Memory Mappings

```
2039/maps
2883                                                          /home/vagrant/dt_infect/test_clean
2883                                                          /home/vagrant/dt_infect/test_clean
2883                                                          /home/vagrant/dt_infect/test_clean
2883                                                          /home/vagrant/dt_infect/test_clean
00404000-00405000 rw-p 00003000 08:05 1052883                 /home/vagrant/dt_infect/test_clean
00ee5000-00f06000 rw-p 00000000 00:00 0                       [heap]
7f69a985e000-7f69a9883000 r--p 00000000 08:05 396635          /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f69a9883000-7f69a99fb000 r-xp 00025000 08:05 396635          /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f69a99fb000-7f69a9a45000 r--p 0019d000 08:05 396635          /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f69a9a45000-7f69a9a46000 ---p 001e7000 08:05 396635          /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f69a9a46000-7f69a9a49000 r--p 001e7000 08:05 396635          /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f69a9a49000-7f69a9a4c000 rw-p 001ea000 08:05 396635          /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f69a9a4c000-7f69a9a52000 rw-p 00000000 00:00 0
7f69a9a65000-7f69a9a66000 r--p 00000000 08:05 396631          /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f69a9a66000-7f69a9a89000 r-xp 00001000 08:05 396631          /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f69a9a89000-7f69a9a91000 r--p 00024000 08:05 396631          /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f69a9a92000-7f69a9a93000 r--p 0002c000 08:05 396631          /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f69a9a93000-7f69a9a94000 rw-p 0002d000 08:05 396631          /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f69a9a94000-7f69a9a95000 rw-p 00000000 00:00 0
7ffe91797000-7ffe917b8000 rw-p 00000000 00:00 0               [stack]
7ffe917eb000-7ffe917ef000 r--p 00000000 00:00 0               [vvar]
7ffe917ef000-7ffe917f1000 r-xp 00000000 00:00 0               [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0       [vsyscall]
```

DT_NEEDED Infections
(Overwrites)



DT_NEEDED Infections

(Insertions)

Dynamic Section

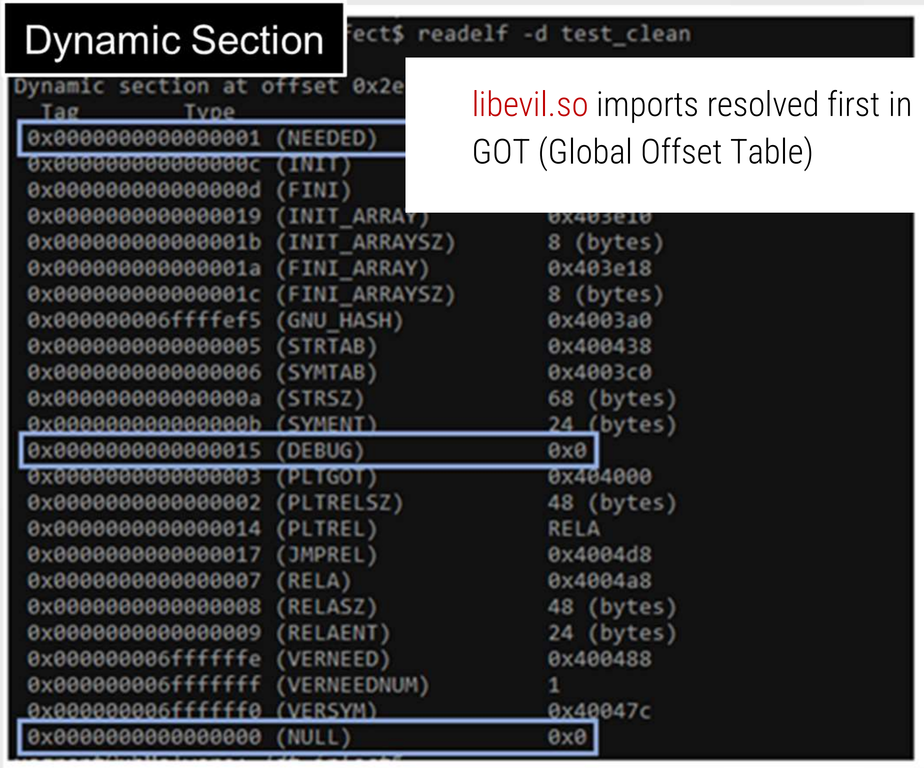
```
fect$ readelf -d test_clean

Dynamic section at offset 0x2e20 contains 24 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x401000
 0x000000000000000d (FINI)               0x401238
 0x0000000000000019 (INIT_ARRAY)         0x403e10
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x403e18
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x4003a0
 0x0000000000000005 (STRTAB)             0x400438
 0x0000000000000006 (SYMTAB)             0x4003c0
 0x000000000000000a (STRSZ)              68 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x404000
 0x0000000000000002 (PLTRELSZ)           48 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x4004d8
 0x0000000000000007 (RELA)               0x4004a8
 0x0000000000000008 (RELASZ)             48 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000006ffffffe (VERNEED)            0x400488
 0x000000006fffffff (VERNEEDNUM)         1
 0x000000006ffffff0 (VERSYM)             0x40047c
 0x0000000000000000 (NULL)               0x0
```

Dynamic Section

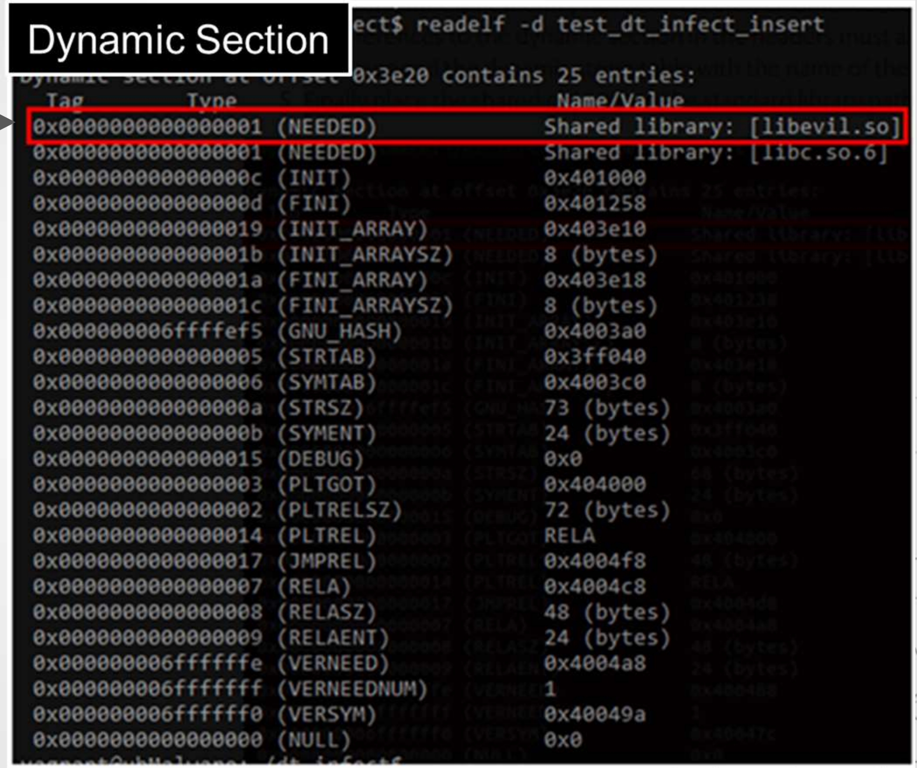
```
ect$ readelf -d test_dt_infect_insert

Dynamic section at offset 0x3e20 contains 25 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libevil.so]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x401000
 0x000000000000000d (FINI)               0x401258
 0x0000000000000019 (INIT_ARRAY)         0x403e10
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x403e18
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x4003a0
 0x0000000000000005 (STRTAB)             0x3ff040
 0x0000000000000006 (SYMTAB)             0x4003c0
 0x000000000000000a (STRSZ)              73 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000003 (PLTGOT)             0x404000
 0x0000000000000002 (PLTRELSZ)           72 (bytes)
 0x0000000000000014 (PLTREL)             RELA
 0x0000000000000017 (JMPREL)             0x4004f8
 0x0000000000000007 (RELA)               0x4004c8
 0x0000000000000008 (RELASZ)             48 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x000000006ffffffe (VERNEED)            0x4004a8
 0x000000006fffffff (VERNEEDNUM)         1
 0x000000006ffffff0 (VERSYM)             0x40049a
 0x0000000000000000 (NULL)               0x0
```

DT_NEEDED Infections

(Insertions)



libevil.so imports resolved first in GOT (Global Offset Table)



DT_NEEDED Infections

(Detection)

OVERWRITES

- Order of DT_NEEDED entries in dynamic section
- Dynamic string table extension
- Missing DT_DEBUG/DT_NULL entries
- Header manipulation

INSERTIONS

- Evidence of Program header relocation
- Dynamic string table extension
- Does SO name ptr point within dynamic string table.
- Duplication of Symbol names across Shared Objects

DT_NEEDED Infections

(Detection)

OVERWRITES

```
"dt_needed_indexes": [
    {
        "dt_needed_index": 0,
        "index_into_strtab": 1,
        "module_name": "libc.so.6",
        "name_in_dynstr": true
    },
    {
        "dt_needed_index": 12,
        "index_into_strtab": 68,
        "module_name": "libevil.so",
        "name_in_dynstr": false
    }
],
"dt_needed_wrong_order": true,
"dt_null_present": true,
"debug_section_present": false,
"dynstr_manipulated": true,
"headers_manipulated": true,
```

INSERTIONS

```
"dt_needed_indexes": [
    {
        "dt_needed_index": 0,
        "index_into_dt_strtab": 68,
        "module_name": "libevil.so",
        "name_in_dynstr": false
    },
    {
        "dt_needed_index": 1,
        "index_into_dt_strtab": 138,
        "module_name": "libdl.so.2",
        "name_in_dynstr": true
    },
    {
        "dt_needed_index": 2,
        "index_into_dt_strtab": 178,
        "module_name": "libc.so.6",
        "name_in_dynstr": true
    }
],
"dt_needed_wrong_order": false,
"dt_null_present": true,
"debug_section_present": true,
"dynstr_manipulated": true,
"headers_manipulated": true,
```

Preloading Abuse (LD_PRELOAD)

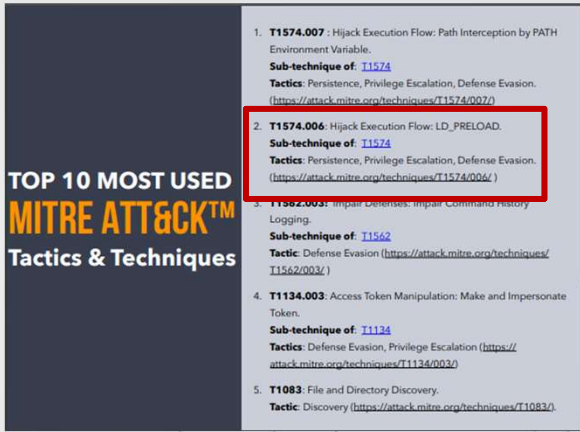
- Preloaded SO functions overwrite functions of non-preloaded SOs. Acting as a search order hijacking mechanism.

- Preloading mechanisms:
  - LD_PRELOAD env var
  - Dynamic linker '—preload' flag
  - /etc/ld.so.preload

- Preloading has legitimate uses: for debugging / compatibility

- Offers attackers a simple way to install hooks / execute constructor code

- Used by:
  - Azazel, BEURK, Jynx, Vlany. Umbreon Usermode rootkits
  - HiddenWasp malware & Threat groups.



https://vfeed.io/wp-content/uploads/2021/02/Top-10-Most-Used-MITRE-ATTCK.pdf

The Problem With Detecting Malicious Preloading

- Current detection solutions only monitor ‘existence’ of preloading rather than ‘effect’:
    - Command lines, paths & env variables.
    - Still requires manual analysis

The Problem With Detecting Malicious Preloading

- Current detection solutions only monitor ‘existence’ of preloading rather than ‘effect’:
  - Command lines, paths & env variables.
  - Still requires manual analysis

- More info required to distinguish malicious use of preloading:
  - Identify the individual hooks?
  - Which preloaded SOs are responsible?
  - Where is the location of the hook in memory?

The Problem With Detecting Malicious Preloading

- Current detection solutions only monitor 'existence' of preloading rather than 'effect':
  - Command lines, paths & env variables.
  - Still requires manual analysis

- More info required to distinguish malicious use of preloading:
  - Identify the individual hooks?
  - Which preloaded SOs are responsible?
  - Where is the location of the hook in memory?

- We need to provide more context and targeted analysis.

The Problem With Detecting Malicious Preloading

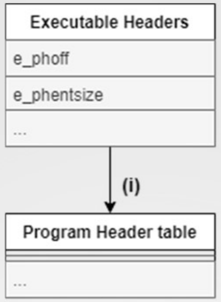

```
"ld_preload": "",
"ld_preload_present": false,
 "preload_hooked_funcs": [
        {
            "original_module_path": "/lib/x86_64-linux-gnu/libc.so.6,",
            "preload_func_addr": 140293666833890,
            "preload_module_path": "/lib/libselinux.so",
            "symbol_name": "__lxstat"
        },
        {
            "original_module_path": "/lib/x86_64-linux-gnu/libc.so.6,",
            "preload_func_addr": 140293666835356,
            "preload_module_path": "/lib/libselinux.so",
            "symbol_name": "__xstat"
        },
        {
            "original_module_path": "/lib/x86_64-linux-gnu/libc.so.6,",
            "preload_func_addr": 140293666833608,
            "preload_module_path": "/lib/libselinux.so",
            "symbol_name": "readdir"
        },
        {
            "original_module_path": "/lib/x86_64-linux-gnu/libc.so.6,",
            "preload_func_addr": 140293666834288,
            "preload_module_path": "/lib/libselinux.so",
            "symbol_name": "open"
        },
        {
            "original_module_path": "/lib/x86_64-linux-gnu/libc.so.6,",
            "preload_func_addr": 140293666833043,
            "preload_module_path": "/lib/libselinux.so",
            "symbol_name": "access"
        },
        {
            "original_module_path": "/lib/x86_64-linux-gnu/libc.so.6,",
            "preload_func_addr": 140293666833284,
            "preload_module_path": "/lib/libselinux.so",
            "symbol_name": "fopen"
        }
    ],
    "preload_hooking_present": true,
    "preloaded_libraries": [
        "/lib/libselinux.so"
    ],
    "proc_path": "/bin/bash",
    "timestamp": 1634739695
```


BEURK rootkit

- Current detection solutions only monitor 'existence' of preloading rather than 'effect':
    - Command lines, paths & env variables.
    - Still requires manual analysis
- More info required to distinguish malicious use of preloading:
    - Identify the individual hooks?
    - Which preloaded SOs are responsible?
    - Where is the location of the hook in memory?
- We need to provide more context and targeted analysis.

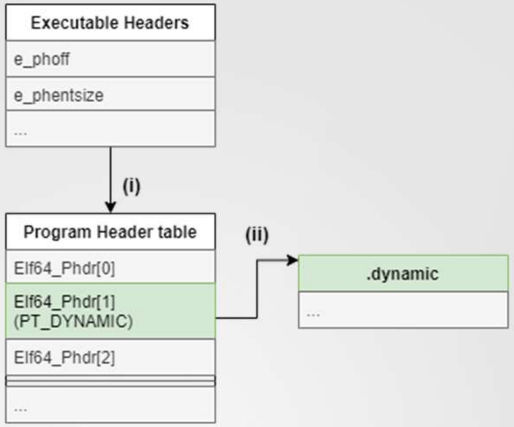
Step 1: Identifying All Imports

i. ELF executable header fields ***e_phoff*** & ***e_phentsize* ->** The program header table.



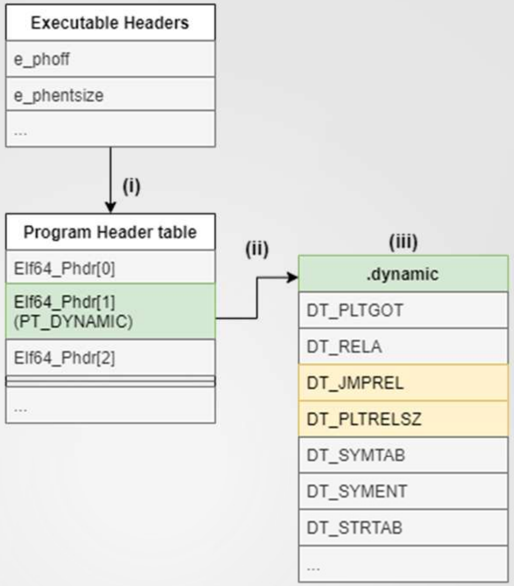
Step 1: Identifying All Imports

i. ELF executable header fields ***e_phoff*** & ***e_phentsize*** **->** The program header table.
ii. Header of entry (**PT_DYNAMIC)** **->** points to the Dynamic segment.



Step 1: Identifying All Imports

i. ELF executable header fields ***e_phoff*** & ***e_phentsize*** **->** The program header table.
ii. Header of entry (**PT_DYNAMIC)** **->** points to the Dynamic segment.
iii. Dynamic segment is 1:1 mapping of the *.dynamic section* containing pointers to:
   - Global offset table (**DT_PLTGOT**).
   - Relocation table (**DT_RELA**).
   - Dynamic symbol table (**DT_SYMTAB**).
   - Dynamic string table (**DT_STRTAB**).



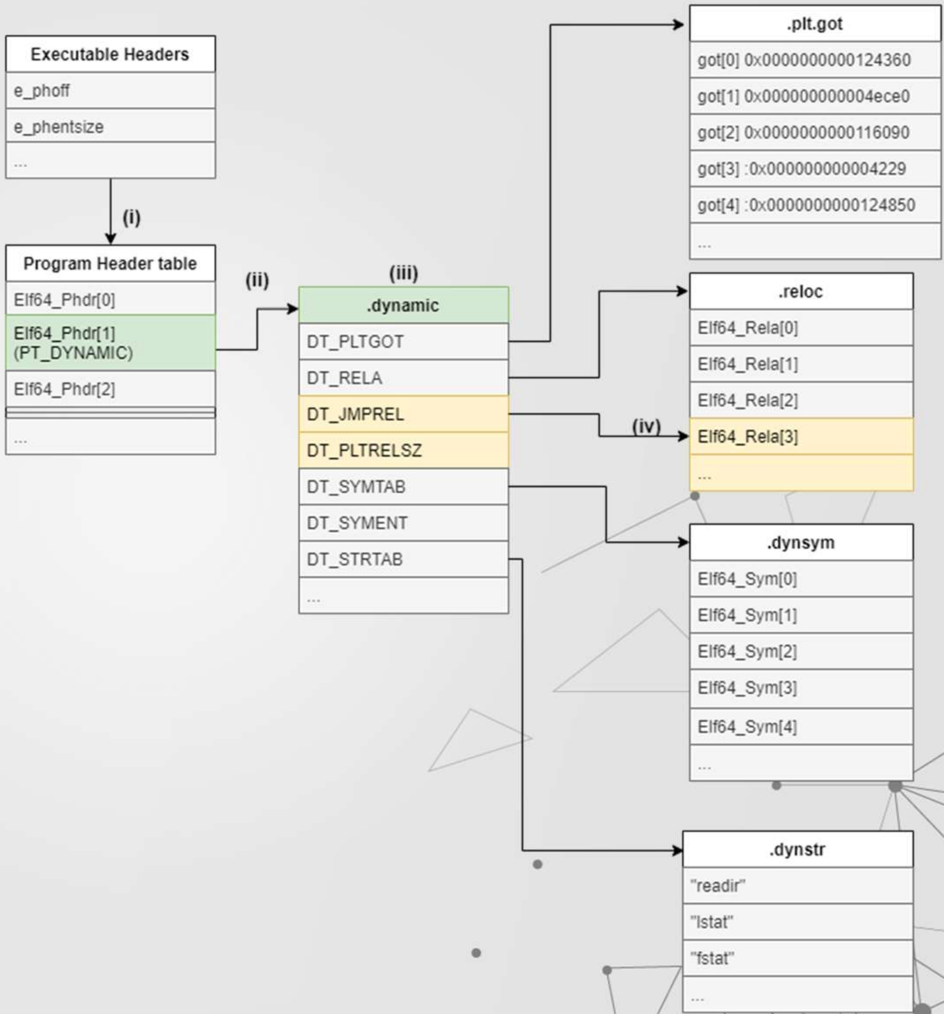
Step 1: Identifying All Imports

i. ELF executable header fields ***e_phoff*** & ***e_phentsize*** **->** The program header table.

ii. Header of entry (**PT_DYNAMIC)** **->** points to the Dynamic segment.

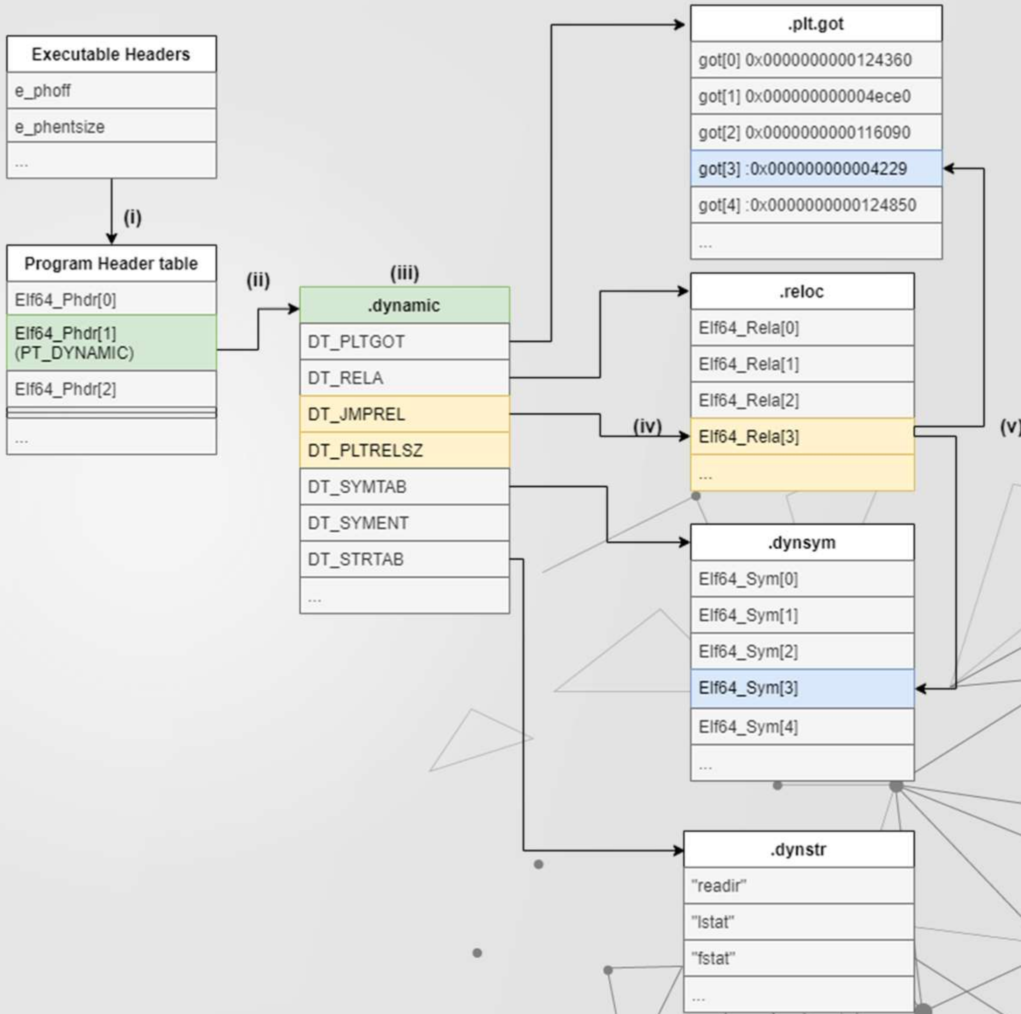
iii. Dynamic segment is 1:1 mapping of the *.dynamic section* containing pointers to:
   - Global offset table (**DT_PLTGOT**).
   - Relocation table (**DT_RELA**).
   - Dynamic symbol table (**DT_SYMTAB**).
   - Dynamic string table (**DT_STRTAB**).

iv. **DT_JMPREL** points to the first relocation table entry responsible for imports. **DT_PLTRELSZ** contains the total size of the relocation entries associated with imports. Together they can identify every relocation table entry (**Elf64_Rela**) responsible for an imported symbol.



Step 1: Identifying All Imports

i. ELF executable header fields ***e_phoff*** & ***e_phentsize* ->** The program header table.
ii. Header of entry (**PT_DYNAMIC)** **->** points to the Dynamic segment.
iii. Dynamic segment is 1:1 mapping of the *.dynamic section* containing pointers to:
  - Global offset table (**DT_PLTGOT**).
  - Relocation table (**DT_RELA**).
  - Dynamic symbol table (**DT_SYMTAB**).
  - Dynamic string table (**DT_STRTAB**).
iv. **DT_JMPREL** points to the first relocation table entry responsible for imports. **DT_PLTRELSZ** contains the total size of the relocation entries associated with imports. Together they can identify every relocation table entry (**Elf64_Rela**) responsible for an imported symbol.
v. Each import relocation entry (**ELF64_Rela)** entry contains:
  - The address of the associated **GOT** entry
  - The offset inside the Dynamic symbol table an entry relates to (**Elf64_Sym**).



Step 1: Identifying All Imports

i. ELF executable header fields ***e_phoff*** & ***e_phentsize*** **->** The program header table.

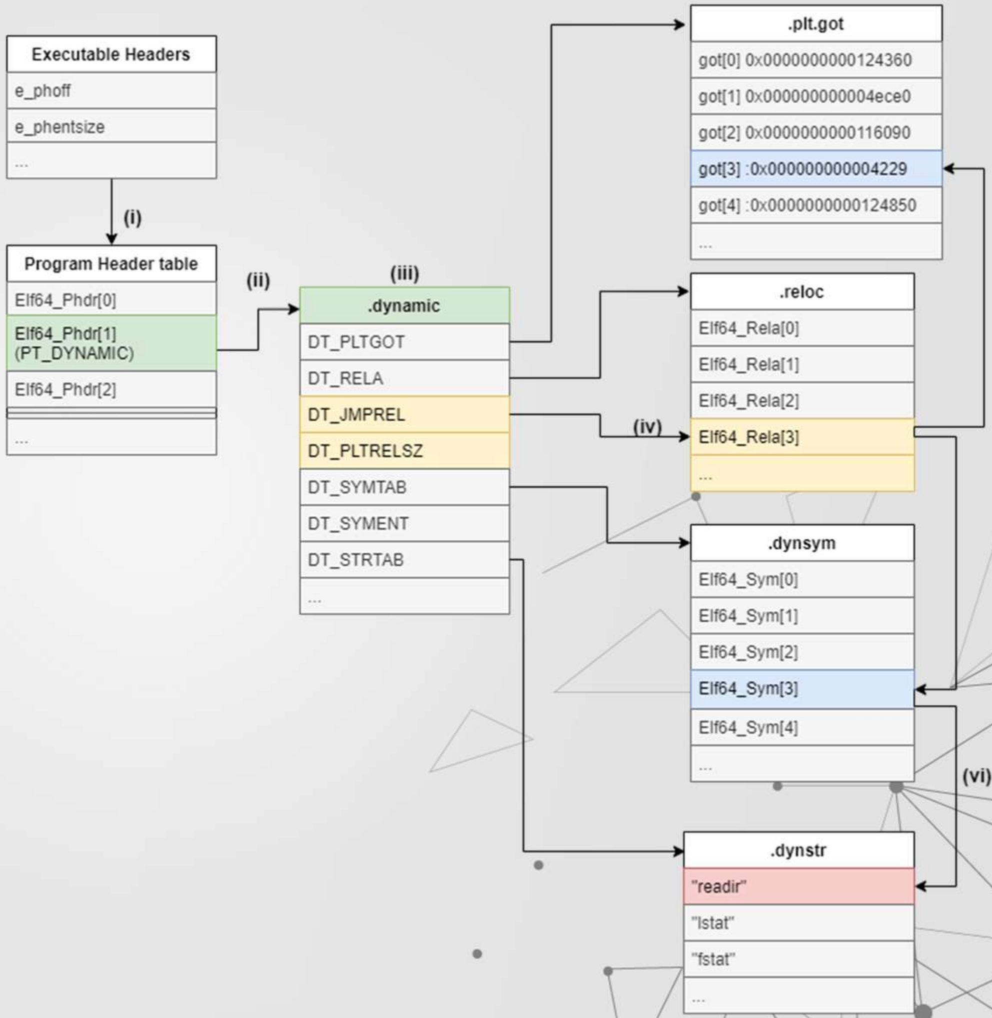
ii. Header of entry (**PT_DYNAMIC)** **->** points to the Dynamic segment.

iii. Dynamic segment is 1:1 mapping of the *.dynamic section* containing pointers to:
   - Global offset table (**DT_PLTGOT**).
   - Relocation table (**DT_RELA**).
   - Dynamic symbol table (**DT_SYMTAB**).
   - Dynamic string table (**DT_STRTAB**).

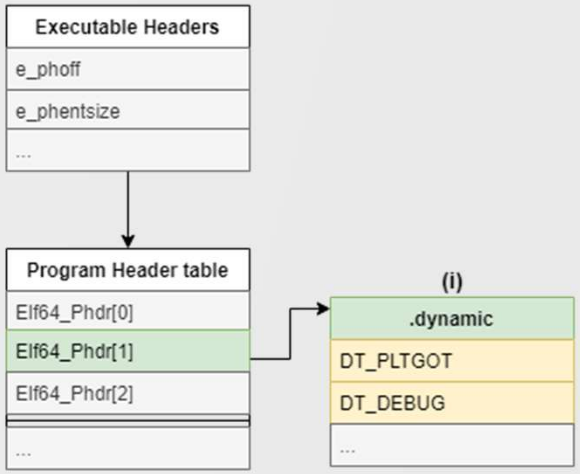
iv. **DT_JMPREL** points to the first relocation table entry responsible for imports. **DT_PLTRELSZ** contains the total size of the relocation entries associated with imports. Together they can identify every relocation table entry (**Elf64_Rela**) responsible for an imported symbol.

v. Each import relocation entry (**ELF64_Rela)** entry contains:
   - The address of the associated **GOT** entry
   - The offset inside the Dynamic symbol table an entry relates to (**Elf64_Sym**).

vi. **Elf64_Sym** entry contains the offset within the dynamic string table associated with the import (symbol) name.

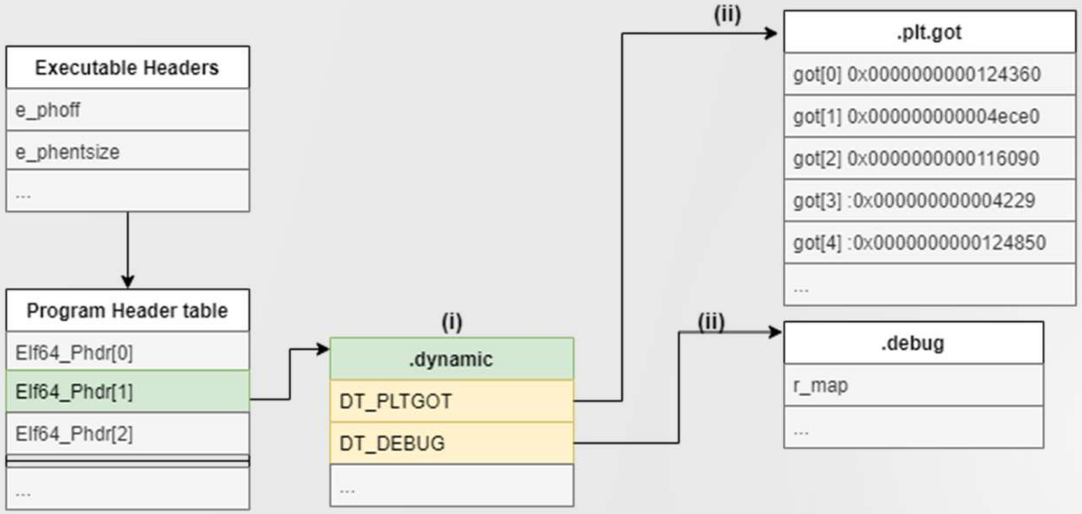


Step 2: Establish A List Of SOs & Their Base Addresses



i. Locate following sections in .dynamic section:
   - Global offset table (DT_PLTGOT).
   - Debug section (DT_DEBUG).

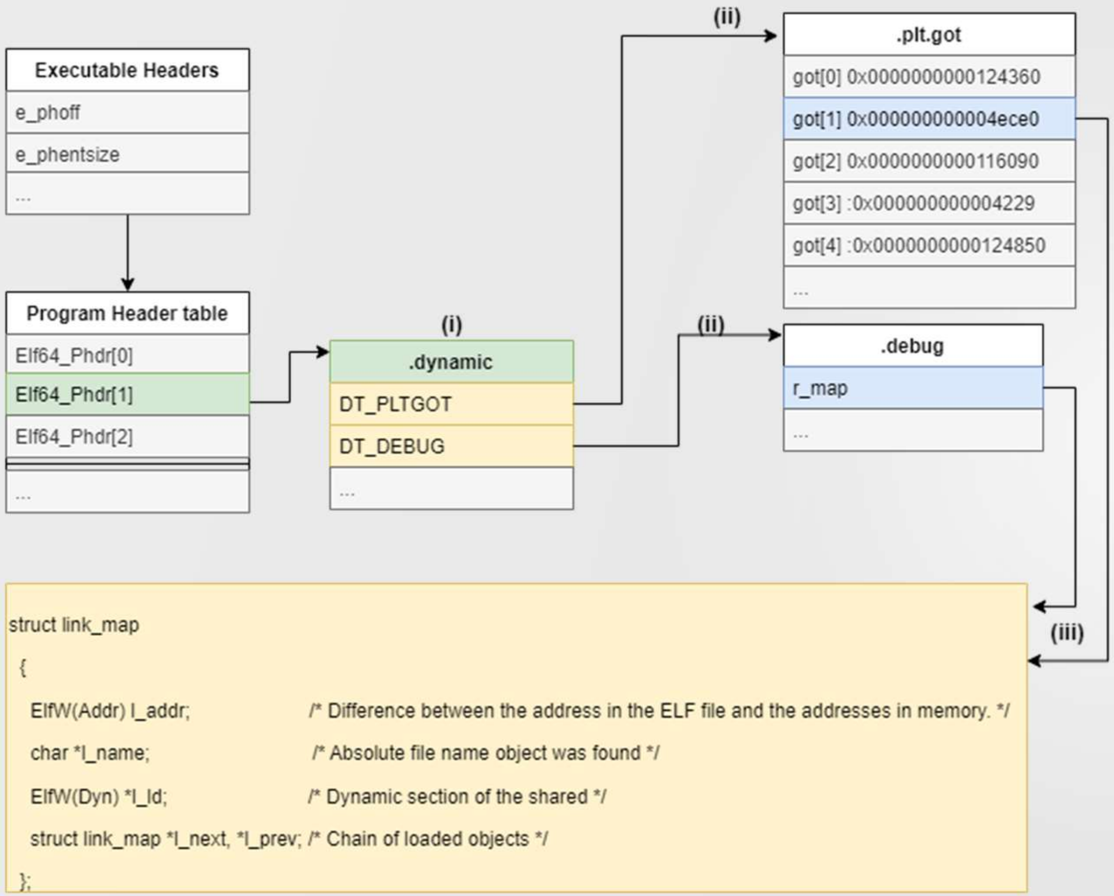
Step 2: Establish A List Of SOs & Their Base Addresses



i. Locate following sections in .dynamic section:
   - Global offset table (DT_PLTGOT).
   - Debug section (DT_DEBUG).

ii. Locate the address of a process' (link_map) using one of two methods:
   - Using DT_DEBUG - *r_map*
   - Using the GOT – *got[1]*

Step 2: Establish A List Of SOs & Their Base Addresses



i. Locate following sections in .dynamic section:
   - Global offset table (DT_PLTGOT).
   - Debug section (DT_DEBUG).

ii. Locate the address of a process' (link_map) using one of two methods:
   - Using DT_DEBUG - *r_map*
   - Using the GOT – *got[1]*

iii. Iterate through the link_map linked list and extract the loaded base address for each SO in memory.

Step 3: Identify Preloaded SOs & Exports

Identify preloaded SOs:

- Reading LD_PRELAOD from the stack.
- Reading */etc/ld.so.preload.*

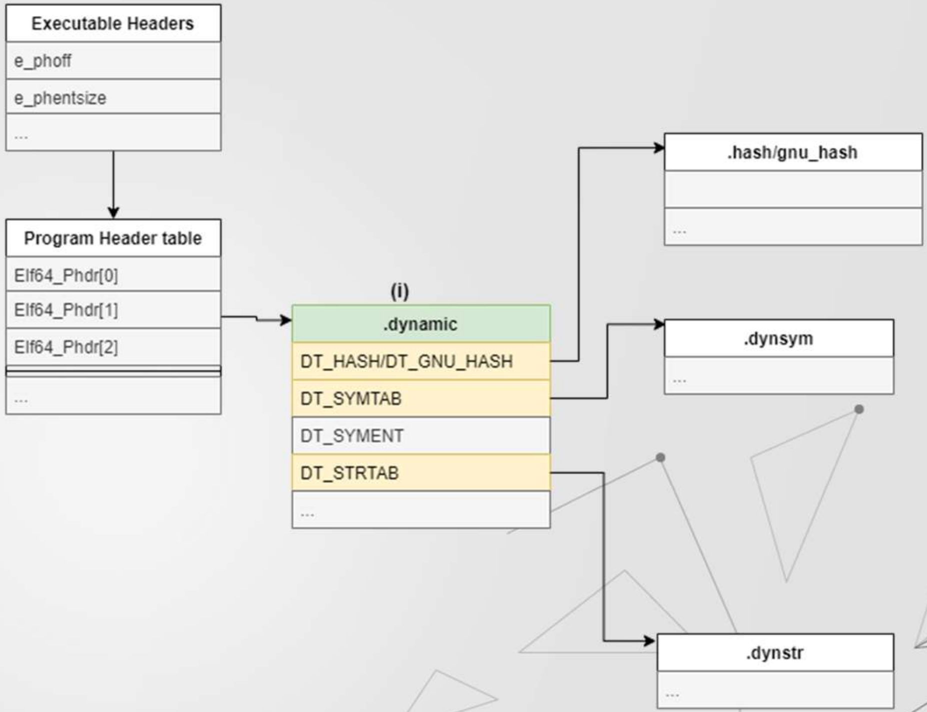
Step 3: Identify Preloaded SOs & Exports

Identify preloaded SOs:

- Reading LD_PRELAOD from the stack.
- Reading */etc/ld.so.preload.*

Resolve exported symbols.

i. Locate following sections in .dynamic section:
   - Hash table / GNU Hash table (DT_HASH / DT_GNU_HASH).
   - Dynamic symbol table (DT_SYMTAB).
   - Dynamic string table (DT_STRTAB).



Step 3: Identify Preloaded SOs & Exports

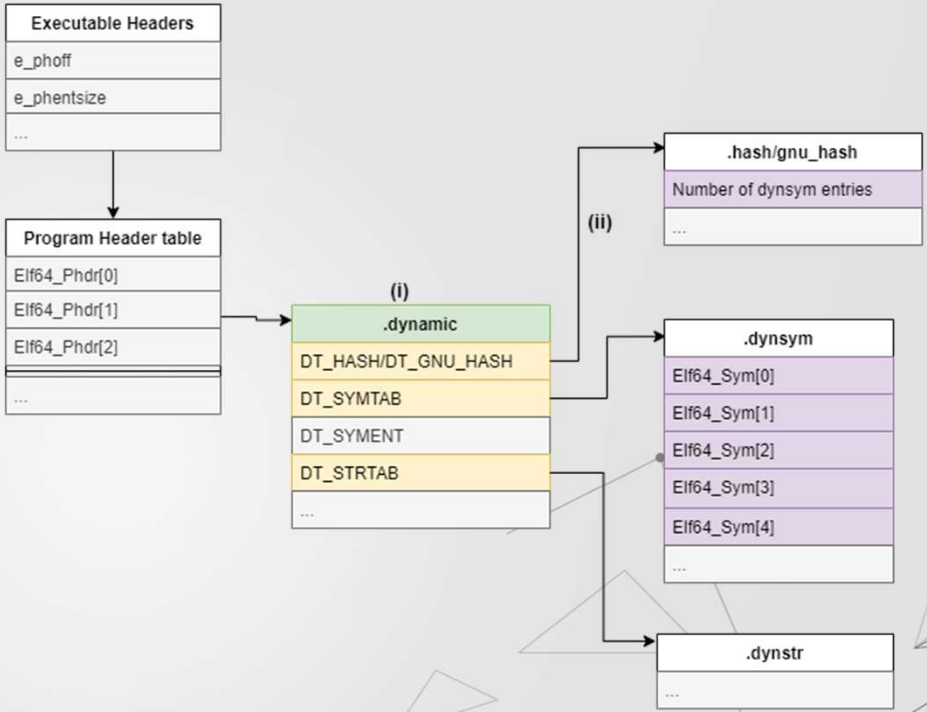
Identify preloaded SOs:

- Reading LD_PRELAOD from the stack.
- Reading */etc/ld.so.preload.*

Resolve exported symbols.

i. Locate following sections in .dynamic section:
   - Hash table / GNU Hash table (DT_HASH / DT_GNU_HASH).
   - Dynamic symbol table (DT_SYMTAB).
   - Dynamic string table (DT_STRTAB).

ii. Determine the number of symbol table entries using either:
   - Hash table.
   - GNU Hash table.



Step 3: Identify Preloaded SOs & Exports

<u>Identify preloaded SOs:</u>

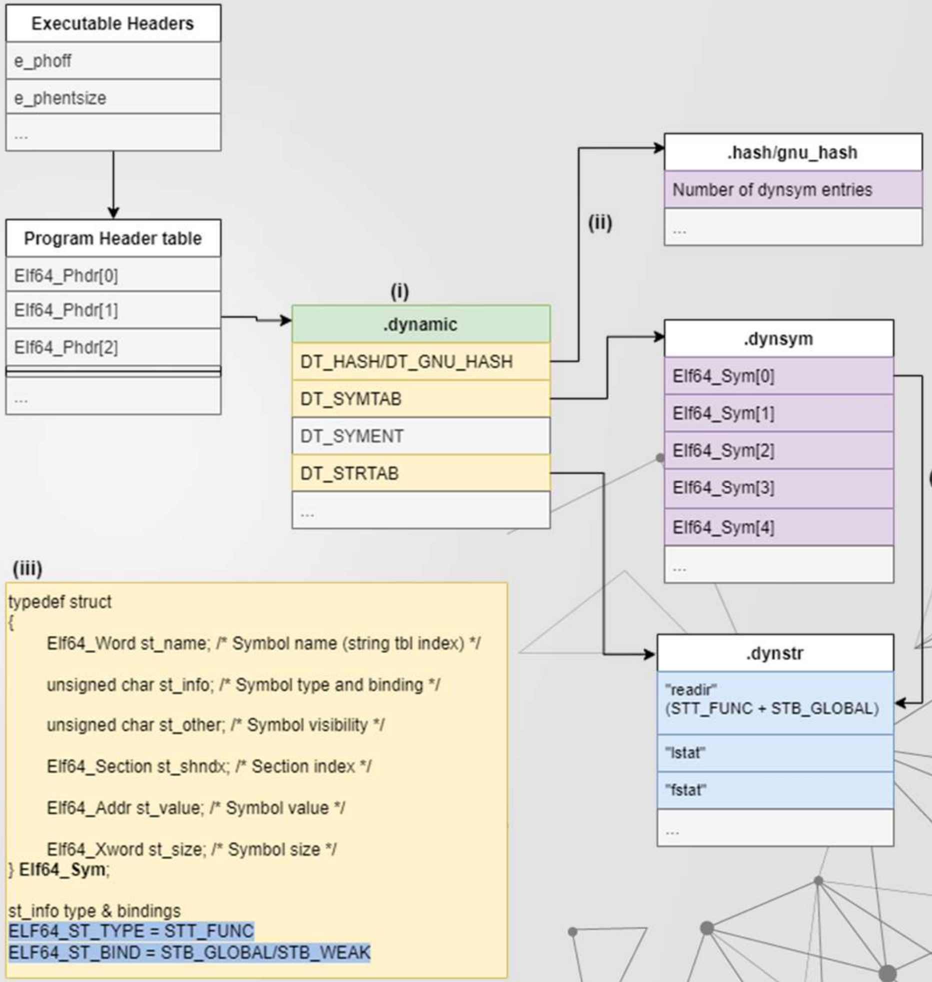
- Reading LD_PRELAOD from the stack.
- Reading */etc/ld.so.preload.*

<u>Resolve exported symbols.</u>

i. Locate following sections in .dynamic section:
   - Hash table / GNU Hash table (DT_HASH / DT_GNU_HASH).
   - Dynamic symbol table (DT_SYMTAB).
   - Dynamic string table (DT_STRTAB).

ii. Determine the number of symbol table entries using either:
   - Hash table.
   - GNU Hash table.

iii. Only collect exported symbols which are:
   - Type - STT_FUNC.
   - Binding - STB_GLOBAL/STB_WEAK.



Step 4 & 5: Comparisons & Matching Symbol Names

Step 4:

- Compare imported symbols with exported symbols from the any preloaded SOs.

Step 4 & 5: Comparisons & Matching Symbol Names

Step 4:

- Compare imported symbols with exported symbols from the any preloaded SOs.

Step 5:

- Resolve exports from non-preloaded SOs.
- Match legitimate export names with names of hooks to identify victim SOs.

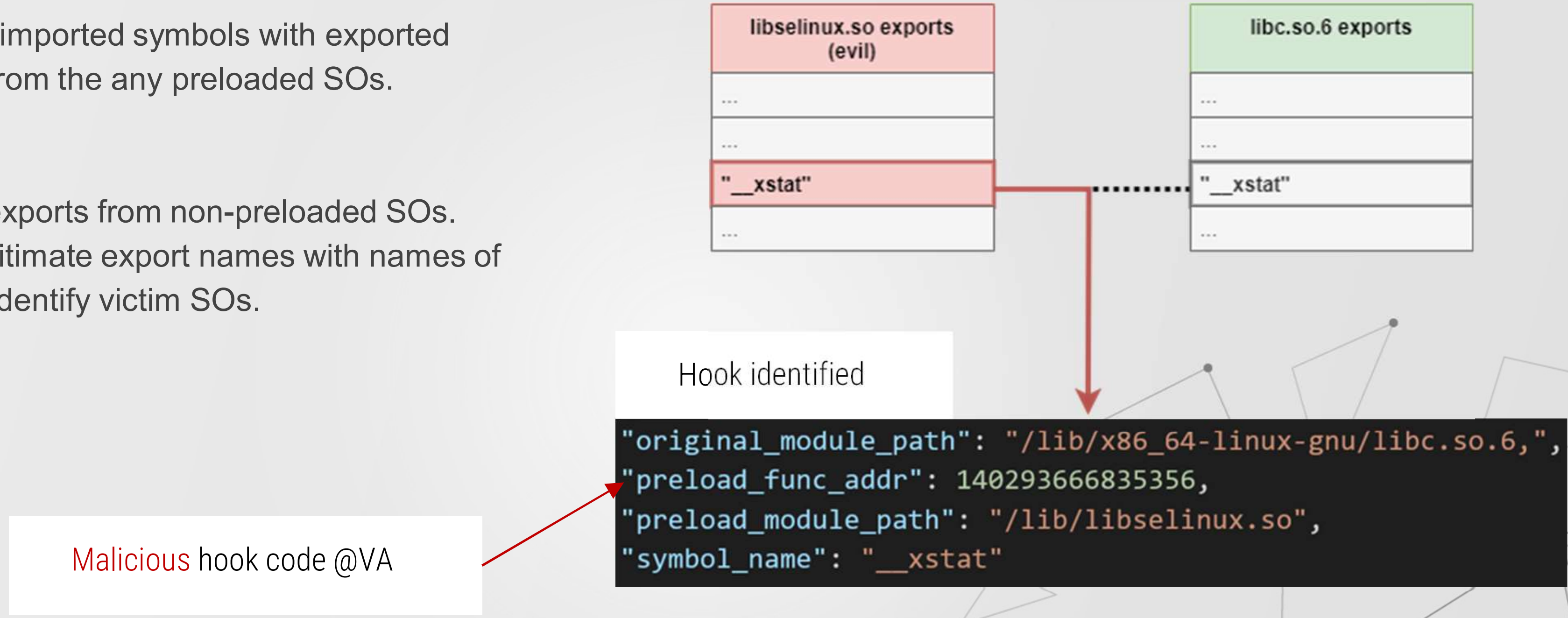
Step 4 & 5: Comparisons & Matching Symbol Names

Step 4:

- Compare imported symbols with exported symbols from the any preloaded SOs.

Step 5:

- Resolve exports from non-preloaded SOs.
- Match legitimate export names with names of hooks to identify victim SOs.



ELFieScanner

- Linux process memory scanning tool that detects various forms of:
  - Shared Object injection.
  - Shellcode injection & Process hollowing.
  - Entry point manipulation.
  - API Hooking.
- 43 different heuristics, controllable via configuration file.
- Multithreaded, written in C++, scans both x86/x64 processes.
- Outputs data into NDJSON file
- https://github.com/JanielDary/ELFieScanner

ELFIE-SCANNER

```
Usage: ./ELFie [options]

Option :
   -p                                              Scan single pid only
   -e                                              Run entrypoint scanner
   -l                                              Run library scanner
   -s                                              Run shellcode scanner
   -c </path/to/config.json>    Supply path to heuristic configuration file
   -h                                              Display this help

Examples :
./ELFie -e -l -s -c c:/test/my_config.json              Run all scanners across all processes
./ELFie -s -p 1604 -c c:/test/my_config.json    Run shellcode scanner on process ID 1604
```

04

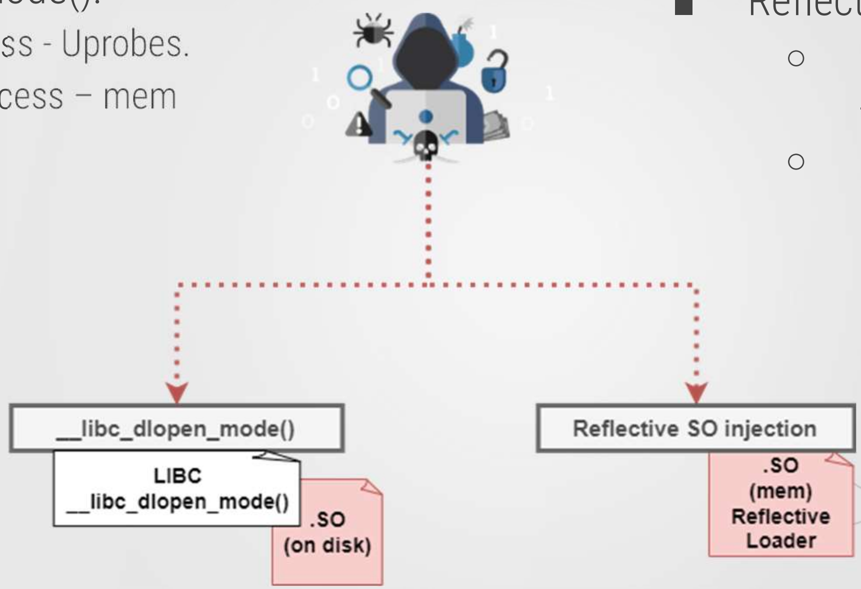
REALTIME INJECTION & TARGETING

Reflective Shared Object Injection & __libc_dlopen_mode()

Attack Techniques

- Direct use of __libc_dlopen_mode():
  - Detecting the victim process - Uprobes.
  - Detecting the attacker process – mem scan

- Reflective Shared object injection:
  - Identifying suspicious memory allocations - Kprobes
  - Identifying anomalous memory regions for injected SOs.



Existing Real-time Detection Strategies (SO Injection)

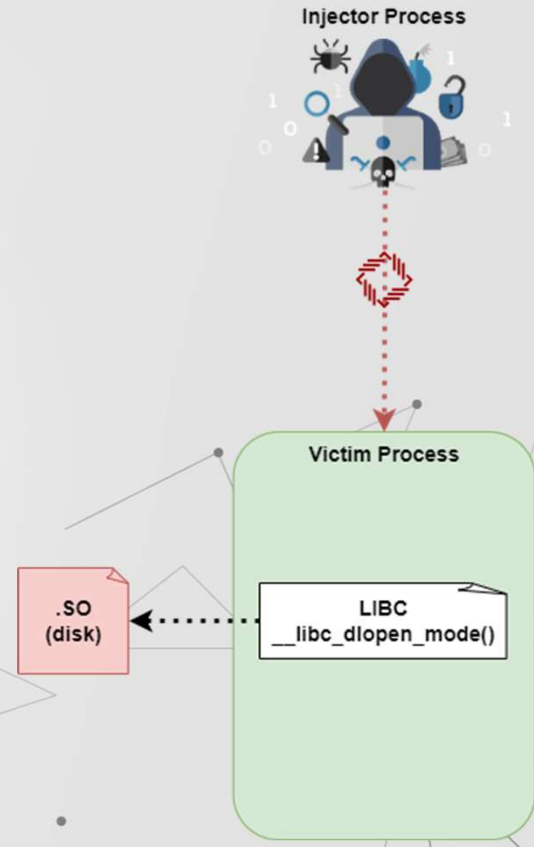
- Solutions include:
  - Monitoring/restricting the use of PTRACE() syscalls.
  - Enumerating /proc/<pid>/maps file for RWX regions.
  - Combining output with file events and command lines on a best effort basis.
  - Blindly scanning memory with Yara signatures.

- Issues with current solutions:
  - Browsers, debuggers, AVs and interpreters can exhibit the same behaviors in a legitimate way.
  - Solutions do not target SO injection specifically.
  - Can introduce data volume and backend performance issues.
  - Lots of data for an analyst to sift through.

__libc_dlopen_mode() Injection

- Two functions can be used to load a SO into a Linux process:
  - dlopen().
  - __libc_dlopen_mode().

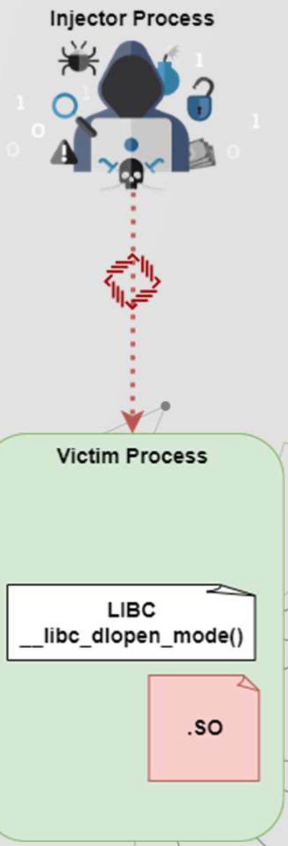
- Force a victim process to call either function ensures the dynamic linker does most of the work.
  - __libc_dlopen_mode() almost always targeted over dlopen().



__libc_dlopen_mode() Injection

- Two functions can be used to load a SO into a Linux process:
  - dlopen().
  - __libc_dlopen_mode().

- Force a victim process to call either function ensures the dynamic linker does most of the work.
  - __libc_dlopen_mode() almost always targeted over dlopen().

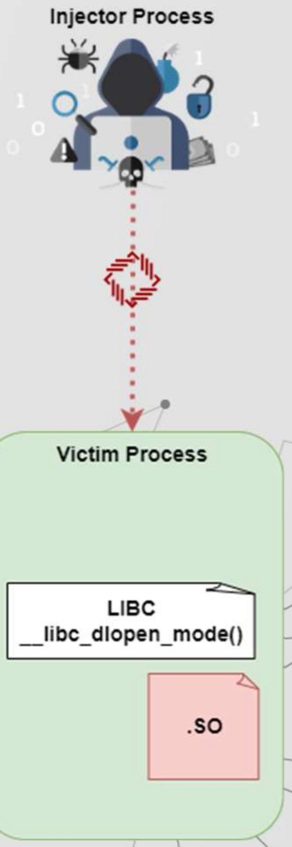


__libc_dlopen_mode() Injection

- Two functions can be used to load a SO into a Linux process:
  - dlopen().
  - __libc_dlopen_mode().

- Force a victim process to call either function ensures the dynamic linker does most of the work.
  - __libc_dlopen_mode() almost always targeted over dlopen().

- Method 1: Writing own injector:
  - Attach to a victim process with LIBC loaded.
  - Resolve the address of __libc_dlopen_mode() & modify the instruction pointer.
  - Replace registers (x64) or stack values (x86) with the correct arguments.
  - Resume execution.



__libc_dlopen_mode() Injection

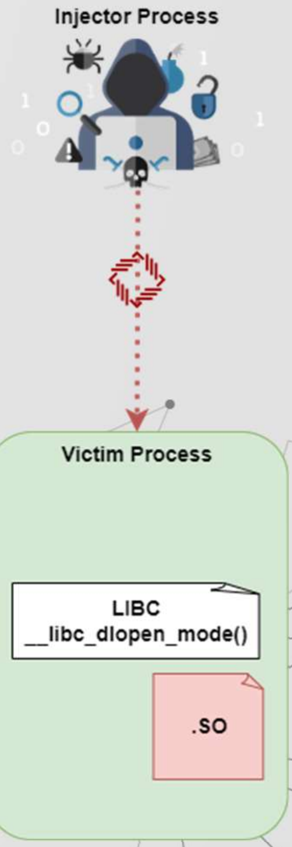
- Two functions can be used to load a SO into a Linux process:
  - dlopen().
  - __libc_dlopen_mode().

- Force a victim process to call either function ensures the dynamic linker does most of the work.
  - __libc_dlopen_mode() almost always targeted over dlopen().

- Method 1: Writing own injector:
  - Attach to a victim process with LIBC loaded.
  - Resolve the address of __libc_dlopen_mode() & modify the instruction pointer.
  - Replace registers (x64) or stack values (x86) with the correct arguments.
  - Resume execution.

- Method 2: Using a GDB bash one-liner.

```
echo 'print __libc_dlopen_mode("/tmp/sample_library.so", 2)' | gdb -p <PID>
```



Monitoring Function Calls

LTrace()

- Can only target individual / groups of processes.
- Uses PTRACE – (Slow + Invasive)
- Malicious processes can prevent itself being debugged using PTRACE_TRACEME

Monitoring Function Calls

LTrace()

- Can only target individual / groups of processes.
- Uses PTRACE – (Slow + Invasive)
- Malicious processes can prevent itself being debugged using PTRACE_TRACEME

Dynamic Instrumentation (DI)

Uprobes

- Introduced in Linux 3.5
- System wide effect
- DI of User level functions & offsets
- Defining a Uprobe requires:
  - Path of SO.
  - Offset to target function
  - Selected function parameters & corresponding register/stack values.

Kprobes

- Introduced in Linux 2.69
- System wide effect
- DI of Kernel level functions & offsets
- Often used by eBPF programs

Calculating The Offset To __libc_dlopen_mode
(Uprobe)

- Method 1: Using nm -

```
nm -D /usr/lib/x86_64-linux-gnu/libc-2.32.so | grep __libc_dlopen_mode
```

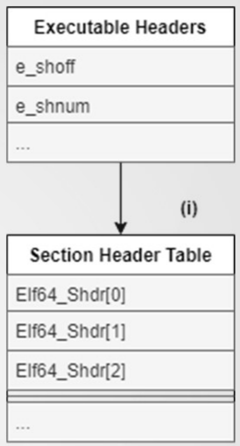
Calculating The Offset To __libc_dlopen_mode
(Uprobe)

- Method 1: Using nm -

```
nm -D /usr/lib/x86_64-linux-gnu/libc-2.32.so | grep __libc_dlopen_mode
```

- Method 2: Manually enumerating the SO on disk:

  i. Locate Section Hdrs table via Exe Hdrs.



Calculating The Offset To __libc_dlopen_mode
(Uprobe)

- Method 1: Using nm -

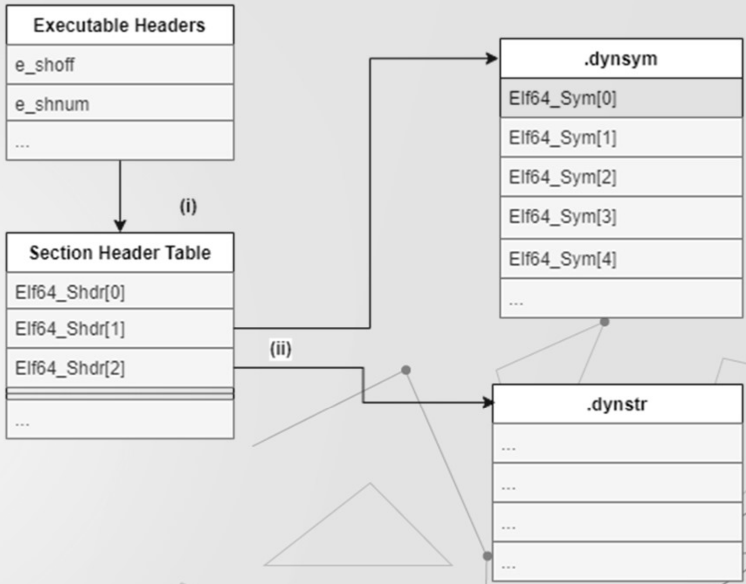
```
nm -D /usr/lib/x86_64-linux-gnu/libc-2.32.so | grep __libc_dlopen_mode
```

- Method 2: Manually enumerating the SO on disk:

i. Locate Section Hdrs table via Exe Hdrs.

ii. Use Section Hdrs table to locate:

  - the dynamic symbol table (.dynsym)
  - dynamic string table (.dynstr)



Calculating The Offset To __libc_dlopen_mode (Uprobe)

- Method 1: Using nm -

```
nm -D /usr/lib/x86_64-linux-gnu/libc-2.32.so | grep __libc_dlopen_mode
```

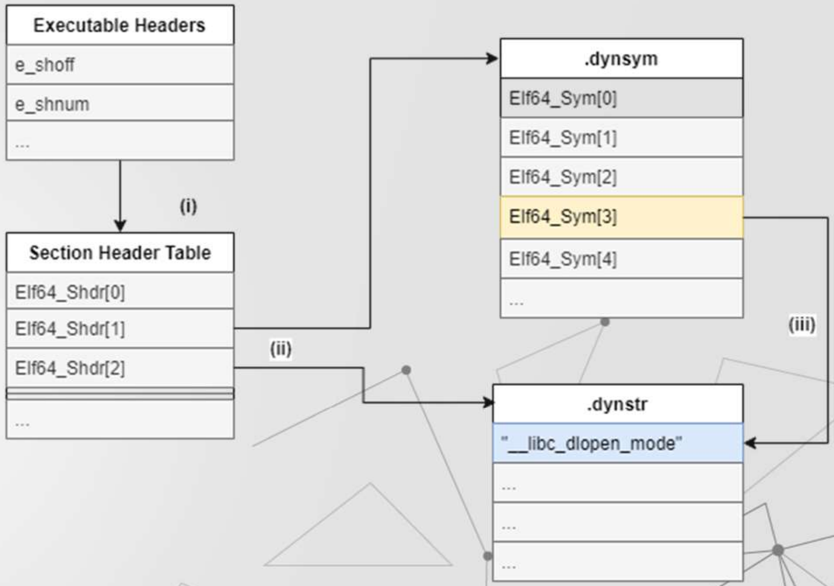
- Method 2: Manually enumerating the SO on disk:

i. Locate Section Hdrs table via Exe Hdrs.

ii. Use Section Hdrs table to locate:

  - the dynamic symbol table (.dynsym)
  - dynamic string table (.dynstr)

iii. Enumerate .dynsym & .dynstr tables to match symbol names with Elfxx_Sym entry.



Calculating The Offset To __libc_dlopen_mode
(Uprobe)

- Method 1: Using nm -

```
nm -D /usr/lib/x86_64-linux-gnu/libc-2.32.so | grep __libc_dlopen_mode
```

- Method 2: Manually enumerating the SO on disk:

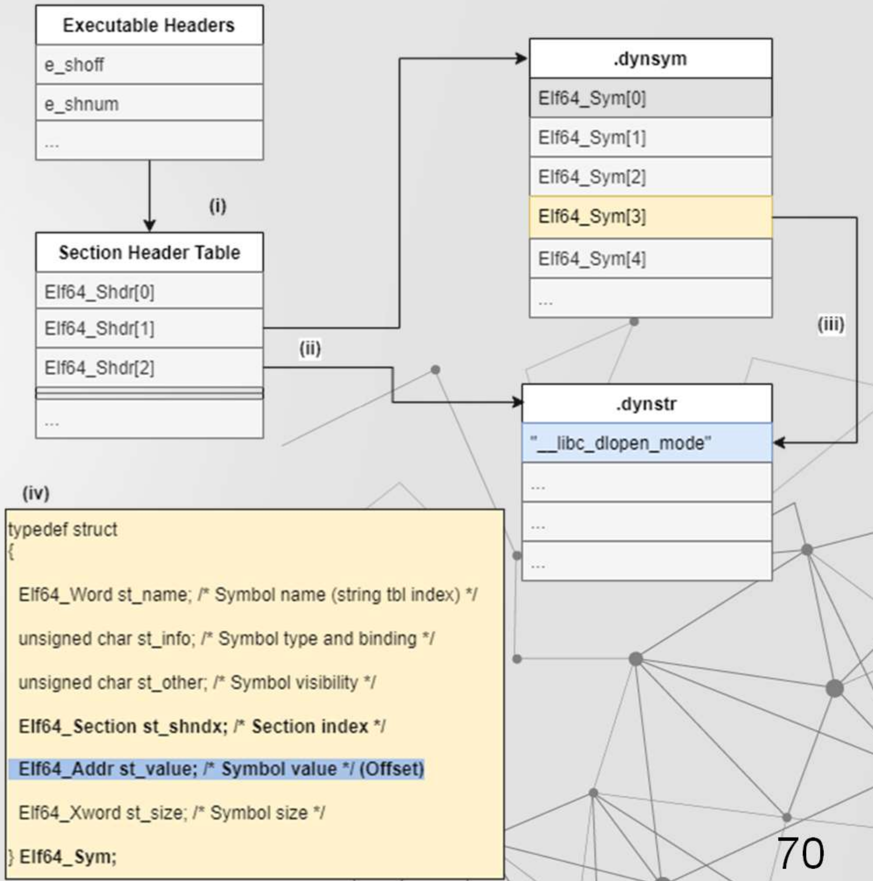
i. Locate Section Hdrs table via Exe Hdrs.

ii. Use Section Hdrs table to locate:

  - the dynamic symbol table (.dynsym)
  - dynamic string table (.dynstr)

iii. Enumerate .dynsym & .dynstr tables to match symbol names with Elfxx_Sym entry.

iv. Read ELF64_Sym.st_value for '__libc_dlopen_mode' to determine its file offset.



Determining The Function Parameters (Uprobe)

- __libc_dlopen_mode() uses same two parameters as dlopen():
  - *name* - Path of SO (rdi).
  - *mode* - Loading method flag (rsi).

- Identify any variations between GLIBC versions.

Function prototype

dl-libc.c 2 ×

elf > dl-libc.c > __libc_dlopen_mode(const char *, i

```c
/* ... and these functions call dler

void *
__libc_dlopen_mode (const char *name, int mode)
{
  struct do_dlopen_args args;
  args.name = name;
  args.mode = mode;
  args.caller_dlopen = RETURN_ADDRESS (0);

#ifdef SHARED
  if (!rtld_active ())
    return GLRO (dl_dlfcn_hook)->libc_dlopen_mode (name, mode);
#endif
  return dlerror_run (do_dlopen, &args) ? NULL : (void *) args.map;
}
```

dlfcn.h ×

usr > include > x86_64-linux-gnu > bits > dlfcn.h > ...

```c
/* The MODE argument to `dlopen' contains one of the following: */
#define RTLD_LAZY 0x00001 /* Lazy function call binding. */
#define RTLD_NOW 0x00002 /* Immediate function call binding. */
#define  RTLD_BINDING_MASK   0x3 /* Mask of binding time value. */
#define RTLD_NOLOAD 0x00004 /* Do not load the object. */
#define RTLD_DEEPBIND   0x00008 /* Use deep binding. */

/* If the following bit is set in the MODE argument to `dlopen',
   the symbols of the loaded object and its dependencies are made
   visible as if the object were linked directly into the program. */
#define RTLD_GLOBAL 0x00100

/* Unix98 demands the following flag which is the inverse to RTLD_GLOBAL.
   The implementation does this by default and so we can define the
   value to zero. */
#define RTLD_LOCAL   0

/* Do not delete object when closed. */
#define RTLD_NODELETE   0x01000
```

Loading method flags

Determining The Function Parameters (Uprobe)

- __libc_dlopen_mode() uses same two parameters as dlopen():
  - *name* - Path of SO (rdi).
  - *mode* - Loading method flag (rsi).
- Identify any variations between GLIBC versions.
- Definition of Uprobe:

Brendan Gregg's F-Trace Uprobe wrapper:

```
./uprobe
```

Function prototype

dl-libc.c

```c
/* ... and these functions call dler
void *
__libc_dlopen_mode (const char *name, int mode)
{
  struct do_dlopen_args args;
  args.name = name;
  args.mode = mode;
  args.caller_dlopen = RETURN_ADDRESS (0);

#ifdef SHARED
  if (!rtld_active ())
    return GLRO (dl_dlfcn_hook)->libc_dlopen_mode (name, mode);
#endif
  return dlerror_run (do_dlopen, &args) ? NULL : (void *) args.map;
}
```

dlfcn.h

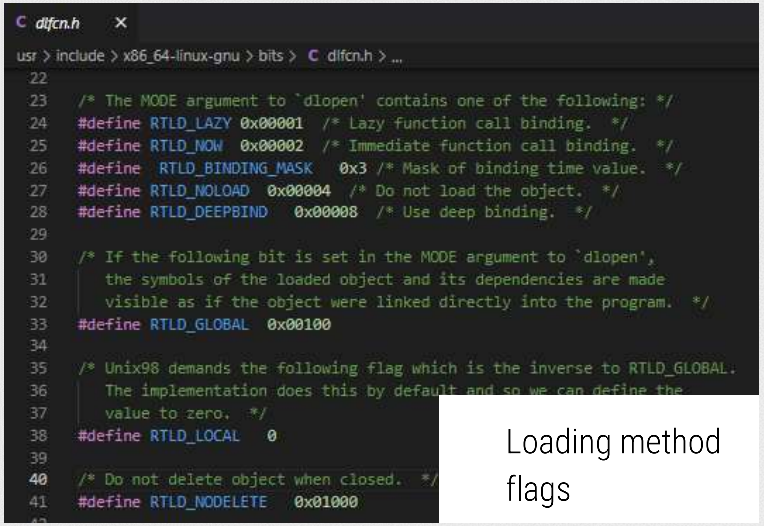
usr > include > x86_64-linux-gnu > bits > dlfcn.h > ...

```c
/* The MODE argument to `dlopen' contains one of the following: */
#define RTLD_LAZY 0x00001 /* Lazy function call binding.  */
#define RTLD_NOW  0x00002 /* Immediate function call binding.  */
#define  RTLD_BINDING_MASK   0x3 /* Mask of binding time value.  */
#define RTLD_NOLOAD  0x00004 /* Do not load the object.  */
#define RTLD_DEEPBIND   0x00008 /* Use deep binding.  */

/* If the following bit is set in the MODE argument to `dlopen',
   the symbols of the loaded object and its dependencies are made
   visible as if the object were linked directly into the program.  */
#define RTLD_GLOBAL  0x00100

/* Unix98 demands the following flag which is the inverse to RTLD_GLOBAL.
   The implementation does this by default and so we can define the
   value to zero.  */
#define RTLD_LOCAL   0

/* Do not delete object when closed.  */
#define RTLD_NODELETE   0x01000
```

Loading method flags

Determining The Function Parameters (Uprobe)

- __libc_dlopen_mode() uses same two parameters as dlopen():
  - *name* - Path of SO (rdi).
  - *mode* - Loading method flag (rsi).
- Identify any variations between GLIBC versions.
- Definition of Uprobe:
  - Path to libc.
  - Offset to __libc_dlopen_mode().

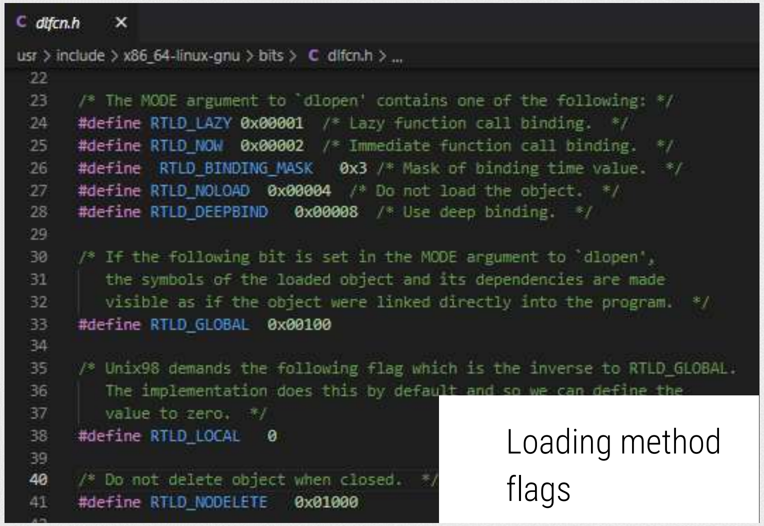


Brendan Gregg's F-Trace Uprobe wrapper:

```
./uprobe -H 'p:/lib/x86_64-linux-gnu/libc-2.32.so:0x1598a0
```

Determining The Function Parameters (Uprobe)

- __libc_dlopen_mode() uses same two parameters as dlopen():
  - *name* - Path of SO (rdi).
  - *mode* - Loading method flag (rsi).
- Identify any variations between GLIBC versions.
- Definition of Uprobe:
  - Path to libc.
  - Offset to __libc_dlopen_mode().
  - The first parameter 'path' renaming this to 'injected lib' from the rdi register.



Brendan Gregg's F-Trace Uprobe wrapper:

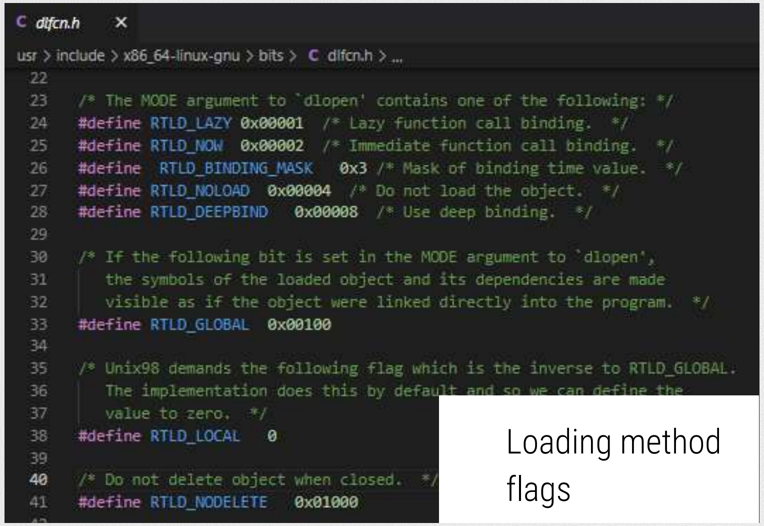
```
./uprobe -H 'p:/lib/x86_64-linux-gnu/libc-2.32.so:0x1598a0 injected_lib=+0(%di):string
```

Determining The Function Parameters (Uprobe)

- __libc_dlopen_mode() uses same two parameters as dlopen():
  - *name* - Path of SO (rdi).
  - *mode* - Loading method flag (rsi).

- Identify any variations between GLIBC versions.

- Definition of Uprobe:
  - Path to libc.
  - Offset to __libc_dlopen_mode().
  - The first parameter ‘path’ renaming this to ‘injected lib’ from the rdi register.
  - The second parameter mode from the rsi register to a 32bit hexadecimal format.



Brendan Gregg’s F-Trace Uprobe wrapper:

```
./uprobe -H 'p:/lib/x86_64-linux-gnu/libc-2.32.so:0x1598a0 injected_lib=+0(%di):string mode=%si:x32'
```

Uprobe Telemetry

(Victim Process)

Detection

kubo / **injector**

gaffe23 / **linux-inject**

```
Tracing uprobe __libc_dlopen_mode (p:__libc_dlopen_mode /lib/x86_64-linux-gnu/libc-2.32.so:0x1598a0 injected_lib=+0(%di):string mode=%si:x32). Ctrl-C to end.
# tracer: nop
#
# entries-in-buffer/entries-written: 0/0   #P:2
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID    CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |    ||||       |         |
 test-target-x86-27445   [001] d... 24219.601016: __libc_dlopen_mode: (0x7f1d6f8a88a0) injected_lib="/root/injector/tests/test-library-x86_64.so" mode=0x1
            bash-29478   [001] d... 24223.567678: __libc_dlopen_mode: (0x7fadb44ff8a0) injected_lib="libnss_files.so.2" mode=0x80000002
          groups-29480   [000] d... 24223.569504: __libc_dlopen_mode: (0x7f9c216628a0) injected_lib="libnss_files.so.2" mode=0x80000002
            bash-29495   [000] d... 24224.039769: __libc_dlopen_mode: (0x7f6bc96e98a0) injected_lib="libnss_files.so.2" mode=0x80000002
           <...>-29497   [001] d... 24224.041122: __libc_dlopen_mode: (0x7fb0a7a018a0) injected_lib="libnss_files.so.2" mode=0x80000002
            bash-29503   [000] d... 24224.505258: __libc_dlopen_mode: (0x7f5ca29128a0) injected_lib="libnss_files.so.2" mode=0x80000002
           <...>-29505   [001] d... 24224.506904: __libc_dlopen_mode: (0x7f8ef46388a0) injected_lib="libnss_files.so.2" mode=0x80000002
```

Uprobe Telemetry

(Victim Process)

Detection | kubo / **injector** | gaffe23 / **linux-inject**

```
Tracing uprobe __libc_dlopen_mode (p:__libc_dlopen_mode /lib/x86_64-linux-gnu/libc-2.32.so:0x1598a0 injected_lib=+0(%di):string mode=%si:x32). Ctrl-C to end.
# tracer: nop
#
# entries-in-buffer/entries-written: 0/0   #P:2
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID     CPU#  ||||    TIMESTAMP  FUNCTION
#              | |         |   ||||       |         |
 test-target-x86-27445   [001] d... 24219.601016: __libc_dlopen_mode: (0x7f1d6f8a88a0) injected_lib="/root/injector/tests/test-library-x86_64.so" mode=0x1
```

Methods Of Detecting The Injector Process

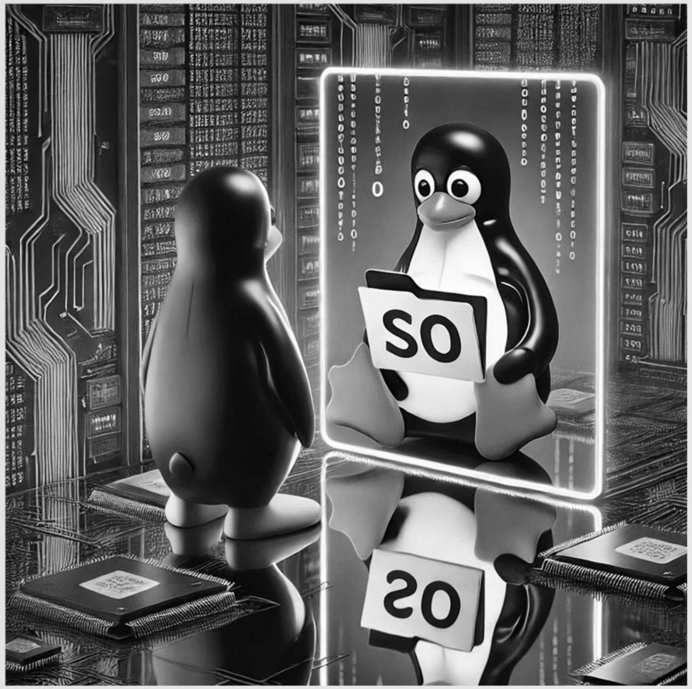
1. Using existing telemetry to find the most recent *PTRACE_ATTACH* event prior to the Uprobe firing. This will be the injector process

2. Signature on command line arguments supplied to GDB containing *'__libc_dlopen_mode'*.

3. Search a running process' *.rodata* section for references to *__libc_dlopen_mode()*:
   - Only works if the injector process still exists.


```
{
    "__libc_dlopen_mode_present": true,
    "__libc_dlopen_mode_present_in": "RODATA",
    "anonymous_memory_mappings": [],
    "base_address": 94759707078656,
    "cmdl": "./injector",
```

Reflective SO Injection



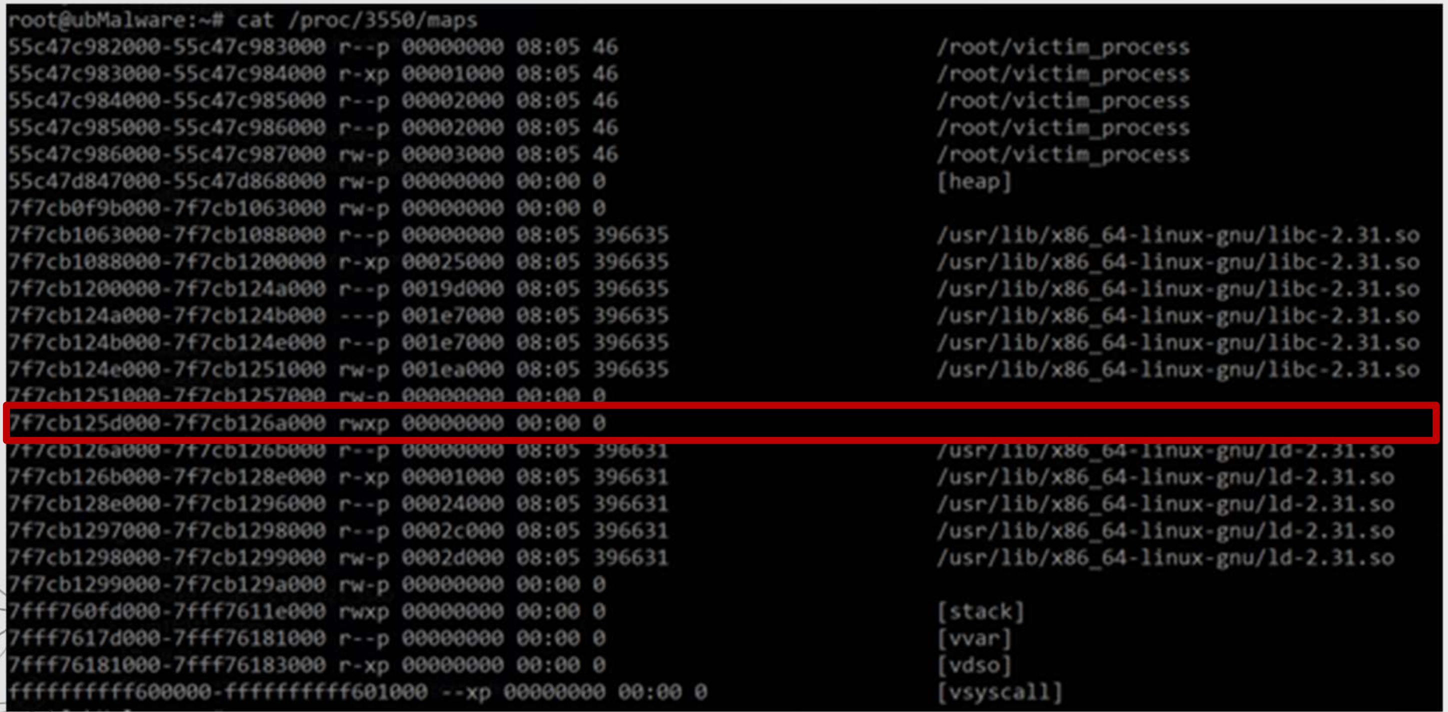
- The Linux equivilent of Reflective DLL injection on Windows, used by:
  - InfoSecguerrilla/ReflectiveSOInjection tool.

  - N1nj4sec/Pupy framework.

- Facilitates the loading of a SO directly from memory by using a custom loader:
  - Allocates a RWX anonymous memory region.
  - Maps a SO into the region.
  - Uses Libc exports to resolve symbols and perform relocations.

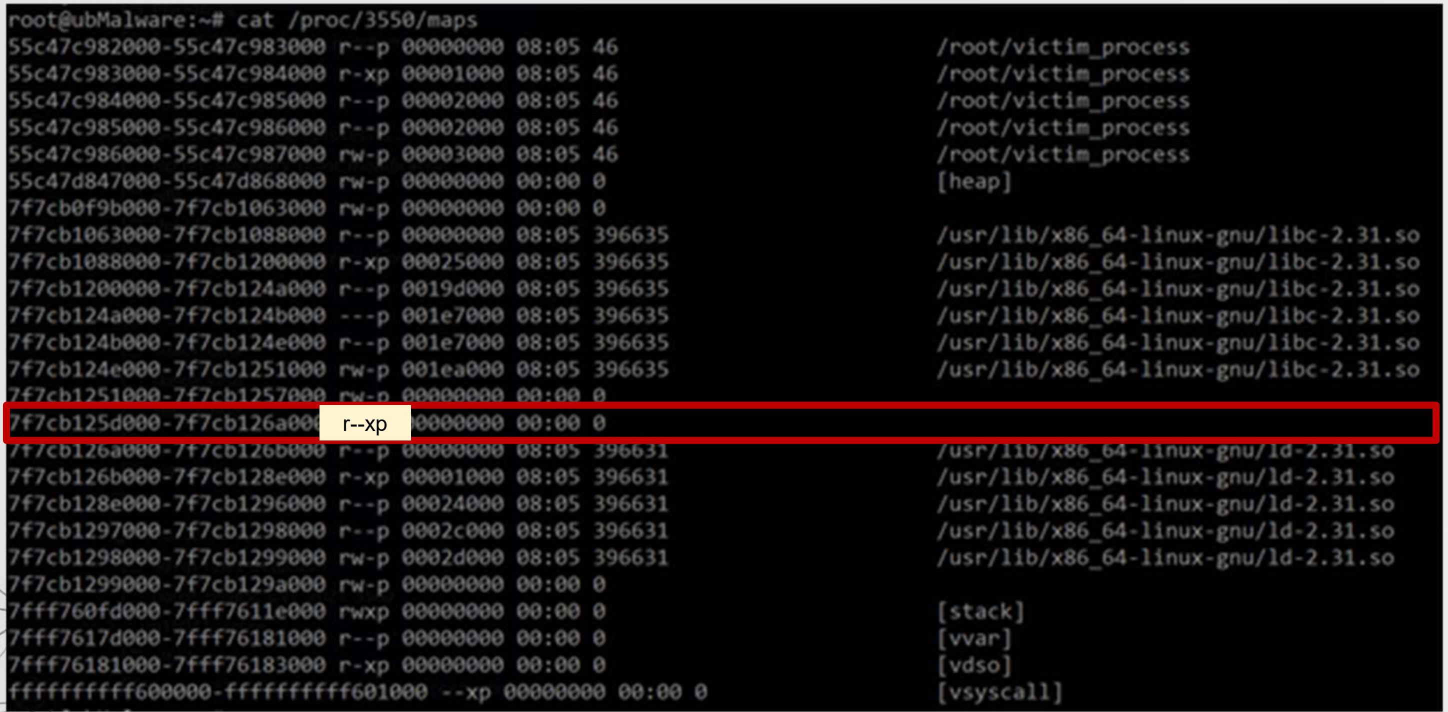
Reflective SO Injection

- Current detection strategies rely on identifying existing RWX regions, this can be easily circumvented by:



Reflective SO Injection

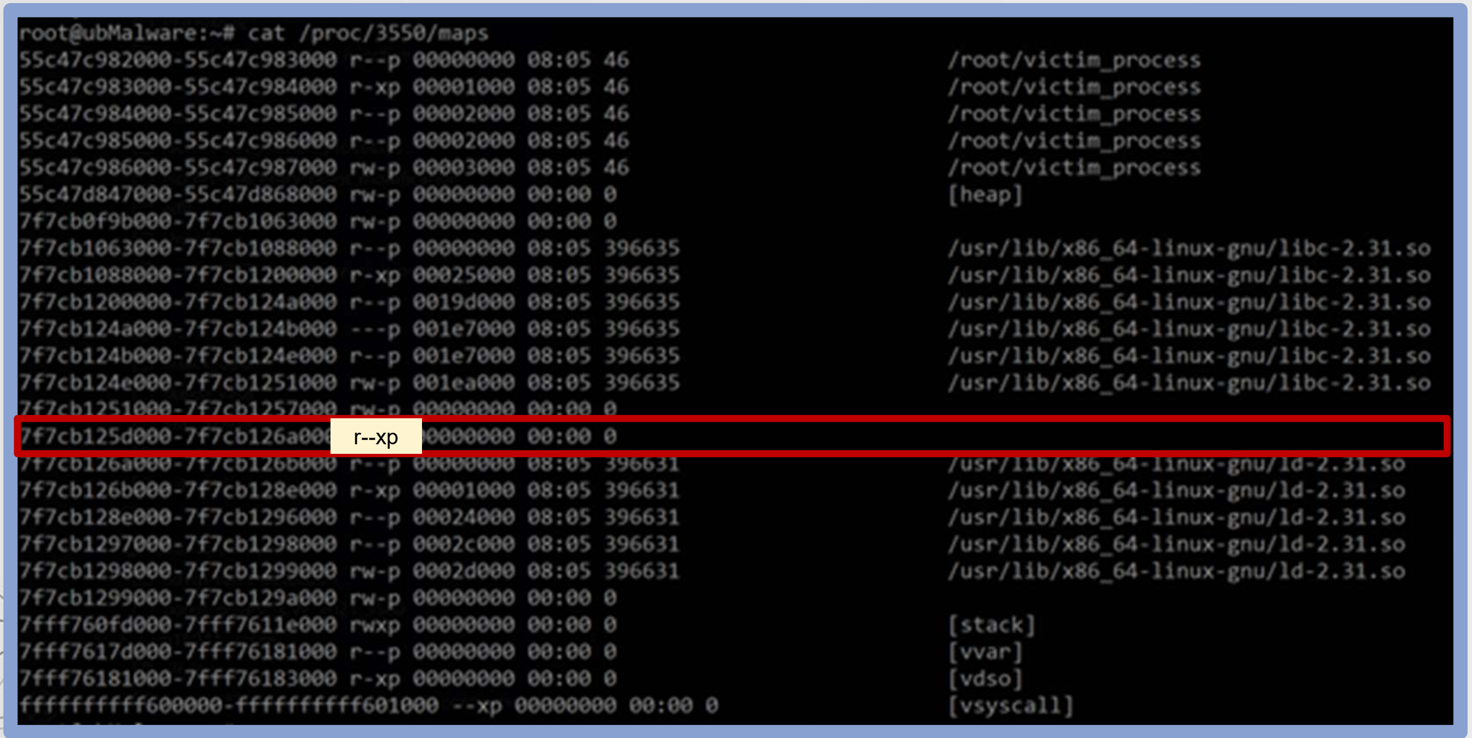
- Current detection strategies rely on identifying existing RWX regions, this can be easily circumvented by:
  - Modifying page permissions - *mprotect()*



```
root@ubMalware:~# cat /proc/3550/maps
55c47c982000-55c47c983000 r--p 00000000 08:05 46                         /root/victim_process
55c47c983000-55c47c984000 r-xp 00001000 08:05 46                         /root/victim_process
55c47c984000-55c47c985000 r--p 00002000 08:05 46                         /root/victim_process
55c47c985000-55c47c986000 r--p 00002000 08:05 46                         /root/victim_process
55c47c986000-55c47c987000 rw-p 00003000 08:05 46                         /root/victim_process
55c47d847000-55c47d868000 rw-p 00000000 00:00 0                          [heap]
7f7cb0f9b000-7f7cb1063000 rw-p 00000000 00:00 0
7f7cb1063000-7f7cb1088000 r--p 00000000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7cb1088000-7f7cb1200000 r-xp 00025000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7cb1200000-7f7cb124a000 r--p 0019d000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7cb124a000-7f7cb124b000 ---p 001e7000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7cb124b000-7f7cb124e000 r--p 001e7000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7cb124e000-7f7cb1251000 rw-p 001ea000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f7cb1251000-7f7cb1257000 rw-p 00000000 00:00 0
7f7cb125d000-7f7cb126a00 r--xp 0000000 00:00 0
7f7cb126a000-7f7cb126b000 r--p 00000000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7cb126b000-7f7cb128e000 r-xp 00001000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7cb128e000-7f7cb1296000 r--p 00024000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7cb1297000-7f7cb1298000 r--p 0002c000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7cb1298000-7f7cb1299000 rw-p 0002d000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f7cb1299000-7f7cb129a000 rw-p 00000000 00:00 0
7fff760fd000-7fff7611e000 rwxp 00000000 00:00 0                          [stack]
7fff7617d000-7fff76181000 r--p 00000000 00:00 0                          [vvar]
7fff76181000-7fff76183000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

Reflective SO Injection

- Current detection strategies rely on identifying existing RWX regions, this can be easily circumvented by:
  - Modifying page permissions - *mprotect()*
  - Spoofing process mappings - */proc/<pid>/maps*



Using Kprobes To Target Memory Allocations

- Target the initial memory allocation.

- Exported Kernel Symbols found in */proc/kallsyms*.

- mmap() not exported:
  - Internally calls *sys_mmap->ksys_mmap_pgoff*.

```
asmlinkage unsigned long
sys_mmap (unsigned long addr, unsigned long len, int prot, int flags, int fd, long off)
{
        if (offset_in_page(off) != 0)
                return -EINVAL;

        addr = ksys_mmap_pgoff(addr, len, prot, flags, fd, off >> PAGE_SHIFT);
        if (!IS_ERR((void *) addr))
                force_successful_syscall_return();
        return addr;
}
```

Using Kprobes To Target Memory Allocations

- Target the initial memory allocation.

- Exported Kernel Symbols found in */proc/kallsyms*.

- mmap() not exported:
  - Internally calls *sys_mmap->ksys_mmap_pgoff*.

```
asmlinkage unsigned long
sys_mmap (unsigned long addr, unsigned long len, int prot, int flags, int fd, long off)
{
        if (offset_in_page(off) != 0)
                return -EINVAL;

        addr = ksys_mmap_pgoff(addr, len, prot, flags, fd, off >> PAGE_SHIFT);
        if (!IS_ERR((void *) addr))
                force_successful_syscall_return();
        return addr;
}
```

```
unsigned long ksys_mmap_pgoff(unsigned long addr, unsigned long len,
                              unsigned long prot, unsigned long flags,
                              unsigned long fd, unsigned long pgoff)
{
```

Using Kprobes To Target Memory Allocations

- Target the initial memory allocation.
- Exported Kernel Symbols found in */proc/kallsyms*.
- mmap() not exported:
  - Internally calls *sys_mmap->ksys_mmap_pgoff*.

```c
asmlinkage unsigned long
sys_mmap (unsigned long addr, unsigned long len, int prot, int flags, int fd, long off)
{
        if (offset_in_page(off) != 0)
                return -EINVAL;

        addr = ksys_mmap_pgoff(addr, len, prot, flags, fd, off >> PAGE_SHIFT);
        if (!IS_ERR((void *) addr))
                force_successful_syscall_return();
        return addr;
}
```

```c
unsigned long ksys_mmap_pgoff(unsigned long addr, unsigned long len,
                              unsigned long prot, unsigned long flags,
                              unsigned long fd, unsigned long pgoff)
{
```

ReflectiveLoader.c | inject.c | mman-linux.h

usr > include > x86_64-linux-gnu > bits > mman-linux.h > PROT_EXEC

```c
#define PROT_READ 0x1       /* Page can be read.  */
#define PROT_WRITE   0x2       /* Page can be written.  */
#define PROT_EXEC 0x4       /* Page can be executed.  */
#define PROT_NONE 0x0       /* Page can not be accessed.  */
```

ReflectiveLoader.c | mman-linux.h

usr > include > x86_64-linux-gnu > bits > mman-linux.h > ...

```c
#define MAP_SHARED   0x01     /* Share changes.  */
#define MAP_PRIVATE  0x02     /* Changes are private.  */
#ifdef __USE_MISC
# define MAP_SHARED_VALIDATE  0x03  /* Share changes and validate
                                       extension flags.  */
# define MAP_TYPE 0x0f      /* Mask for type of mapping.  */
#endif

/* Other flags.  */
#define MAP_FIXED 0x10      /* Interpret addr exactly.  */
#ifdef __USE_MISC
# define MAP_FILE 0
# ifdef __MAP_ANONYMOUS
#  define MAP_ANONYMOUS  __MAP_ANONYMOUS   /* Don't use a file.  */
# else
#  define MAP_ANONYMOUS 0x20     /* Don't use a file.  */
# endif
```

The Probe & Telemetry

```
./kprobe 'p:ksys_mmap_pgoff addr=%di:x32 len=%si:x32 prot=%dx:x32 flags=%cx:x32 fd=%r8:x32
off=%r9:x32' 'flags==0x22&&prot==0x7'
```

```
root@ubMalware:~/perf-tools/kernel# ./kprobe 'p:ksys_mmap_pgoff addr=%di:x32 len=%si:x32 prot=%dx:x32 flags=%cx:x32 fd=%r8:x32 off=%r9:x32' 'flags==0x22&&prot==0x7'
Tracing kprobe ksys_mmap_pgoff. Ctrl-C to end.
  victim_process-6030    [001] .... 10682.995045: ksys_mmap_pgoff: (ksys_mmap_pgoff+0x0/0x2a0) addr=0x0 len=0xc930 prot=0x7 flags=0x22 fd=0xffffffff off=0x0
```

- A Kprobe can be used to target:
  - Anonymous memory allocations.
  - With initial RWX / RX permissions.
- Multiple probes can be set for each allocation variation & change e.g. mprotect()

The Probe & Telemetry

```
./kprobe 'p:ksys_mmap_pgoff addr=%di:x32 len=%si:x32 prot=%dx:x32 flags=%cx:x32 fd=%r8:x32 off=%r9:x32' 'flags==0x22&&prot==0x7'
```

```
root@ubMalware:~/perf-tools/kernel# ./kprobe 'p:ksys_mmap_pgoff addr=%di:x32 len=%si:x32 prot=%dx:x32 flags=%cx:x32 fd=%r8:x32 off=%r9:x32' 'flags==0x22&&prot==0x7'
Tracing kprobe ksys_mmap_pgoff. Ctrl-C to end.
  victim_process-6030    [001] .... 10682.995045: ksys_mmap_pgoff: (ksys_mmap_pgoff+0x0/0x2a0) addr=0x0 len=0xc930 prot=0x7 flags=0x22 fd=0xffffffff off=0x0
```

- A Kprobe can be used to target:
  - Anonymous memory allocations.
  - With initial RWX / RX permissions.
- Multiple probes can be set for each allocation variation & change e.g. mprotect()
- Capture the memory address & length supplied to ksys_mmap_pgoff to trigger a targeted scan.


```
"anonymous_memory_mappings": [
    {
        "elf_magic_index": 1024,
        "elf_magic_present": true,
        "end_addr": 140428348862464,
        "is_p": 1,
        "is_r": 1,
        "is_w": 1,
        "is_x": 1,
        "start_addr": 140428348858368
    }
],
"base_address": 94719054659584,
"cmdl": "/root/victim_process",
```

05

HIDE & SEEK

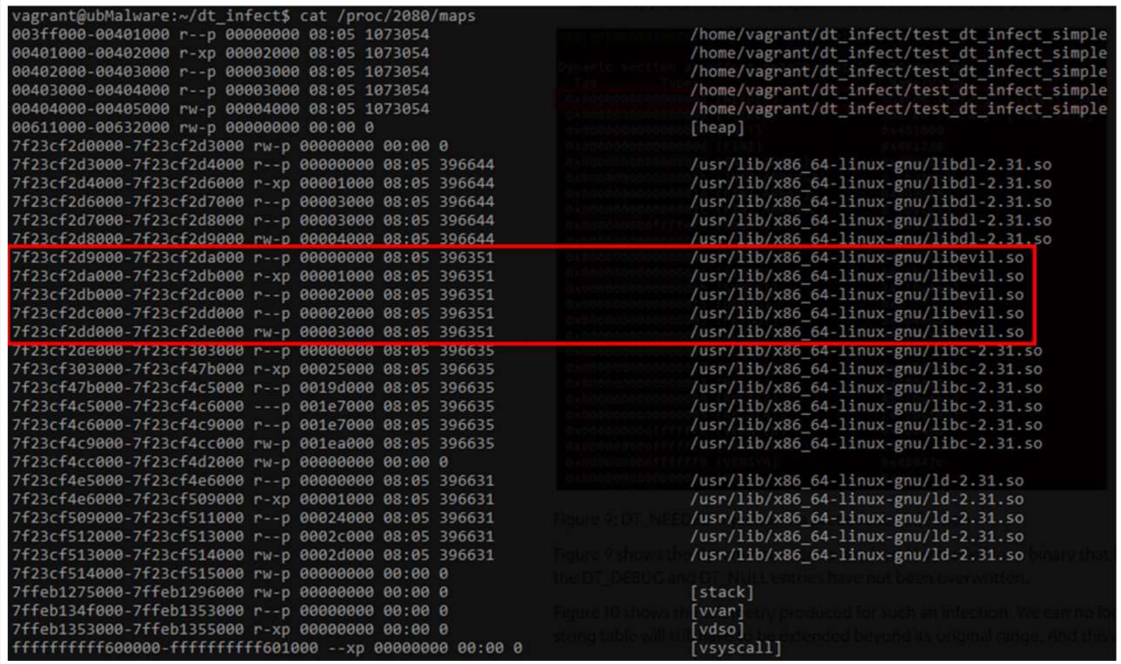
Hidden Shared Objects
& Detection Rules

Hidden Shared Objects

Process Mappings

The ‘proc/<pid>/maps’ is the pseudo-filesystem representation of a process’ memory mappings, this includes it’s loaded SOs

Monero miner (libprocesshider)

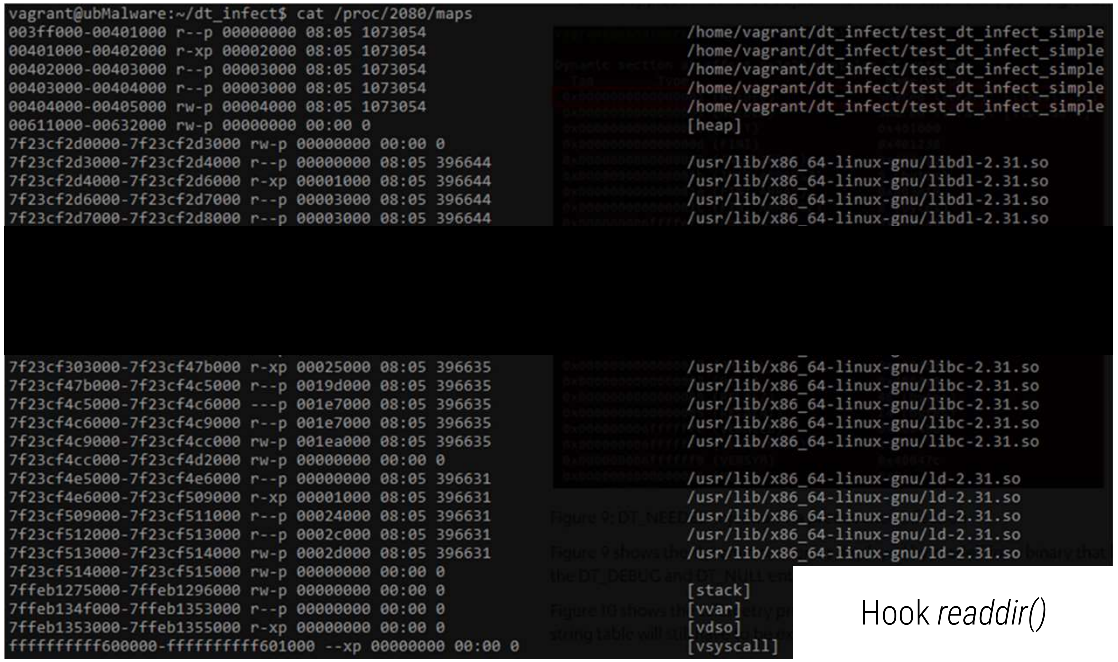


Hidden Shared Objects

Process Mappings

The 'proc/<pid>/maps' is the pseudo-filesystem representation of a process' memory mappings, this includes it's loaded SOs

Monero miner (libprocesshider)



```
vagrant@ubMalware:~/dt_infect$ cat /proc/2080/maps
003ff000-00401000 r--p 00000000 08:05 1073054                            /home/vagrant/dt_infect/test_dt_infect_simple
00401000-00402000 r-xp 00002000 08:05 1073054                            /home/vagrant/dt_infect/test_dt_infect_simple
00402000-00403000 r--p 00003000 08:05 1073054                            /home/vagrant/dt_infect/test_dt_infect_simple
00403000-00404000 r--p 00003000 08:05 1073054                            /home/vagrant/dt_infect/test_dt_infect_simple
00404000-00405000 rw-p 00004000 08:05 1073054                            /home/vagrant/dt_infect/test_dt_infect_simple
00611000-00632000 rw-p 00000000 00:00 0                                  [heap]
7f23cf2d0000-7f23cf2d3000 rw-p 00000000 00:00 0
7f23cf2d3000-7f23cf2d4000 r--p 00000000 08:05 396644                     /usr/lib/x86_64-linux-gnu/libdl-2.31.so
7f23cf2d4000-7f23cf2d6000 r-xp 00001000 08:05 396644                     /usr/lib/x86_64-linux-gnu/libdl-2.31.so
7f23cf2d6000-7f23cf2d7000 r--p 00003000 08:05 396644                     /usr/lib/x86_64-linux-gnu/libdl-2.31.so
7f23cf2d7000-7f23cf2d8000 r--p 00003000 08:05 396644                     /usr/lib/x86_64-linux-gnu/libdl-2.31.so
[redacted]
7f23cf303000-7f23cf47b000 r-xp 00025000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f23cf47b000-7f23cf4c5000 r--p 0019d000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f23cf4c5000-7f23cf4c6000 ---p 001e7000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f23cf4c6000-7f23cf4c9000 r--p 001e7000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f23cf4c9000-7f23cf4cc000 rw-p 001ea000 08:05 396635                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f23cf4cc000-7f23cf4d2000 rw-p 00000000 00:00 0
7f23cf4e5000-7f23cf4e6000 r--p 00000000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f23cf4e6000-7f23cf509000 r-xp 00001000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f23cf509000-7f23cf511000 r--p 00024000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f23cf512000-7f23cf513000 r--p 0002c000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f23cf513000-7f23cf514000 rw-p 0002d000 08:05 396631                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f23cf514000-7f23cf515000 rw-p 00000000 00:00 0
7ffeb1275000-7ffeb1296000 rw-p 00000000 00:00 0                          [stack]
7ffeb134f000-7ffeb1353000 r--p 00000000 00:00 0                          [vvar]
7ffeb1353000-7ffeb1355000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

Hook *readdir()*

Hidden Shared Objects

(Enumeration Methods)

Process Mappings

The 'proc/<pid>/maps' is the pseudo-filesystem representation of a process' memory mappings, this includes it's loaded SOs

link_map

GOT[1] / DT_DEBUG Contains the address of the *link_map* structure linked list, containing the base address & name of loaded SO's

DT_NEEDED

.Dynamic Section DT_NEEDED entry type contains names of SOs to load at runtime via standard search order mechanisms.

Hidden Shared Objects

(Rules)

1. SOs that only appear in either the *link_map* OR *proc/<pid>/maps* but not both!
2. SOs with the same name but different base addresses in *proc/<pid>/maps* & the *link_map*.
3. DT_NEEDED entries missing from either the *link_map* or *proc/<pid>/maps*.
4. Shared objects not backed on disk.
5. SOs with non-standard paths.


```
"module_cross_references": [
    {
        "base_addr": 140493334355968,
        "disk_backed": false,
        "in_dt_needed_list": false,
        "in_link_maps_list": true,
        "in_proc_maps_list": false,
        "module_path": "/home/dev/attack_data/libcallback.so"
    }
],
```

Cheat Sheet

DT_NEEDED Overwrites	DT_NEEDED Insertions	Preloading abuse	Search order manipulation
Non-sequential DT_NEEDED entries.	Dynamic string table manipulation.	Suspicious use of LD_PRELOAD & ld.so.preload file.	Directories specified in /etc/ld.so.conf.d/*.conf or LD_CONFIG env var.
Missing DT_NULL/DT_DEBUG.	DT_NEEDED name pointing outside the dynamic string table.	Hooking of common functions in LIBC by preloaded SOs.	Custom LD_LIBRARY_PATH, LD_RUN_PATH, env vars.
	Relocated program headers. (Not at 52/64 byte offsets).		Custom DT_RPATH/DT_RUNPATH Dynamic section entries.
			Non-standard program interpreter pointed to by PT_INTERP.

__libc_dlopen_mode()	Reflective SO Injection	Hidden SOs
Uprobe monitoring direct use of __libc_dlopen_mode(), specifying path outside /lib or RTLD_LAZY flags.	Kprobe monitoring real-time anonymous memory allocations with executable permissions.	Shared objects that only appear in either the link_map or proc/<pid>/maps but not both. Or have different base addresses but the same name.
__libc_dlopen_mode string in .rodata	Scanning targeted memory regions for executable headers.	DT_NEEDED entries that don't appear in either the link_map or proc/<pid>/maps.
__libc_dlopen_mode in GOT		Shared object not backed on disk. Or located in non-standard paths.
GDB being used to resolve __libc_dlopen_mode().		

06 KEY TAKEAWAYS

Key Takeaways

1.

Less spotlight on the Linux threat landscape leading to lower detection maturity when compared to Windows

Telemetry & tooling needs to be kept up to date otherwise simple modifications can sidestep existing rules.

2.

3.

Utilizing K/Uprobes as targeted triggers can greatly reduce performance overheads when running memory scanners, opening up their applicable use cases.

THANK YOU

JanielDary