BINARY ANALYSIS 101

JIM ALVES-FOSS

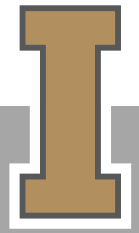
University of Idaho
College of Engineering

RELATED TOPIC

You have discussed forensics. With respect to host-based (not network or media-based):

Why do we do forensics? I am looking for three very broad descriptions of how the computer played a part in the incident being investigated:

1. The computer was the victim of an attack
2. The computer was used as a “weapon” to launch the attack.
3. The computer was used as a tool to store/access digital information related to the attack.

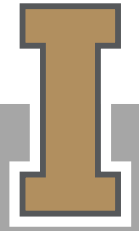


BINARY ANALYSIS

Binary analysis is the act of evaluating an executable program (or set of related programs and libraries) to determine characteristics of that program, usually cybersecurity related.

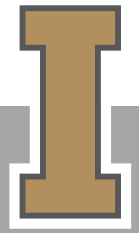
What are three broad categories of sources of binaries we may want to evaluate?

1. The binary is untrusted and may contain malware.
2. The binary is developed in-house and needs to be evaluated for security vulnerabilities.
3. The binary is obtained from third-party and does not contain malware, but will be incororpated into one of our systems and needs to be evaluated for security vulnerabilities.



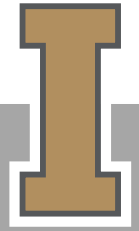
WHERE DO WE USE BINARY ANALYSIS

- Capture the Flag (CTF) competitions
- Bug bounties
- Whitehat (ethical) hacking
- Certification of software
- DoD Hard Problem List:
  - Automate response to attacks (auto scan and patch when attack detected)
  - Automate proactive security (auto scan and patch before attacked)
    - Bonus points if you can automate the attack
    - *See:* DARPA Cyber Grand Challenge (Google it)



MAJOR STEPS IN A BINARY ANALYSIS METHODOLOGY

- Perform binary discovery
- Information gathering
- Static analysis
- Dynamic Analysis
- Iterating each step
- Automatinc methodology tasks
- Adopting the methodology steps.
  - Credit to: M. Born. “Binary Analysis Cookbook” Packt Publishing, 2019
  - https://github.com/packtpublishing/Binary-Analysis-Cookbook



LETS GET STARTED (BINARY DISCOVERY)

Type the following, without the $ or #. (Enter your password when asked)

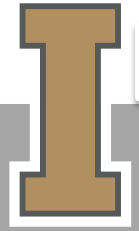
```
$ sudo su
# find / -executable -type f
```

Wait for command to finish, then type. Study the output.

```
# file -i /bin/cat
# file /bin/cat
```

Type the following and review the output. Notice the sort order of the information displayed

```
# ls -alt /bin/
```



BINARY DISCOVERY (2)

When ready, type the following:

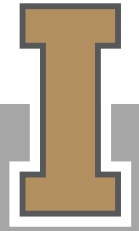
```
# updatedb
# locate 'cat'
```

(may need to install: (`yum install mlocate`)

Go over the output and, when ready, type the following:

```
# ps -ef | more
```

The 'l' is the pipe command and says to show one page at a time. Hit <Spacebar> to go to next page / <Enter> for one line at a time / 'q' to quit, and '?' for help. *Can use the "less" command instead, use 'h' for help*.



BINARY DISCOVERY (3)

Once you're done reviewing the output, type the following (all on one line):

```
# for i in $(find / -executable -type f);do file -i $i | grep -i 'x-
executable; charset=binary';done
```

Now type the following (notice the output)?

```
$ exit
$ file /bin/yumdownloader
```

Type the following and look at the first line

```
$ more /bin/yumdownloader
```

BINARY DISCOVERY (4)

Which program runs?

```
$ which bash
```

Where is it? (If locate gives too much information)

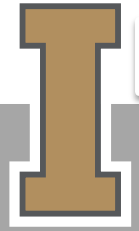
```
$ whereis bash
```

If you want to see which commands and aliases are available to us, you can run the following in a Terminal session:

```
$ compgen -ac
```

Manual pages

```
$ man <tool-name>     such as   man find   or   man file
```

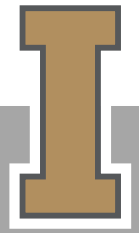


BINARY DISCOVERY (5)

So now that we have an idea of how to search our systems for a potentially malicious binary, let's focus on what we can do to gather as much information about the binary as possible.

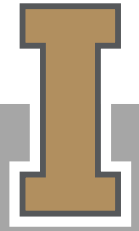
Like penetration testing, this is probably the most important phase of the methodology and will determine whether we set ourselves up for success or not.

Assuming you know the name of the file in question, the types of information we want to gather and the questions we need to answer include the following:



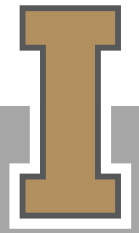
BINARY DISCOVERY (6)

- Is the file executable?
- Is the file a binary?
- For which architecture (x86, or x86_64) is the binary compiled?
- Which format is the binary? (Hopefully ELF, otherwise the rest of this book is going to be pointless.)
- Is the binary stripped of its symbol table?
- Can we identify any useful strings within the binary?
- Is there a running process associated with this binary?



BINARY DISCOVERY (7)

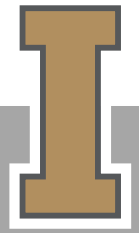
- What's the SHA hash of the binary?
- Does the hash come back as a known malicious file hash?
- What was the original programming language used?
- Can we identify any useful function names?
- Can we identify any libraries used?
- When was the binary written to disk?



BINARY DISCOVERY (8)

For a non-malicious binary, such as an application developed within your organization meant to run on Linux, we can ask similar questions but in a more targeted approach for vulnerability analysis, as opposed to analysis designed to identify malicious functionality:

- Does the application take any input (user or otherwise)?
- Does the application validate all input?
- Does the application safely manage memory?
- Does the application use up-to-date libraries or third-party frameworks?
- How is the application compiled?
- Are there any noticeable strings containing sensitive data such as hardcoded passwords?



GET READY

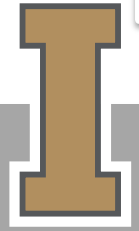
Type the following (if still in a root shell). Why?

```
# exit
```

The type the following (all one line)

```
$ wget https://github.com/PacktPublishing/Binary-Analysis-Cookbook/
archive/refs/heads/master.zip
```

```
$ unzip master.zip
$ cd Binary-Analysis-Cookbook-master/Chapter-04/64bit
```



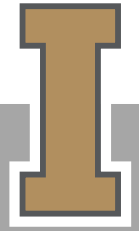
EXAMINING BINARIES

Type the following:

```
$ file ch04-example
```

And for comparison:

```
$ file ../32bit/ch04-example
```



EXAMINING BINARIES (2)

Type the following (remember you can always add ‘ l more ‘ to the end of the line):

```
$ strings ch04-example
```

When you have finished reviewing strings, type each of the following and review the output

```
$ readelf –h ch04-example
```

```
$ readelf –l -W ch04-example
```

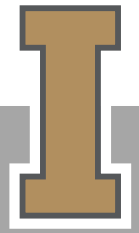
```
$ readelf –S -W ch04-example
```

BONUS QUESTION

Run the program.

Guess the password.

What is the secret messages.



EXAMINING BINARIES (3)

Type each of the following and review the output

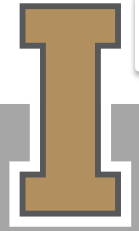
```
$ readelf –s -W ch04-example
```

```
$ readelf –p .text ch04-example
```

```
$ readelf –x .text -W ch04-example
```

```
$ readelf –R .text -W ch04-example
```

```
$ readelf –p .strtab -W ch04-example
```



EXAMINING BINARIES (4)

Type each of the following and review the output

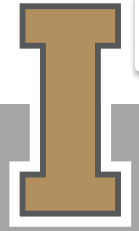
```
$ objdump –f ch04-example
```

```
$ objdump –j -.text -s ch04-example
```

```
$ objdump –x ch04-example
```

```
$ ldd –v ch04-example
$ ldd –v /usr/bin/cp
```

```
$ hexdump –C ch04-example | more
```



STATIC ANALYSIS

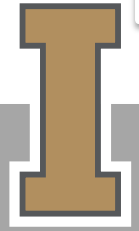
Type each of the following and review the output:

```
$ ndisasm –a –p intel ch04-example | more
```

May need to install (yum install nasm)

```
$ objdump –d ch04-example > obj.out
$ more obj.out
```


```
$ objdump –d ch04-example > obj.out
$ more obj.out
```

STATIC ANALYSIS (2)

```
#include<stdio.h>

int foo(int i) {
  int k;
  int res =0;
  char buf[20];
  for (k=1;k<10;k++){
     res=res+i;
  }
  printf("Enter your name :");
  fscanf(stdin,"%s",buf);
  printf("Hello %s\n",buf);
  return res;
}
```

```
int main() {
   int k;
   k=foo(2);
   printf("Sum = %d\n",k);
   return 0;
}
```

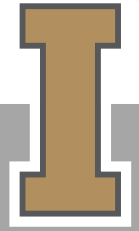
LETS COMPILE

We will compile with each of the following options, and then look at the code:

```
$ gcc tmp1.c -o tmp1
$ objdump -dj .text tmp1 > obj.normal

$ gcc -fstack-protector tmp1.c -o tmp1
$ objdump -dj .text tmp1 > obj.stackprotector

$ gcc -O -D_FORTIFY_SOURCE=2 tmp1.c -o tmp1
$ objdump -dj .text tmp1 > obj.fortify
```



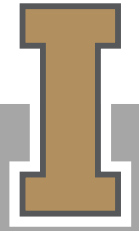
EXAMINE THE OUTPUTS (NORMAL)

```
0000000000400577 <foo>:
  400577:  55                    push   %rbp
  400578:  48 89 e5              mov    %rsp,%rbp
  40057b:  48 83 ec 30           sub    $0x30,%rsp
  40057f:  89 7d dc              mov    %edi,-0x24(%rbp)
...
  4005ce:  48 8d 45 e0           lea    -0x20(%rbp),%rax
  4005d2:  48 89 c6              mov    %rax,%rsi
  4005d5:  bf b5 06 40 00        mov    $0x4006b5,%edi
  4005da:  b8 00 00 00 00        mov    $0x0,%eax
  4005df:  e8 bc fe ff ff        callq  4004a0 <printf@plt>
  4005e4:  8b 45 f8              mov    -0x8(%rbp),%eax
  4005e7:  c9                    leaveq
  4005e8:  c3                    retq
```

Good additional references on reverse engineering: https://nsa-codebreaker.org/resources

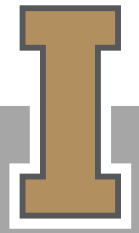
EXAMINE THE OUTPUTS (STACK)

```
00000000004005e7 <foo>:
  4005e7:    55                       push   %rbp
  4005e8:    48 89 e5                 mov    %rsp,%rbp
  4005eb:    48 83 ec 40              sub    $0x40,%rsp
  4005ef:    89 7d cc                 mov    %edi,-0x34(%rbp)
  4005f2:    64 48 8b 04 25 28 00     mov    %fs:0x28,%rax
  4005f9:    00 00
  4005fb:    48 89 45 f8              mov    %rax,-0x8(%rbp)
...
  40065e:    e8 ad fe ff ff           callq  400510 <printf@plt>
  400663:    8b 45 dc                 mov    -0x24(%rbp),%eax
  400666:    48 8b 4d f8              mov    -0x8(%rbp),%rcx
  40066a:    64 48 33 0c 25 28 00     xor    %fs:0x28,%rcx
  400671:    00 00
  400673:    74 05                    je     40067a <foo+0x93>
  400675:    e8 86 fe ff ff           callq  400500 <__stack_chk_fail@plt>
  40067a:    c9                       leaveq
  40067b:    c3                       retq
```



EXAMINE THE OUTPUTS (FORTIFY)

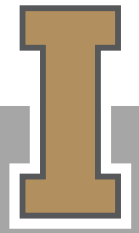
```
400597:    53                       push   %rbx
  400598:  48 83 ec 20              sub    $0x20,%rsp
  40059c:  89 fb                    mov    %edi,%ebx
...
  4005cb:  48 89 e2                 mov    %rsp,%rdx
  4005ce:  be b5 06 40 00           mov    $0x4006b5,%esi
  4005d3:  bf 01 00 00 00           mov    $0x1,%edi
  4005d8:  b8 00 00 00 00           mov    $0x0,%eax
  4005dd:  e8 de fe ff ff           callq  4004c0 <__printf_chk@plt>
  4005e2:  8d 04 db                 lea    (%rbx,%rbx,8),%eax
  4005e5:  48 83 c4 20              add    $0x20,%rsp
  4005e9:  5b                       pop    %rbx
  4005ea:  c3                       retq
```



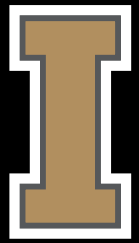
OTHER COMMANDS TO TRY

There are other tools out there for examining/modifying binaries, many in the binutils package, including, but not limited to

- `nm`
- `objcopy`
- `strace/ltrace`
- `dd`
- `gdb`



PLEASE ATTRIBUTE DR. JIM ALVES-FOSS AND DR. JIA SONG, UNIVERSITY OF IDAHO



University of Idaho
College of Engineering