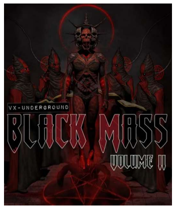
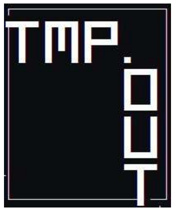
ELF Binaries: One Algorithm to Infect Them All

ABOUT ME

- Pentest and Offensive Security R&D
- Contributor to VX-Underground Black Mass Volume 2 and tmp.0ut Volume 2 hacker journals.





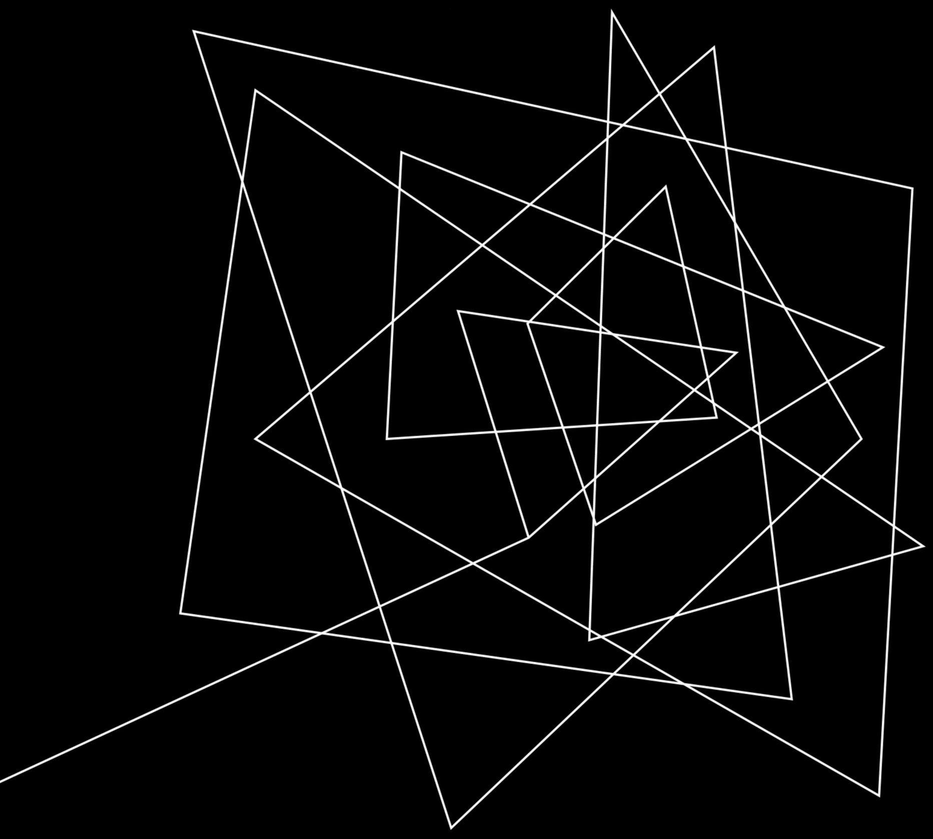
Twitter:@sad0pr
Email: sad0p@protonmail.com
Github: github.com/sad0p
Discord:malcomvx

AGENDA

- Background information on ELF infection and ELF format.
- Infection algorithms.
- Relative Relocation Poisoning.
- Demos.

WHAT ARE ELF BINARIES ?

- ELF stands for Executable and Linkable Format.
- Used for holding code and data for executables, libraries/shared objects (.so) object files (.o), kernel modules (.ko) and even the Linux Kernel itself.
- Often embedded in firmware images.
- Supports wide variety of CPU architectures.
- Mostly prevalent in UNIX environments.



WHAT IS ELF BINARY INFECTION ?

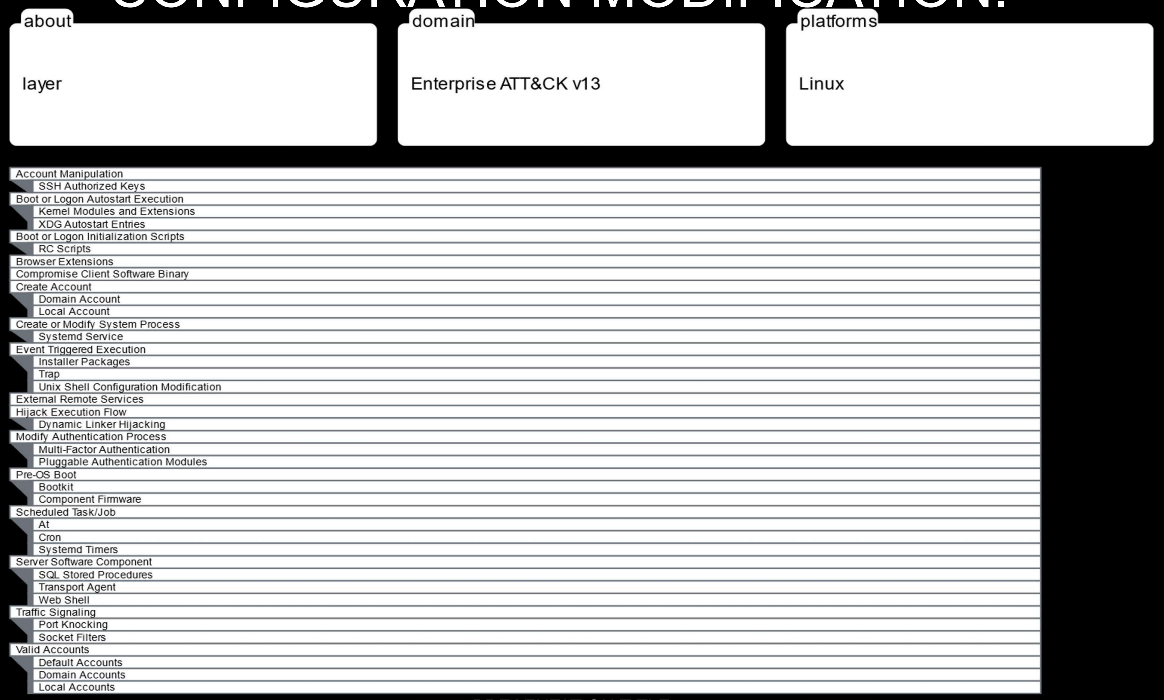
- Conceived by Silvio Cesar and published in his paper "UNIX ELF Parasites and Virus" on Oct, 1998.

- Idea is to patch the target binary and "reroute" execution to the inserted parasite (payload) while conforming to the ELF specification.

- Use infection algorithms to confirm to the ELF specification.

- Not abiding to the ELF specification leads to no-execution or improper execution / crash.

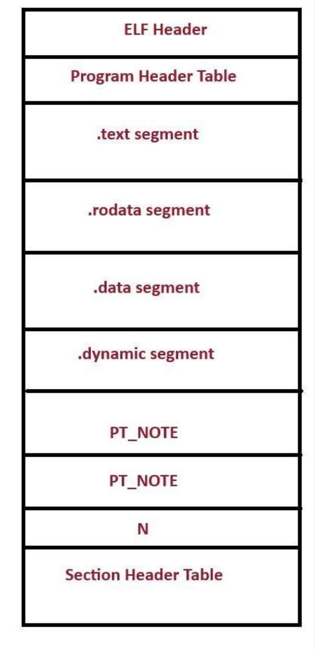
BENEFITS OF ELF BINARY INFECTION FOR PERSISTENCE VERSUS TRADITIONAL PLAINTEXT CONFIGURATION MODIFICATION.



about	domain	platforms
layer	Enterprise ATT&CK v13	Linux

- Account Manipulation
  - SSH Authorized Keys
- Boot or Logon Autostart Execution
  - Kernel Modules and Extensions
  - XDG Autostart Entries
- Boot or Logon Initialization Scripts
  - RC Scripts
- Browser Extensions
- Compromise Client Software Binary
- Create Account
  - Domain Account
  - Local Account
- Create or Modify System Process
  - Systemd Service
- Event Triggered Execution
  - Installer Packages
  - Trap
  - Unix Shell Configuration Modification
- External Remote Services
- Hijack Execution Flow
  - Dynamic Linker Hijacking
- Modify Authentication Process
  - Multi-Factor Authentication
  - Pluggable Authentication Modules
- Pre-OS Boot
  - Bootkit
  - Component Firmware
- Scheduled Task/Job
  - At
  - Cron
  - Systemd Timers
- Server Software Component
  - SQL Stored Procedures
  - Transport Agent
  - Web Shell
- Traffic Signaling
  - Port Knocking
  - Socket Filters
- Valid Accounts
  - Default Accounts
  - Domain Accounts
  - Local Accounts

ELF BINARY FORMAT OVERVIEW (ELF HEADER)



ELF Header

Program Header Table

.text segment

.rodata segment

.data segment

.dynamic segment

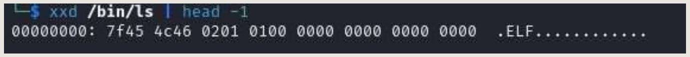
PT_NOTE

PT_NOTE

N

Section Header Table

- ELF Header – First 4 bytes contains ELF magic



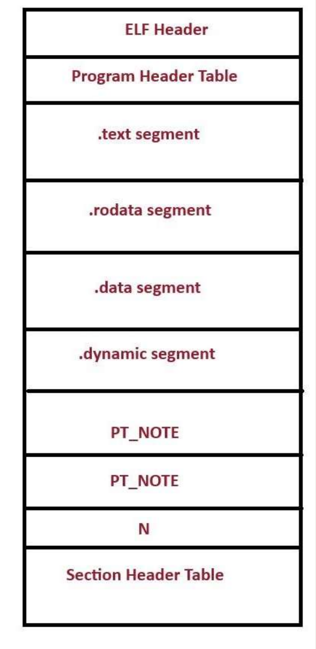
```
└─$ xxd /bin/ls | head -1
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
```

- Stores the program's entry-point, tells the kernel where code execution in the binary should start (nice target for infection algorithms).

- Contains file offsets for finding structures in the ELF binary, such as the Program Header Table and Section Header Table and how many entries are in those structures.

- Other data specifying architecture requirements.

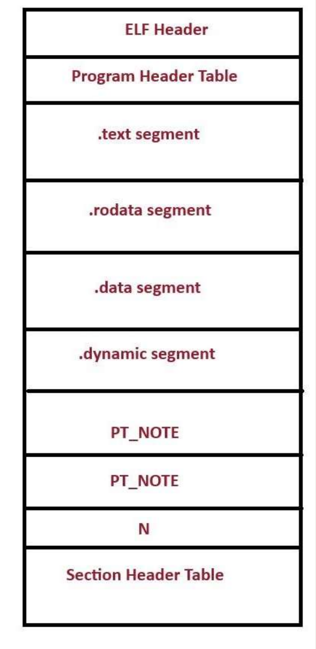
ELF BINARY FORMAT OVERVIEW CONTINUED (PROGRAM HEADER TABLE).



- An array of entries describing segments in the binary.
- Each program header entry, contains fields for file offsets, virtual address, memory permission, etc.
- We can locate executable segments to insert code or create new segments entries to store code with the appropriate infection algorithms.

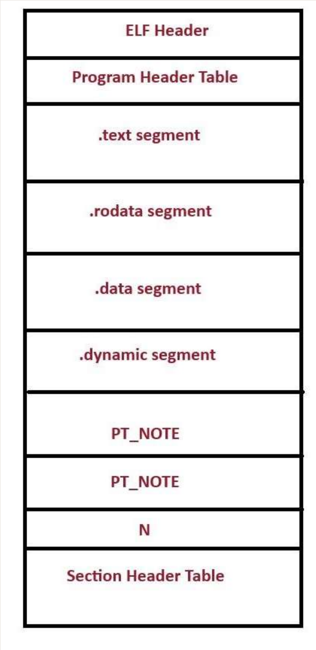
- Must be careful to stay within the ELF specification, segments are used for program loading and the kernel strictly adheres to the ELF specification.

ELF BINARY FORMAT OVERVIEW CONTINUED(SEGMENTS).



- .text segment – R-X perm, contains executable code.
- .rodata segment – READ ONLY perm, used for the storage of string literals.
- .data segment – Has R-W permissions. Used for the storage of global variables.
- .dynamic segment – Has R-W perm. Present in dynamically linked binaries and utilized by the dynamic linker (ld-linux.so).
- PT_NOTE segment – Contains vendor specific information to help the operate system determine if it can load or run the ELF binary (think BSD ELF on Linux vice versa).

ELF BINARY FORMAT OVERVIEW CONTINUED(SECTION HEADER TABLE).



ELF Header

Program Header Table

.text segment

.rodata segment

.data segment

.dynamic segment

PT_NOTE

PT_NOTE

N

Section Header Table

- Often confused with segments, they are not the same.
- Not important for program execution, but vital in linking ELF objects (ET_REL) during the compilation and linking process.
- Used by binary analysis tools such as debuggers, disassemblers and decompilers (vector for obfuscation).
- Describe areas within a given segment, .init, .fini and .text sections are all within the text segment.
- GCC & GNU Linker by default place section header table at the end of the binary (we can abuse this for embedding payloads).

ELF BINARY INFECTION ALGORITHMS

- Allows for insertion of code into the binary while adhering to ELF specifications.
- Enables execution to flow to our code (insertion is not enough) by modifying the OEP (Original Entry Point), function pointers, relocation records, function trampolines and many more.
- Keeps original binary functionality seemingly intact (user or other applications are unaware of any modification), no crashes.
- Text Segment Padding, Reverse Text Segment Padding, PT_NOTE to PT_LOAD, and Data Segment Infection (maybe embedded, NX-bit changed the game), DT_NEEDED (permanent LD_PRELOAD) and Relocation Poisoning / Hijacking are notable file (on disk) infection algorithms (there are others).
- Memory infection techniques (very stealthy) also exists.

ELF BINARY INFECTION ALGORITHMS
(TEXT SEGMENT PADDING)

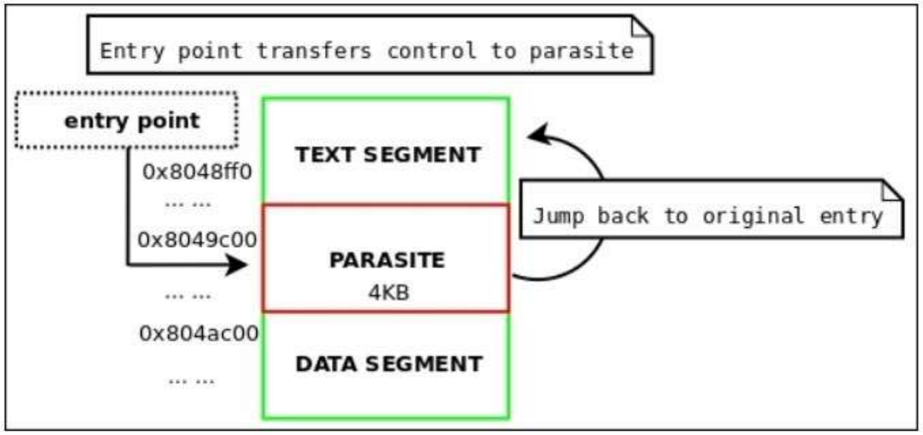


Diagram from *Linux Binary Analysis* by *Ryan Oneil* (elfmaster)

- Takes advantage of the fact there will be a page of memory (4096 bytes) between the text and data segment in memory.
- The additional space can be used to host a parasite/payload.
- 64-bit systems are capable of larger pages and a theoretical 0x200000 (2MB) infection is possible when configured for HUGE PAGES.
- Systems hosting databases might have this enabled (Linux Kernel HugePages configuration), but in X86_64 Linux Kernel still defaults to 32-bit page size (4096 bytes).
- Note, 4096 bytes is the maximum potential space, in most cases there has been less space available.

ELF BINARY INFECTION ALGORITHMS
(PT_NOTE TO PT_LOAD)

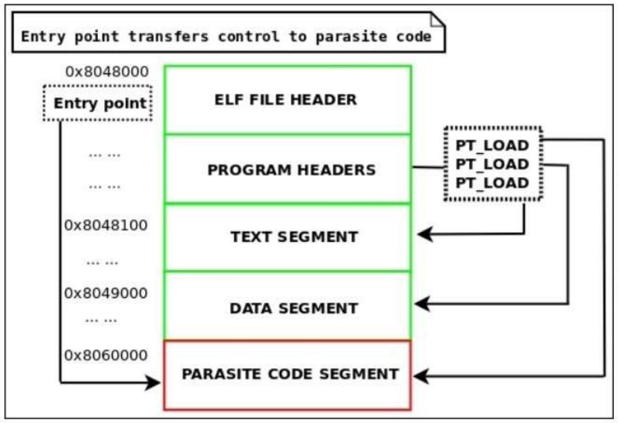


Diagram from *Linux Binary Analysis* by *Ryan Oneil* (elfmaster)

- Converts PT_NOTE segment to a PT_LOAD segment to store our parasite.
- Segments with PT_LOAD are essential for loading, so the ELF loader (kernel) will automatically try to load this segment.
- No space constraints.
- Easy to implement and very popular.
- Also EASY to detect, there will be two segments of type PT_LOAD with executable permission (screams malware to an informed analyst).

Brief Overview: Relocations

- Relocations (general def): Process of patching binaries either at link time or at runtime to properly link references with definitions when they “move”.

- Architecture dependent: Each architecture will have their own relocation types.

Relative Relocations

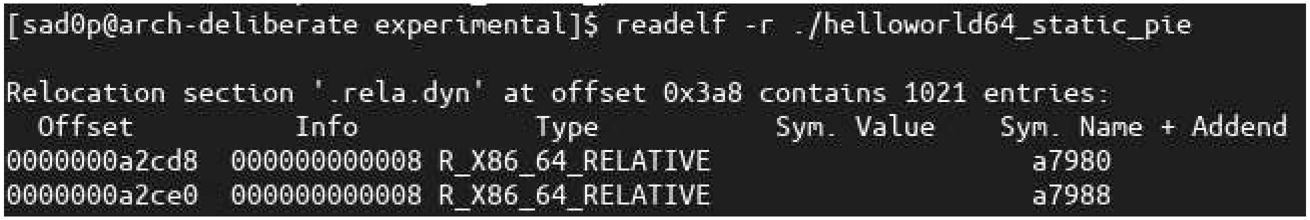
- Relative Relocations: Converts offsets to absolute addresses at runtime.

- Used in Position Independent Executables (ET_DYN) and is important for making binaries ASLR compatible.

- Dynamic and static-pie binaries are of type ET_DYN.

- R_X86_64_RELATIVE and R_X86_RELATIVE.

- Relocation records readily viewable with *readelf* utility:



```
[sad0p@arch-deliberate experimental]$ readelf -r ./helloworld64_static_pie

Relocation section '.rela.dyn' at offset 0x3a8 contains 1021 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
0000000a2cd8  000000000008 R_X86_64_RELATIVE                    a7980
0000000a2ce0  000000000008 R_X86_64_RELATIVE                    a7988
```

Relative Relocations Applications

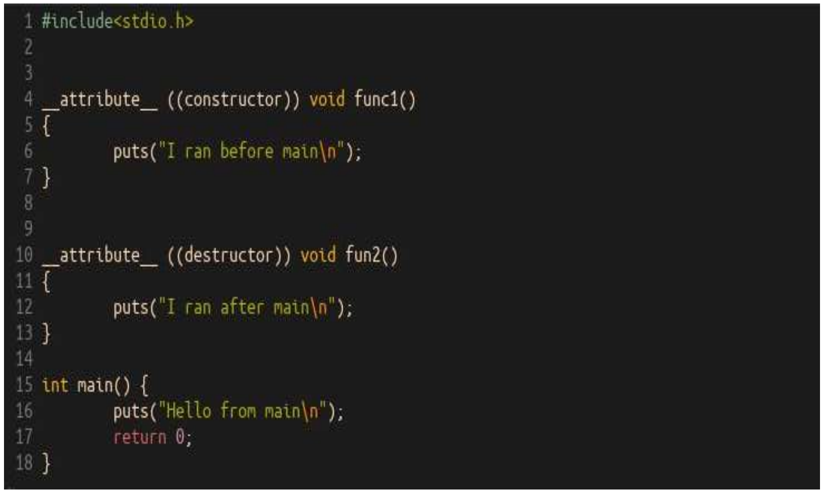
- Used for implementing constructor and destructor routines.

- We focus on their usage in C programs to execute functions before and after main().

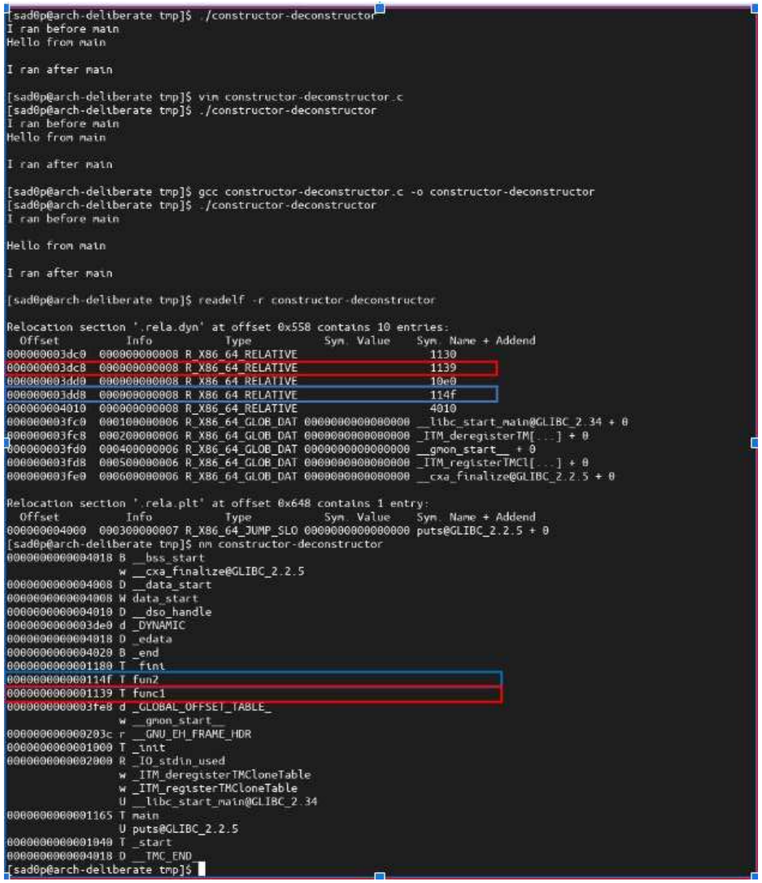
- .init_array and .fini_array section of function ptrs.

- In the context of shared objects (.so) these can be thought of as initialization routines that executes when the library is mapped into the executable address space.

Relative Relocations



```c
#include<stdio.h>


__attribute__ ((constructor)) void func1()
{
        puts("I ran before main\n");
}


__attribute__ ((destructor)) void fun2()
{
        puts("I ran after main\n");
}

int main() {
        puts("Hello from main\n");
        return 0;
}
```



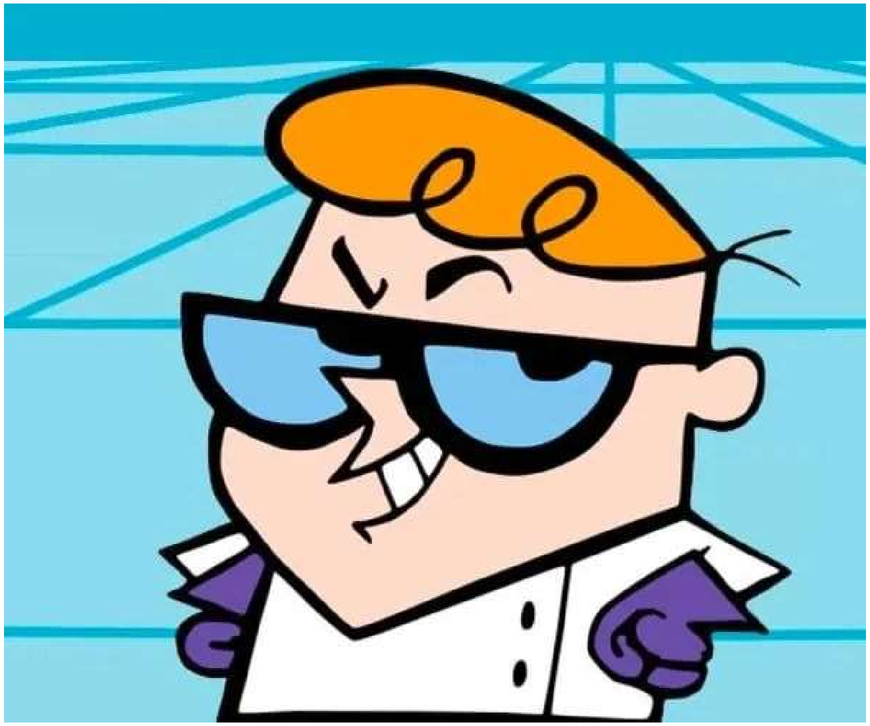
```
[sad0p@arch-deliberate tmp]$ ./constructor-deconstructor
I ran before main
Hello from main

I ran after main

[sad0p@arch-deliberate tmp]$ vim constructor-deconstructor.c
[sad0p@arch-deliberate tmp]$ ./constructor-deconstructor
I ran before main
Hello from main

I ran after main

[sad0p@arch-deliberate tmp]$ gcc constructor-deconstructor.c -o constructor-deconstructor
[sad0p@arch-deliberate tmp]$ ./constructor-deconstructor
I ran before main

Hello from main

I ran after main

[sad0p@arch-deliberate tmp]$ readelf -r constructor-deconstructor

Relocation section '.rela.dyn' at offset 0x558 contains 10 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000003dc0  000000000008 R_X86_64_RELATIVE                    1130
000000003dc8  000000000008 R_X86_64_RELATIVE                    1139
000000003dd0  000000000008 R_X86_64_RELATIVE                    10e0
000000003dd8  000000000008 R_X86_64_RELATIVE                    114f
000000004010  000000000008 R_X86_64_RELATIVE                    4010
000000003fc0  000100000006 R_X86_64_GLOB_DAT 0000000000000000 __libc_start_main@GLIBC_2.34 + 0
000000003fc8  000200000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_deregisterTM[...] + 0
000000003fd0  000400000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
000000003fd8  000500000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_registerTMCl[...] + 0
000000003fe0  000600000006 R_X86_64_GLOB_DAT 0000000000000000 __cxa_finalize@GLIBC_2.2.5 + 0

Relocation section '.rela.plt' at offset 0x648 contains 1 entry:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000004000  000300000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
[sad0p@arch-deliberate tmp]$ nm constructor-deconstructor
0000000000004018 B __bss_start
                 w __cxa_finalize@GLIBC_2.2.5
0000000000004008 D __data_start
0000000000004008 W data_start
0000000000004010 D __dso_handle
0000000000003de0 d _DYNAMIC
0000000000004018 D _edata
0000000000004020 B _end
0000000000001180 T _fini
000000000000114f T fun2
0000000000001139 T func1
0000000000003fe8 d _GLOBAL_OFFSET_TABLE_
                 w __gmon_start__
000000000000203c r __GNU_EH_FRAME_HDR
0000000000001000 T _init
0000000000002000 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
                 U __libc_start_main@GLIBC_2.34
0000000000001165 T main
                 U puts@GLIBC_2.2.5
0000000000001040 T _start
0000000000004018 D __TMC_END__
[sad0p@arch-deliberate tmp]$
```

Infecting Shared Objects (.so)

- Text Segment Padding and PT_NOTE were traditionally used for executable ELF binaries.
- Executable ELF binaries can be defined as those without an entry-point (like most shared objects).
- We can re-think or re-define “Infection Algorithm”, code insertion to abide by the ELF specification as one thing, achieving code execution is another.
- Abusing Relative Relocation records to get the dynamic-linker to assist in code execution now widens the “attack surface” (potential binaries we can infect).

Demo - To the lab !!!

- Anansi - A real life computer virus I wrote.
- D0zer - An ELF binary injector, capable of arbitrary payload injection into ELF binaries, useful for persistence.



Resources

- github.com/sad0p/d0zer
- github.com/sad0p/elfdoc
- hxxp://tmp.0ut
- github.com/elfmaster
- hxxps://a.co/d/jfynVjc (Linux Binary Analysis Amazon link)
- hxxps://tinyurl.com/38fmtrr7 (Black Mass Volume 2 by VXUG)
- hxxps://tinyurl.com/3bdf3w84 (Silvio Cesar Unix Parasites and Virus)
- hxxps://refspecs.linuxfoundation.org/elf/elf.pdf
- hxxps://github.com/tmpout/awesome-elf