Dynamic Analysis

Objectives

- Explore the concept of dynamic libraries and how they are used to implement shared functionality.
- Discuss how obfuscation may complicate reverse engineering; and introduce movcc as an example of an obfuscation.
- Examine tracing the dynamic libraries of an obfuscated programming to understand the programs intent.

References

- https://github.com/xoreaxeaxeax/movfuscator
- https://man7.org/linux/man-pages/man1/ltrace.1.html
- https://attack.mitre.org/techniques/T1027/

Binary Obfuscation

Adversaries may attempt to make an executable or file difficult to discover or analyze by encrypting, encoding, or otherwise obfuscating its contents on the system or in transit. This is common behavior that can be used across different platforms and the network to evade defenses.

- Binary Padding
- Software Packing
- Steganography
- Use of advanced transformations to obscure source

Text copied from: https://attack.mitre.org/techniques/T1027/

Binary Obfuscation Example: Anita Borg

```
0x08051222 <+31784>: mov    DWORD PTR ds:0x83fcd54,0x85fcd8c
0x0805122c <+31794>: mov    eax,DWORD PTR [ecx*4+0x83fcd50]
0x08051233 <+31801>: mov    edx,DWORD PTR ds:0x8053c40
0x08051239 <+31807>: mov    DWORD PTR [eax],edx
0x0805123b <+31809>: mov    edx,DWORD PTR ds:0x8053c44
0x08051241 <+31815>: mov    DWORD PTR [eax+0x4],edx
0x08051244 <+31818>: mov    edx,DWORD PTR ds:0x8053c48
0x0805124a <+31824>: mov    DWORD PTR [eax+0x8],edx
0x0805124d <+31827>: mov    edx,DWORD PTR ds:0x8053c4c
0x08051253 <+31833>: mov    DWORD PTR [eax+0xc],edx
0x08051256 <+31836>: mov    edx,DWORD PTR ds:0x8053c50
0x0805125c <+31842>: mov    DWORD PTR [eax+0x10],edx
0x0805125f <+31845>: mov    edx,DWORD PTR ds:0x8053c54
0x08051265 <+31851>: mov    DWORD PTR [eax+0x14],edx
0x08051268 <+31854>: mov    eax,ds:0x83fcd38
0x0805126d <+31859>: mov    eax,DWORD PTR [eax*4+0x83fcd30]
0x08051274 <+31866>: mov    DWORD PTR [eax],0x0
```

anita-borg was a purposely obfuscated challenge we wrote for a cybersecurity competition. The program is compiled with movcc so every single instruction in the program is a mov instruction, making it very hard to understand whats going on.

Dynamic Libraries

```
ldd ./fLag_TRACE
        linux-gate.so.1 (0xe8375000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xe8126000)
        libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0xe8021000)
        /lib/ld-linux.so.2 (0xe8377000)
```

However, the program relies on the standard C library (aka libc) to implement some functionality. This is code that we didn't write but is standard to the C language. For example, we didn't write the printf() function. We just call it from libc.

Ltrace: Library Tracing

ltrace is a debugging utility in Linux, used to display the calls a userspace application makes to shared libraries. It does this by hooking into the dynamic loading system, allowing it to insert shims which display the parameters which the applications uses when making the call, and the return value which the library call reports. ltrace can also trace Linux system calls. Because it uses the dynamic library hooking mechanism, ltrace cannot trace calls to libraries which are statically linked directly to the target binary.

```
$man ltrace

NAME
       ltrace - A library call tracer

DESCRIPTION

       ltrace is a program that simply runs the specified command until
       it exits.  It intercepts and records the dynamic library calls
       which are called by the executed process and the signals which
       are received by that process.  It can also intercept and print
       the system calls executed by the program.
```

Text copied from ltrace manpage and https://en.wikipedia.org/wiki/Ltrace

Example: Anita Borg

```
pwndbg> got
Filtering out read-only entries (display them with -r or --show-readonly)

State of the GOT of /root/gencyber/re/examples/cyber-heroines/anita-borg/fLag_TRACE:
GOT protection: Partial RELRO | Found 6 GOT entries passing the filter
[0x8053000] read@GLIBC_2.0 -> 0x8049016 (read@plt+6) ◂— push 0 /* 'h' */
[0x8053004] sigaction@GLIBC_2.0 -> 0xf75376e0 (sigaction) ◂— push esi
[0x8053008] fflush@GLIBC_2.0 -> 0x8049036 (fflush@plt+6) ◂— push 0x10
[0x805300c] printf@GLIBC_2.0 -> 0x8049046 (printf@plt+6) ◂— push 0x18
[0x8053010] strncmp@GLIBC_2.0 -> 0x8049056 (strncmp@plt+6) ◂— push 0x20 /* 'h ' */
[0x8053014] exit@GLIBC_2.0 -> 0x8049066 (exit@plt+6) ◂— push 0x28 /* 'h(' */
```

In fact, the program reiles on the additional external functions: read() sigaction(), fflush(), strncmp(), exit().

Example: Anita Borg

```
strcmp(3)
Library Functions Manual
strcmp(3)

NAME
       strcmp, strncmp - compare two strings

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <string.h>

       int strcmp(const char *s1, const char *s2);
       int strncmp(const char s1[.n], const char s2[.n], size_t n);

DESCRIPTION
       The strcmp() function compares the two strings s1 and s2.  The locale is not taken
into account (for a locale-aware comparison, see strcoll(3)).  The comparison is done using
unsigned characters.
```

If we do not know the purpose of a function, we can always ask man to reveal the purpose of the function.

Example: Anita Borg

```
ltrace ./fLag_TRACE

<...snipped...>

fflush(0 Please submit the flag >>> )
= 0

read(0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
, "AAAAAAAAAAAAAAAAAAAAA", 23)
= 23
--- SIGSEGV (Segmentation fault) ---
strncmp("AAAAAAAAAAAAAAAAAAAAA\363chctf{b_"..., "chctf{b_A_V1s10NArY_2}", 22)
= -1
```

We can use ltrace to trace all the C library our challenge makes. Notice how it makes the call to strncmp(), a function used to compare two strings. The first “AAAAA….” is a test value we entered. The second looks a lot like the expected flag for the challenge!

Example: BigBrother re-2

```
Ltrace ./re-2.bin

<... snipped...>
printf(" Please login >>> ")
= 18
fflush(0 Please login >>> )
= 0
read(0AAAA
, "AAAA\n", 10)
= 5
strncmp("AAAA\n\274|\3550\ny\355B1gBr0th3R", "B1gBr0th3R", 9)
= -1
printf("<<< Voice your gratitude to Big "...)
= 65
exit(1<<< Voice your gratitude to Big Brother for this new, happy life. <no return ...>
+++ exited (status 1) +++
```

Here is another example, can you identify the password for the challenge?