Find SW vulnerability

via automatic method

WHO AM I

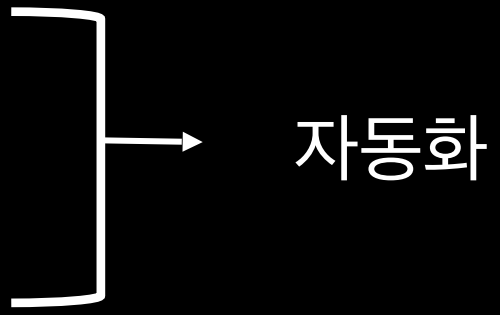
- 정재영 - JJY
- BOB 8기 취약점 분석 트랙
- Reverselab - Pwner
- Bug hunting
  - CVE-2020-12830
  - CVE-2020-7860
  - Lots of KVEs

How to find Bugs?

- Fuzzing
- IDA python
- Monitoring tools

How to find Bugs?

- Fuzzing
- IDA python
- Monitoring tools



적게 일하고 많이 벌자!!

Fuzzing

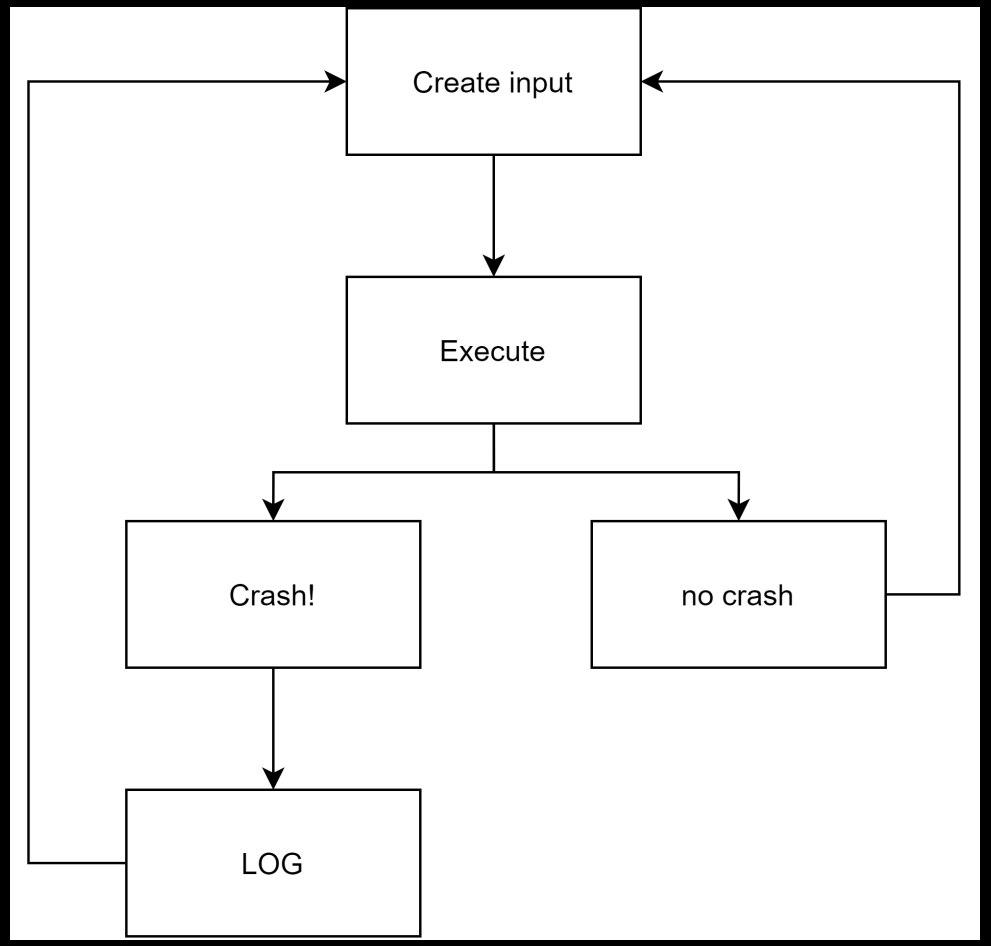
- Fuzzer
  - 랜덤한 인풋을 생성하여 프로그램의 버그를 찾는 도구

Fuzzing

- Black box fuzzer
  - BFF
- Coverage based fuzzer
  - AFL

BFF

- Basic Fuzzing Framework
- Black box fuzzer
- 시드 제공 + 간단한 설정 파일 설정



BFF

- 국내 소프트웨어 32개를 대상으로 퍼저를 (대략 1일 정도) 돌린 결과...

취약점 개수	프로그램 수
0	7
0~10	13
10~100	8
100~150	2
150~200	2

BFF

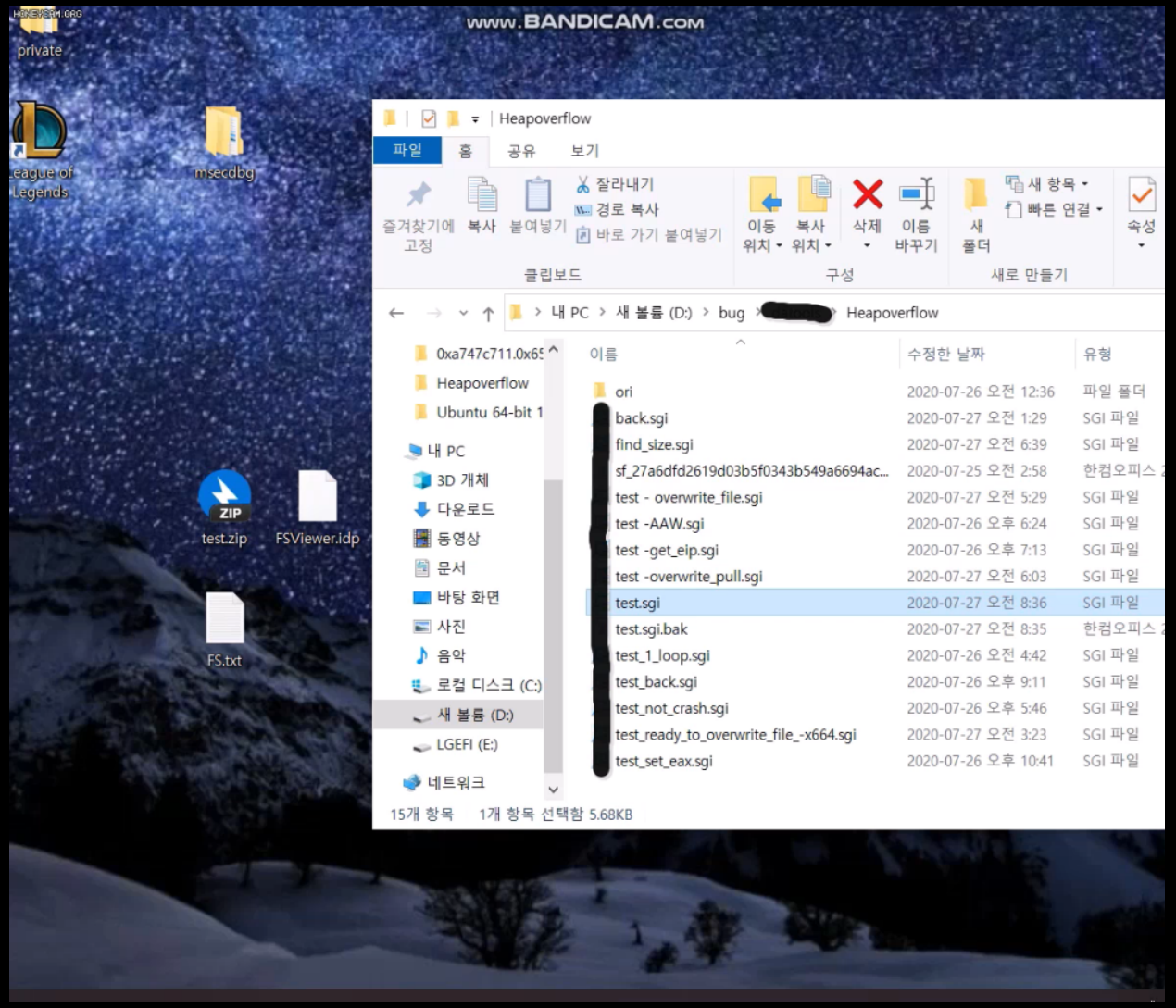
- 국내 소프트웨어 32개를 대상으로 퍼저를 (대략 1일 정도) 돌린 결과...

취약점 개수	프로그램 수
0	7
0~10	13
10~100	8
100~150	2
150~200	2

BFF

- Buffer Overflow, Integer Overflow, Double free ...
- Exploit!
  - Heap Logic
  - SEH overwrite
  - ROP

DEMO


www.BANDICAM.com
private
League of Legends
msecdbg
test.zip
FSViewer.idp
FS.txt
Heapoverflow
파일
홈
공유
보기
잘라내기
경로 복사
바로 가기 붙여넣기
즐겨찾기에 고정
복사
붙여넣기
이동 위치
복사 위치
삭제
이름 바꾸기
새 폴더
새 항목
빠른 연결
속성
클립보드
구성
새로 만들기
내 PC › 새 볼륨 (D:) › bug › [redacted] › Heapoverflow
0xa747c711.0x65
Heapoverflow
Ubuntu 64-bit 1
내 PC
3D 개체
다운로드
동영상
문서
바탕 화면
사진
음악
로컬 디스크 (C:)
새 볼륨 (D:)
LGEFI (E:)
네트워크
이름
수정한 날짜
유형
ori 2020-07-26 오전 12:36 파일 폴더
back.sgi 2020-07-27 오전 1:29 SGI 파일
find_size.sgi 2020-07-27 오전 6:39 SGI 파일
sf_27a6dfd2619d03b5f0343b549a6694ac... 2020-07-25 오전 2:58 한컴오피스
test - overwrite_file.sgi 2020-07-27 오전 5:29 SGI 파일
test -AAW.sgi 2020-07-26 오후 6:24 SGI 파일
test -get_eip.sgi 2020-07-26 오후 7:13 SGI 파일
test -overwrite_pull.sgi 2020-07-27 오전 6:03 SGI 파일
test.sgi 2020-07-27 오전 8:36 SGI 파일
test.sgi.bak 2020-07-27 오전 8:35 한컴오피스
test_1_loop.sgi 2020-07-26 오전 4:42 SGI 파일
test_back.sgi 2020-07-26 오후 9:11 SGI 파일
test_not_crash.sgi 2020-07-26 오후 5:46 SGI 파일
test_ready_to_overwrite_file_-x664.sgi 2020-07-27 오전 3:23 SGI 파일
test_set_eax.sgi 2020-07-26 오후 10:41 SGI 파일
15개 항목
1개 항목 선택함 5.68KB

Let’s go to coverage based fuzzing

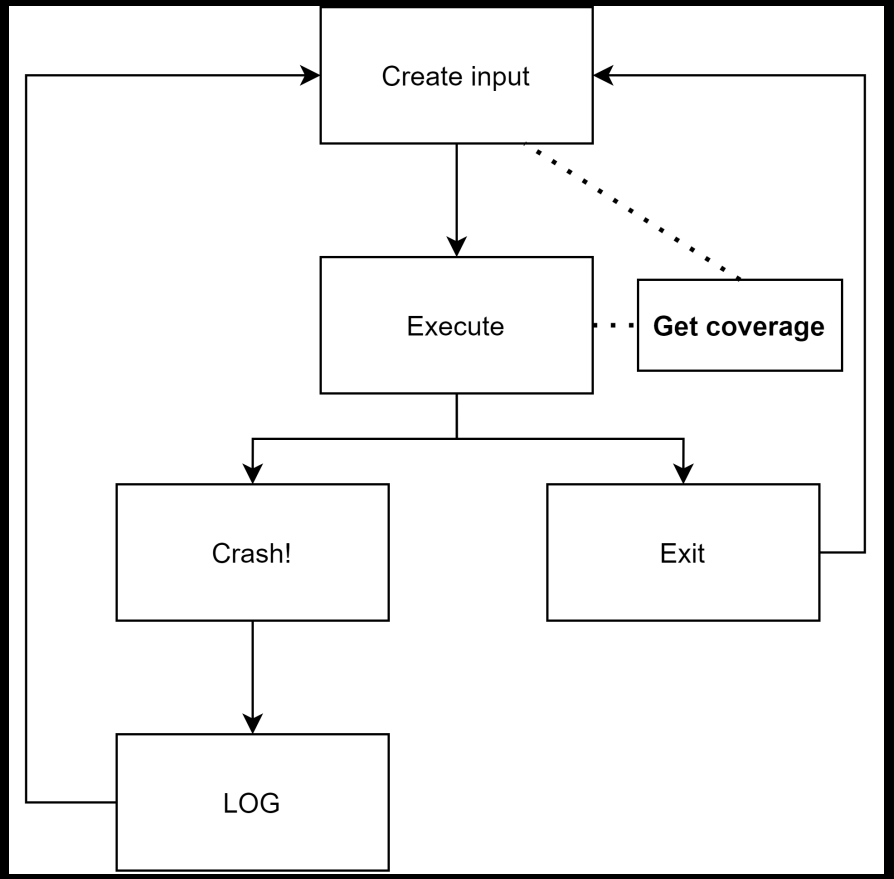
메이저 프로그램의 경우 크래시가 잘 나오지 않음

취약점 개수	프로그램 수
0	7
0~10	13
10~100	8
100~150	2
150~200	2

→ Unexploitable?

Let’s go to coverage based fuzzing

- american fuzzy lop



- Code coverage를 이용하여 Input을 smart하게 생성

Hard ...?

- GUI 프로그램을 coverage based fuzzing하기 힘든 이유
  - 종료 시점이 정해져 있지 않음.
    - 어느 시점에 coverage 측정을 끝내야 하지?

Hard ...?

- GUI 프로그램을 coverage based fuzzing하기 힘든 이유
  - 종료 시점이 정해져 있지 않음.
    - 어느 시점에 coverage 측정을 끝내야 하지?

==> 라이브러리에서 **핵심 함수만 가져와서 실행**하는 harness를 만들면 해결 가능

How to find target function

- 파일 or 패킷을 분석하는 함수.
- 공격자가 인자를 마음대로 제어할 수 있는 함수.
- Entry point에 해당하는 함수

Make harness

- Target 함수를 import해서 실행하는 프로그램 만들기

```c
typedef void (*lib_func)(wchar_t *,int ,int * );
int  main(int argc,char ** argv){
    int test;
    lib_func  func    = NULL;
    void  * handle = dlopen("censored_lib_path",RTLD_NOW | RTLD_GLOBAL);
    func = (lib_func  ) dlsym(handle, "censored_func_name");
    wchar_t * ws1;
    char * st  = (char *)calloc(0x100000,1);
    int fd = open(argv[1],O_RDWR);
    read(fd,st,0xff000);
    func(st,0,&test);
    free(st);
    return 1;
}
```

Load library

Make harness

- Target 함수를 import해서 실행하는 프로그램 만들기

```
typedef void (*lib_func)(wchar_t *,int ,int * );
int  main(int argc,char ** argv){
    int test;
    lib_func  func    = NULL;
    void  * handle = dlopen("censored_lib_path",RTLD_NOW | RTLD_GLOBAL);
    func = (lib_func  ) dlsym(handle, "censored_func_name");
    wchar_t * ws1;
    char * st  = (char *)calloc(0x100000,1);
    int fd = open(argv[1],O_RDWR);
    read(fd,st,0xff000);
    func(st,0,&test);
    free(st);
    return 1;
}
```

Load function

Make harness

- Target 함수를 import해서 실행하는 프로그램 만들기

```
typedef void (*lib_func)(wchar_t *,int ,int * );
int  main(int argc,char ** argv){
    int test;
    lib_func  func    = NULL;
    void  * handle = dlopen("censored_lib_path",RTLD_NOW | RTLD_GLOBAL);
    func = (lib_func  ) dlsym(handle, "censored_func_name");
    wchar_t * ws1;
    char * st  = (char *)calloc(0x100000,1);
    int fd = open(argv[1],O_RDWR);
    read(fd,st,0xff000);
    func(st,0,&test);
    free(st);
    return 1;
}
```

Get input

Make harness

- Target 함수를 import해서 실행하는 프로그램 만들기

```
typedef void (*lib_func)(wchar_t *,int ,int * );
int  main(int argc,char ** argv){
    int test;
    lib_func  func    = NULL;
    void  * handle = dlopen("censored_lib_path",RTLD_NOW | RTLD_GLOBAL);
    func = (lib_func  ) dlsym(handle, "censored_func_name");
    wchar_t * ws1;
    char * st  = (char *)calloc(0x100000,1);
    int fd = open(argv[1],O_RDWR);
    read(fd,st,0xff000);
    func(st,0,&test);
    free(st);
    return 1;
}
```

Execute function

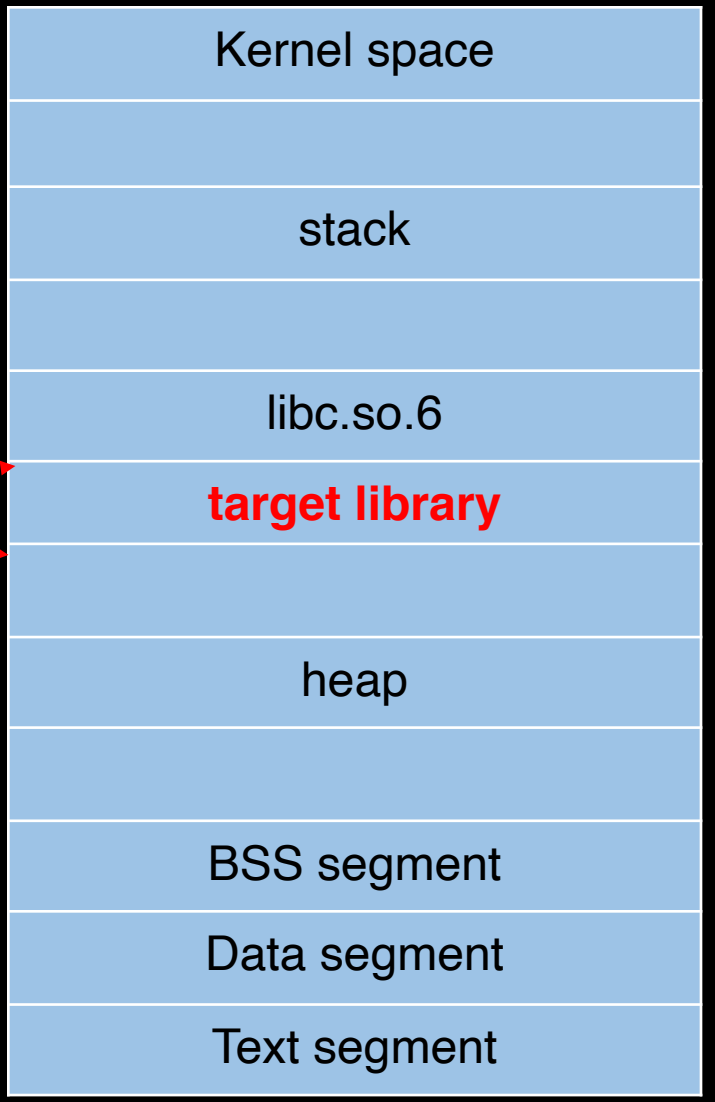
Coverage in AFL qemu mode

- AFL with no source code --> Qemu mode

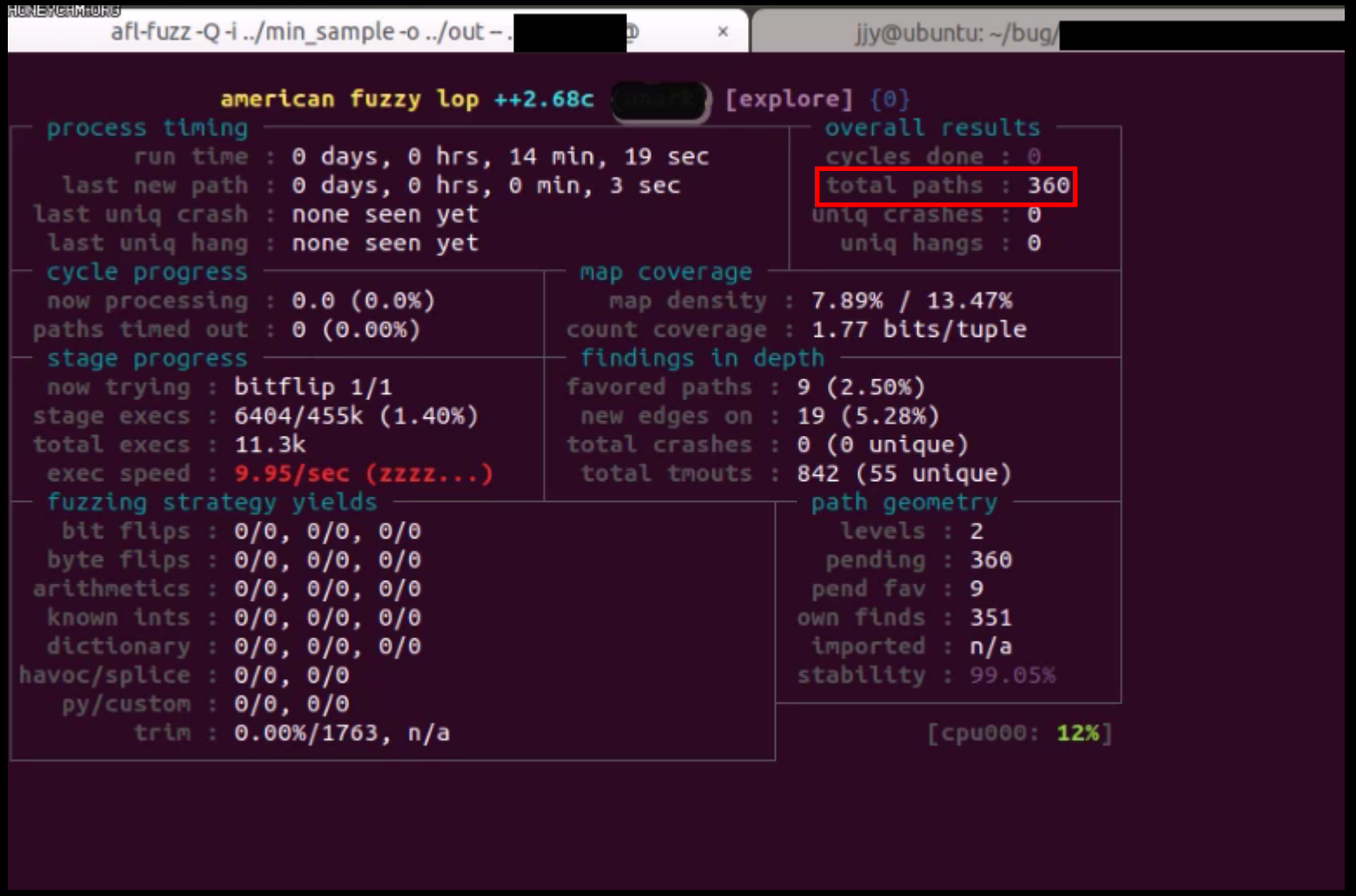
```
static inline void afl_maybe_log(abi_ulong cur_loc) {

  static __thread abi_ulong prev_loc;

  /* Optimize for cur_loc > afl_end_code, which is the most likely case on
     Linux systems. */

  if (cur_loc > afl_end_code || cur_loc < afl_start_code || !afl_area_ptr)
    return;

  /* Looks like QEMU always maps to fixed locations, so ASAN is not a
     concern. Phew. But instruction addresses may be aligned. Let's mangle
     the value to get something quasi-uniform. */

  cur_loc  = (cur_loc >> 4) ^ (cur_loc << 8);
  cur_loc &= MAP_SIZE - 1;

  /* Implement probabilistic instrumentation by looking at scrambled block
     address. This keeps the instrumented locations stable across runs. */

  if (cur_loc >= afl_inst_rms) return;

  afl_area_ptr[cur_loc ^ prev_loc]++;
  prev_loc = cur_loc >> 1;

}
```

Coverage in AFL qemu mode

```
//patches/afl-qemu-cpu-inl.h
if (getenv("AFL_CODE_START"))
    afl_start_code = strtoll(getenv("AFL_CODE_START"), NULL, 16);
if (getenv("AFL_CODE_END"))
    afl_end_code = strtoll(getenv("AFL_CODE_END"), NULL, 16);
```



Gooood!


afl-fuzz -Q -i ../min_sample -o ../out – .
jjy@ubuntu: ~/bug/
american fuzzy lop ++2.68c [explore] {0}
process timing
run time : 0 days, 0 hrs, 14 min, 19 sec
last new path : 0 days, 0 hrs, 0 min, 3 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
overall results
cycles done : 0
total paths : 360
uniq crashes : 0
uniq hangs : 0
cycle progress
now processing : 0.0 (0.0%)
paths timed out : 0 (0.00%)
map coverage
map density : 7.89% / 13.47%
count coverage : 1.77 bits/tuple
stage progress
now trying : bitflip 1/1
stage execs : 6404/455k (1.40%)
total execs : 11.3k
exec speed : 9.95/sec (zzzz...)
findings in depth
favored paths : 9 (2.50%)
new edges on : 19 (5.28%)
total crashes : 0 (0 unique)
total tmouts : 842 (55 unique)
fuzzing strategy yields
bit flips : 0/0, 0/0, 0/0
byte flips : 0/0, 0/0, 0/0
arithmetics : 0/0, 0/0, 0/0
known ints : 0/0, 0/0, 0/0
dictionary : 0/0, 0/0, 0/0
havoc/splice : 0/0, 0/0
py/custom : 0/0, 0/0
trim : 0.00%/1763, n/a
path geometry
levels : 2
pending : 360
pend fav : 9
own finds : 351
imported : n/a
stability : 99.05%
[cpu000: 12%]

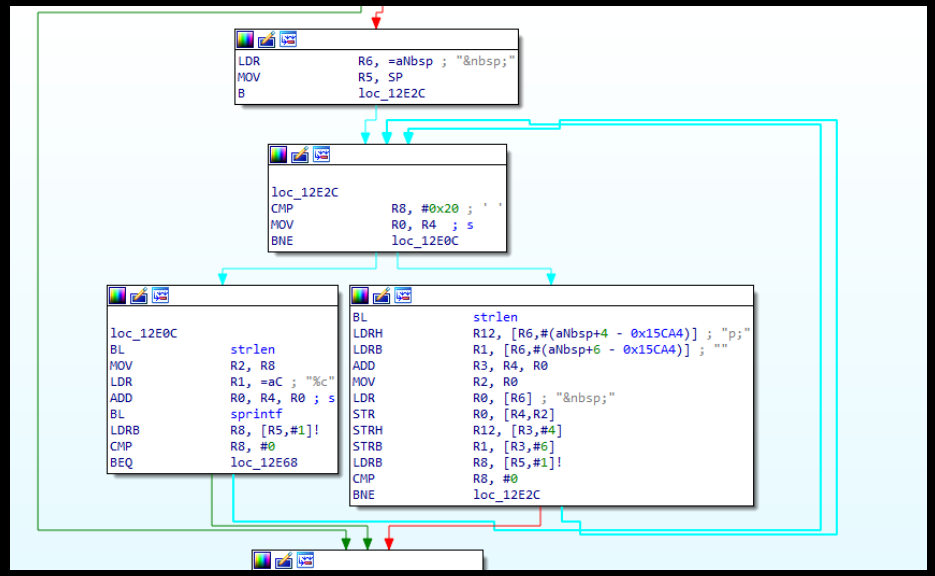
Result

- 1개의 프로그램에서 7개의 취약점 발견
  - Type confusion, Heap Buffer Overflow, OOB, DOS ...

Feedback driven fuzzing is powerfull!

IDA python

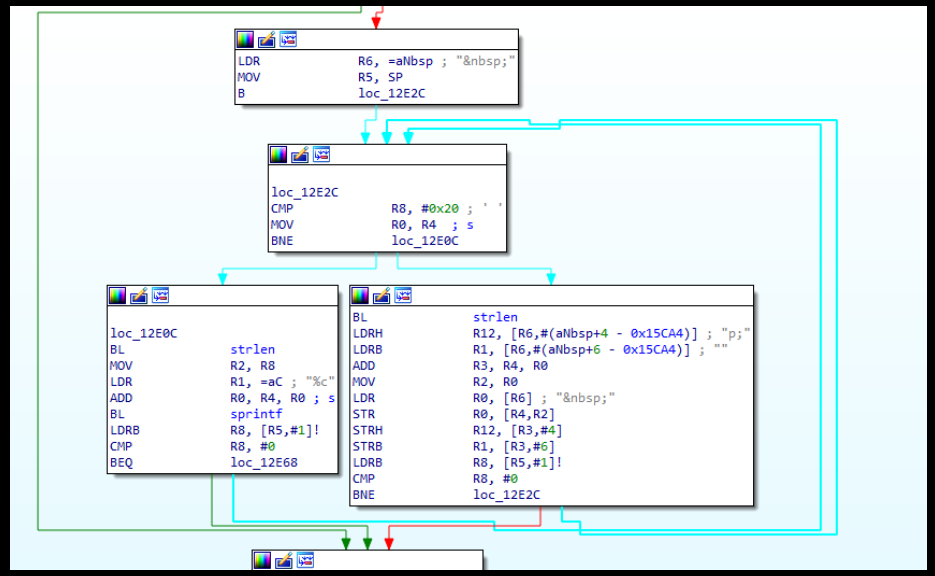
- IDA에서 정적 분석을 위해 지원하는 python 플러그인




```
if ( v9[0] )
{
  v4 = v9;
  do
  {
    while ( v3 != ' ' )
    {
      v5 = strlen(v10);
      sprintf(&v10[v5], "%c", v3);
      v6 = (unsigned __int8)*++v4;
      v3 = v6;
      if ( !v6 )
        return strcpy(a2, v10);
    }
    strcat(v10, " ");
    v7 = (unsigned __int8)*++v4;
    v3 = v7;
  }
  while ( v7 );
}
```

IDA python

- IDA에서 정적 분석을 위해 지원하는 python 플러그인



asm

```
if ( v9[0] )
{
  v4 = v9;
  do
  {
    while ( v3 != ' ' )
    {
      v5 = strlen(v10);
      sprintf(&v10[v5], "%c", v3);
      v6 = (unsigned __int8)*++v4;
      v3 = v6;
      if ( !v6 )
        return strcpy(a2, v10);
    }
    strcat(v10, " ");
    v7 = (unsigned __int8)*++v4;
    v3 = v7;
  }
  while ( v7 );
}
```

Hex-ray

Fuzzing vs IDA python

- 장점
  - Faster than Fuzzing
  - Can detect Non-Memory-curruption bug
- 단점
  - Hard to make good script
  - Etc...

- hard case

```
v3 = *(_DWORD *)(a1 + 72);
v4 = a3;
v5 = a2;
v6 = a1;
if ( !a3 )
{
  *(_DWORD *)(**(_QWORD **)(a1 + 80) + 40i64) = 40;
  (***(void (****)(void))(a1 + 80))();
}
if ( !*(_BYTE *)(v6 + 40) )
{
  v7 = v4 + v3;
  v8 = *(_QWORD *)(v6 + 64) | ((v5 & ((1i64 << v4) - 1)) << (24 - (unsigned __int8)v7));
  if ( (signed int)v7 >= 8 )
  {
    v9 = (unsigned __int64)v7 >> 3;
    v7 -= 8 * (v7 >> 3);
    do
    {
      **(_BYTE **)(v6 + 48) = BYTE2(v8);
      v10 = (_BYTE *)(*(_QWORD *)(v6 + 48) + 1i64);
      v11 = (*(_QWORD *)(v6 + 56))-- == 1i64;
      *(_QWORD *)(v6 + 48) = v10;
      if ( v11 )
      {
        v12 = *(_QWORD *)(*(_QWORD *)(v6 + 80) + 40i64);
        if ( !(*(unsigned __int8 (**)(void))(v12 + 24))() )
```

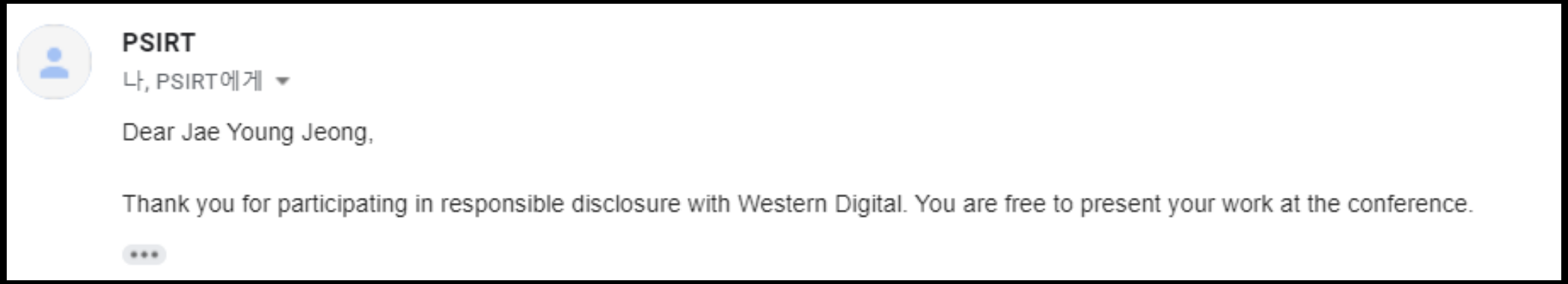
다음 코드에서 유저 인풋 변수와
호출되는 함수 주소를 구하시오 (4점)

IDA python

- 간단한 구조를 가진 프로그램에 유리
- IOT web server, CGI ...
  - Stack buffer overflow
  - command injection
  - etc

Let’s go to the real world!

- Western Digital



PSIRT

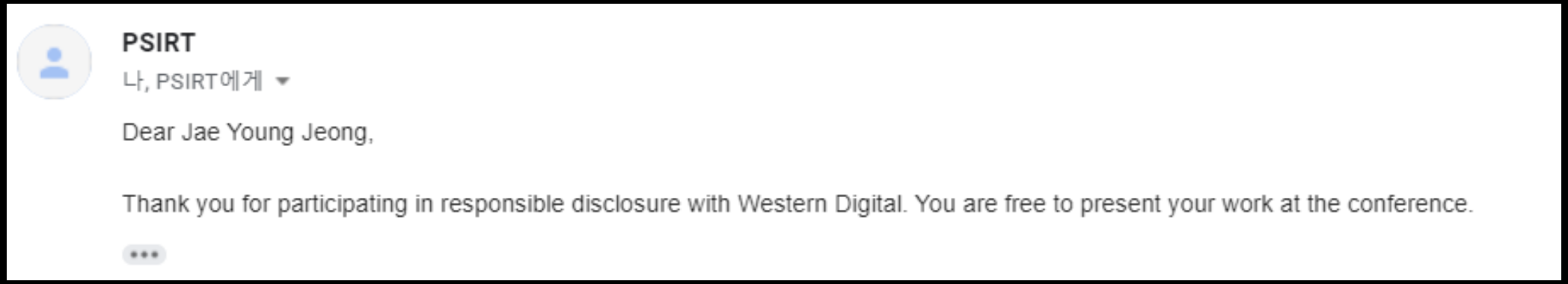
나, PSIRT에게

Dear Jae Young Jeong,

Thank you for participating in responsible disclosure with Western Digital. You are free to present your work at the conference.

Let’s go to the real world!

- Western Digital



PSIRT

나, PSIRT에게

Dear Jae Young Jeong,

Thank you for participating in responsible disclosure with Western Digital. You are free to present your work at the conference.

- Find Stack buffer overflow in CGI
  - Sprintf -> too many 1-days

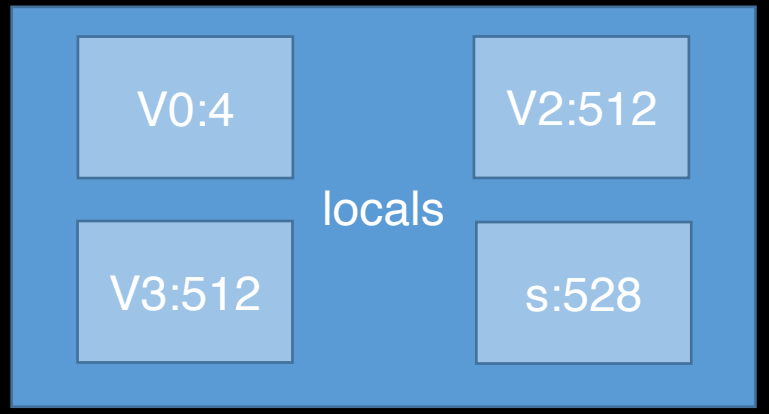
Progress

- Get local variable info
- Get user input info
- Get function arg info
- Check vulnerability

Get local variable info

```
size_t sub_12AB0()
{
  int v0; // r5
  char v2[512]; // [sp+0h] [bp-610h] BYREF
  char v3[512]; // [sp+200h] [bp-410h] BYREF
  char s[528]; // [sp+400h] [bp-210h] BYREF

  cgiFormString((int)"dir", (int)v2, 512);
  cgiFormString((int)"filename", (int)v3, 512);
  fwrite("Pragma: no-cache\r\nCache-Control: no-cache\r\n", 1u, 0x2Bu, (FILE *)cgiOut);
  cgiHeaderContentType("text/xml");
  fwrite("<?xml version=\"1.0\" encoding=\"UTF-8\"?>", 1u, 0x26u, (FILE *)cgiOut);
  fwrite("<mkdir>", 1u, 7u, (FILE *)cgiOut);
  v0 = sub_1290C(v2, v3);
  sprintf(s, "%s/%s", v2, v3);
  if ( v0 )
  {
    fwrite("<status>error</status>", 1u, 0x16u, (FILE *)cgiOut);
  }
  else
  {
    fwrite("<status>ok</status>", 1u, 0x13u, (FILE *)cgiOut);
    mkdir(s, 0x41FFu);
    chmod(s, 0x41FFu);
    chown(s, 0x1F5u, 0x1F5u);
  }
  return fwrite("</mkdir>", 1u, 8u, (FILE *)cgiOut);
}
```

- V0 -> int
- V2 -> char[512]
- V3 -> char[512]
- s ->char[528]



Get user input info

- 대부분의 IOT web 페이지 실행파일에서는 유저의 input을 parsing하기 위한 함수가 정해져 있음
- 라이브러리 함수를 쓰지 않고 직접 구현한 경우엔 리버싱...

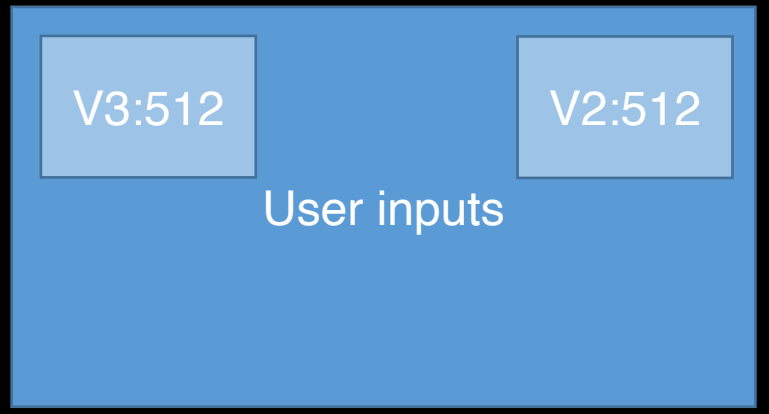
```
cgiFormString("show_file", v42, 2);
cgiFormString("dir", s, 512);
cgiFormString("chk_flag", &v40, 4);
cgiFormString("file_type", &v32, 32);
cgiFormString("function_id", v37, 12);
```

Get user input info

```
size_t sub_12AB0()
{
  int v0; // r5
  char v2[512]; // [sp+0h] [bp-610h] BYREF
  char v3[512]; // [sp+200h] [bp-410h] BYREF
  char s[528]; // [sp+400h] [bp-210h] BYREF

  cgiFormString((int)"dir", (int)v2, 512);
  cgiFormString((int)"filename", (int)v3, 512);
  fwrite("Pragma: no-cache\r\nCache-Control: no-cache\r\n", 1u, 0x2Bu, (FILE *)cgiOut);
  cgiHeaderContentType("text/xml");
  fwrite("<?xml version=\"1.0\" encoding=\"UTF-8\"?>", 1u, 0x26u, (FILE *)cgiOut);
  fwrite("<mkdir>", 1u, 7u, (FILE *)cgiOut);
  v0 = sub_1290C(v2, v3);
  sprintf(s, "%s/%s", v2, v3);
  if ( v0 )
  {
    fwrite("<status>error</status>", 1u, 0x16u, (FILE *)cgiOut);
  }
  else
  {
    fwrite("<status>ok</status>", 1u, 0x13u, (FILE *)cgiOut);
    mkdir(s, 0x41FFu);
    chmod(s, 0x41FFu);
    chown(s, 0x1F5u, 0x1F5u);
  }
  return fwrite("</mkdir>", 1u, 8u, (FILE *)cgiOut);
}
```

V2 -> char[512]

V3 -> char[512]

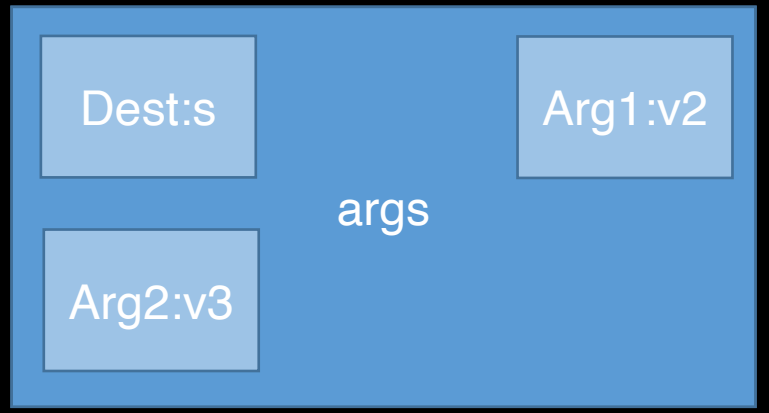


Get function arg info

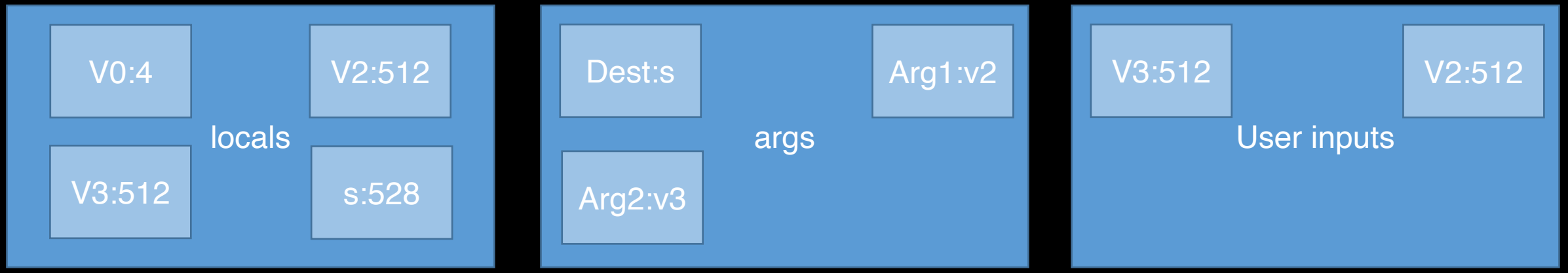
```
size_t sub_12AB0()
{
  int v0; // r5
  char v2[512]; // [sp+0h] [bp-610h] BYREF
  char v3[512]; // [sp+200h] [bp-410h] BYREF
  char s[528]; // [sp+400h] [bp-210h] BYREF

  cgiFormString((int)"dir", (int)v2, 512);
  cgiFormString((int)"filename", (int)v3, 512);
  fwrite("Pragma: no-cache\r\nCache-Control: no-cache\r\n", 1u, 0x2Bu, (FILE *)cgiOut);
  cgiHeaderContentType("text/xml");
  fwrite("<?xml version=\"1.0\" encoding=\"UTF-8\"?>", 1u, 0x26u, (FILE *)cgiOut);
  fwrite("<mkdir>", 1u, 7u, (FILE *)cgiOut);
  v0 = sub_1290C(v2, v3);
  sprintf(s, "%s/%s", v2, v3);
  if ( v0 )
  {
    fwrite("<status>error</status>", 1u, 0x16u, (FILE *)cgiOut);
  }
  else
  {
    fwrite("<status>ok</status>", 1u, 0x13u, (FILE *)cgiOut);
    mkdir(s, 0x41FFu);
    chmod(s, 0x41FFu);
    chown(s, 0x1F5u, 0x1F5u);
  }
  return fwrite("</mkdir>", 1u, 8u, (FILE *)cgiOut);
}
```

Format = “%s/%s”

source = V2/V3

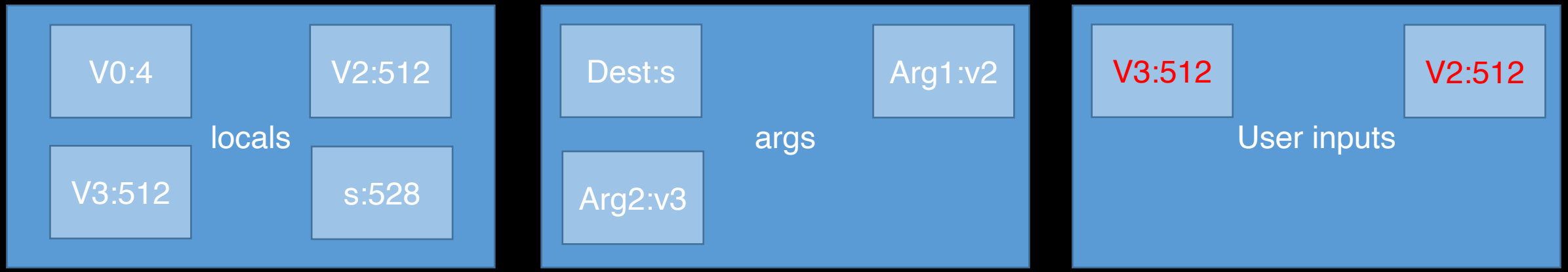


Check vulnerability



IF (dest len < source len) -> Stack buffer overflow

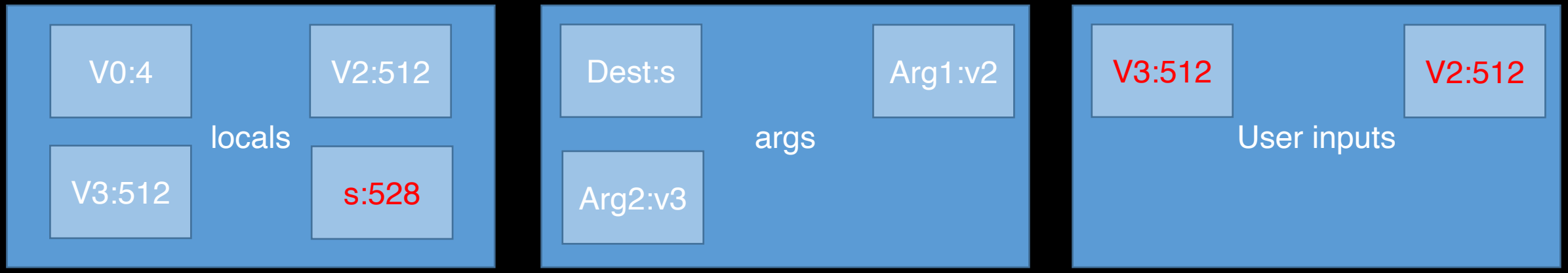
Check vulnerability



IF (dest len < 1024+1) -> Stack buffer overflow

Source = “%s/%s”,v2,v3

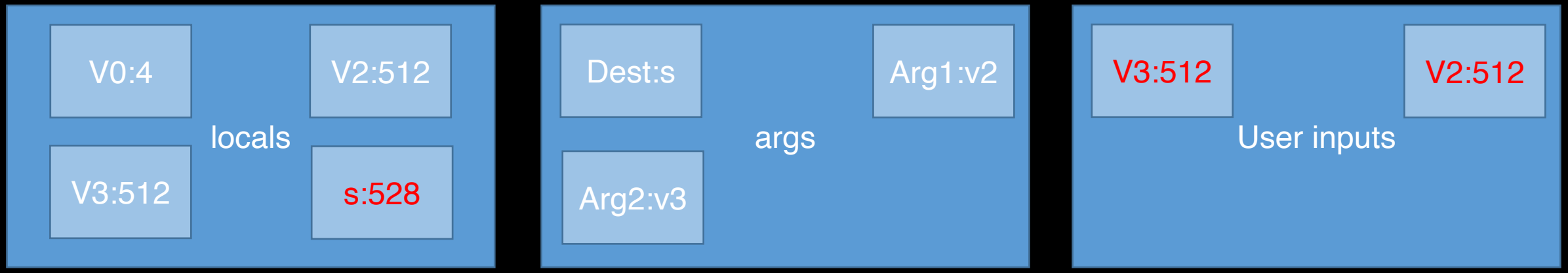
Check vulnerability



IF (528 < 1024+1) -> Stack buffer overflow

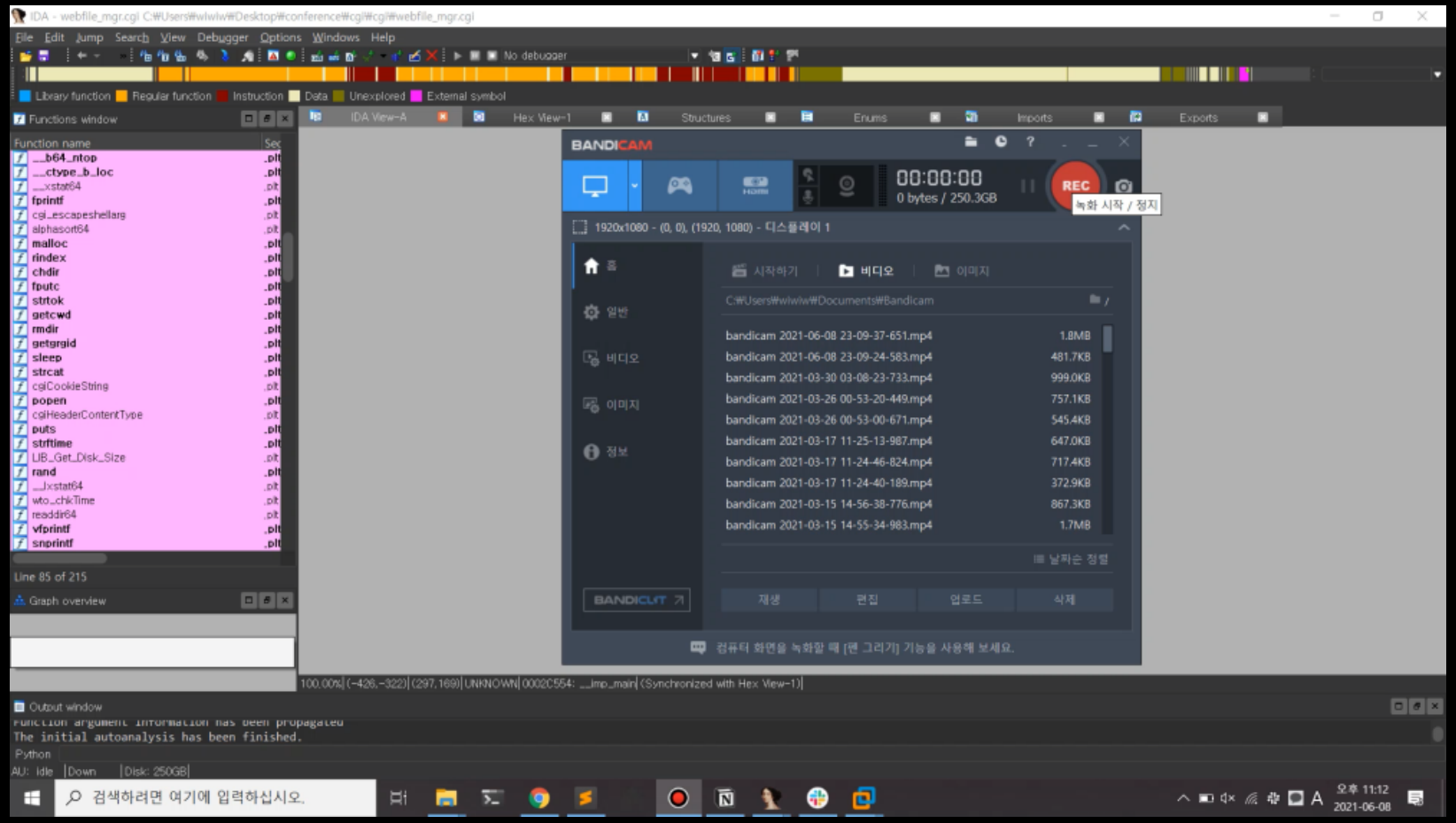
Dest = s:528

Check vulnerability



IF (528 < 1024+1) -> **Stack buffer overflow**

DEMO


IDA - webfile_mgr.cgi C:₩Users₩wiwiw₩Desktop₩conference₩cgi₩cgi₩webfile_mgr.cgi
File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol
Functions window
Function name
__b64_ntop
__ctype_b_loc
__xstat64
fprintf
cgi_escapeshellarg
alphasort64
malloc
rindex
chdir
fputc
strtok
getcwd
rmdir
getgrgid
sleep
strcat
cgiCookieString
popen
cgiHeaderContentType
puts
strftime
LIB_Get_Disk_Size
rand
__lxstat64
wto_chkTime
readdir64
vfprintf
snprintf
Line 85 of 215
Graph overview
IDA View-A
Hex View-1
Structures
Enums
Imports
Exports
BANDICAM
00:00:00
0 bytes / 250.3GB
REC
녹화 시작 / 정지
1920x1080 - (0, 0), (1920, 1080) - 디스플레이 1
홈
일반
비디오
이미지
정보
시작하기
비디오
이미지
C:₩Users₩wiwiw₩Documents₩Bandicam
bandicam 2021-06-08 23-09-37-651.mp4 1.8MB
bandicam 2021-06-08 23-09-24-583.mp4 481.7KB
bandicam 2021-03-30 03-08-23-733.mp4 999.0KB
bandicam 2021-03-26 00-53-20-449.mp4 757.1KB
bandicam 2021-03-26 00-53-00-671.mp4 545.4KB
bandicam 2021-03-17 11-25-13-987.mp4 647.0KB
bandicam 2021-03-17 11-24-46-824.mp4 717.4KB
bandicam 2021-03-17 11-24-40-189.mp4 372.9KB
bandicam 2021-03-15 14-56-38-776.mp4 867.3KB
bandicam 2021-03-15 14-55-34-983.mp4 1.7MB
남은 정렬
BANDICUT 기
재생
편집
업로드
삭제
컴퓨터 화면을 녹화할 때 [펜 그리기] 기능을 사용해 보세요.
100.00% (-426,-322) (297,169) UNKNOWN 0002C554: __imp_main (Synchronized with Hex View-1)
Output window
The initial autoanalysis has been finished.
Python
AU: idle Down Disk: 250GB
검색하려면 여기에 입력하십시오.
오후 11:12
2021-06-08

Monitoring tools

- Windows
  - Process monitor
- Linux
  - ltrace, strace
  - Gdb script

Monitoring tools

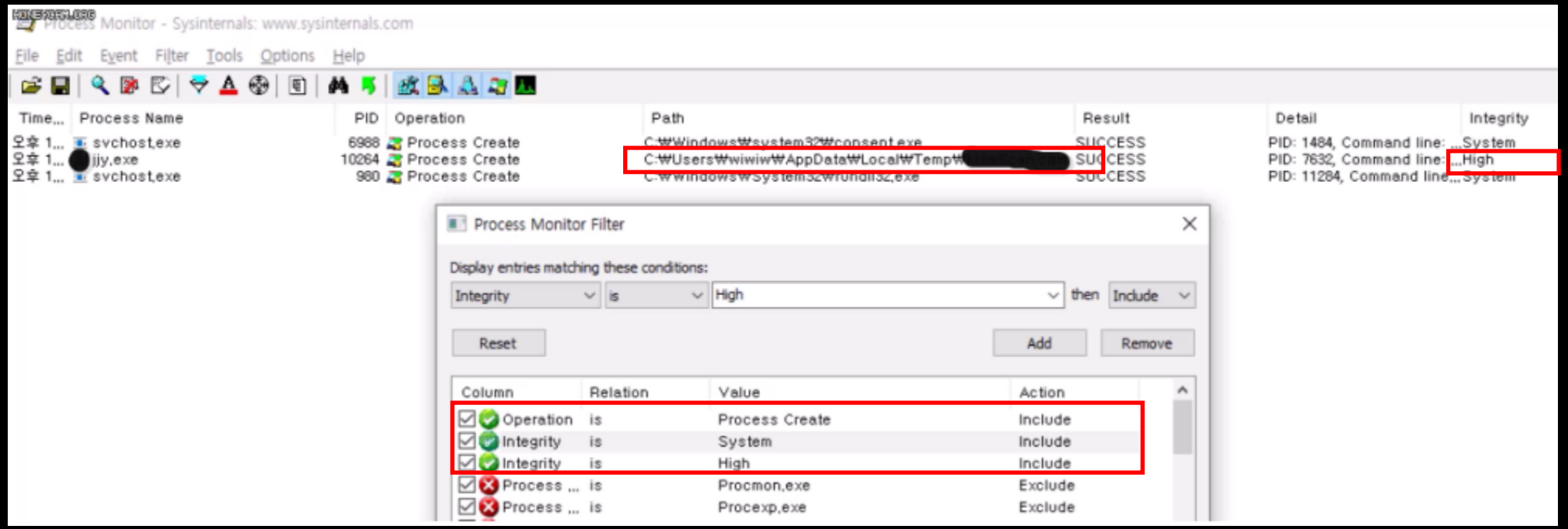
- LPE 취약점을 쉽게 찾을 수 있음.
- Target
  - Root process - linux
  - System(admin) process - windows

Monitoring tools

- 타겟 프로세스가 하는 행위를 관찰
  - Execute file
  - Read file
  - Write file
  - Load Library

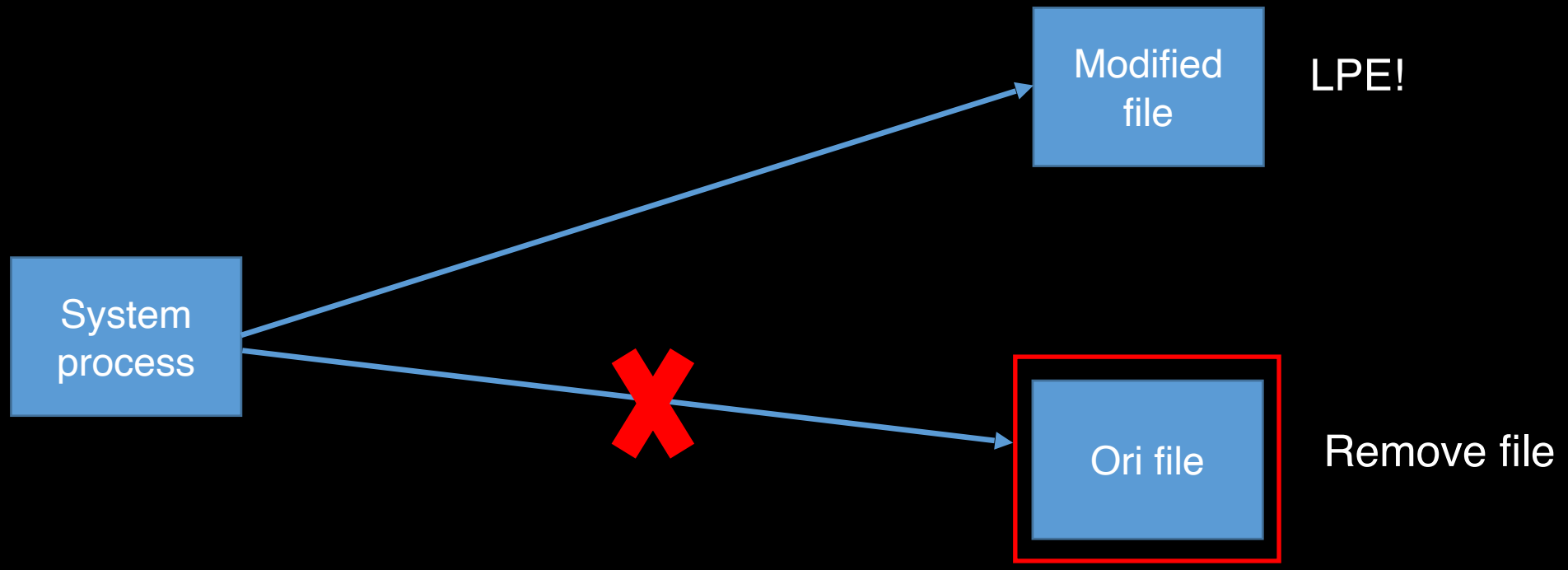
Monitoring tools

- **Execute file at writeable file**



Monitoring tools

- **Execute file at writeable file**
  - **Just replace exe file!**



Monitoring tools

- **Read file from writeable file**
  - 해당 벡터를 공격 벡터로 사용 가능해짐.

```
[config]
    "config_key" = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAA\\RETURN_ADDRESS"
[/config]
```

Monitoring tools

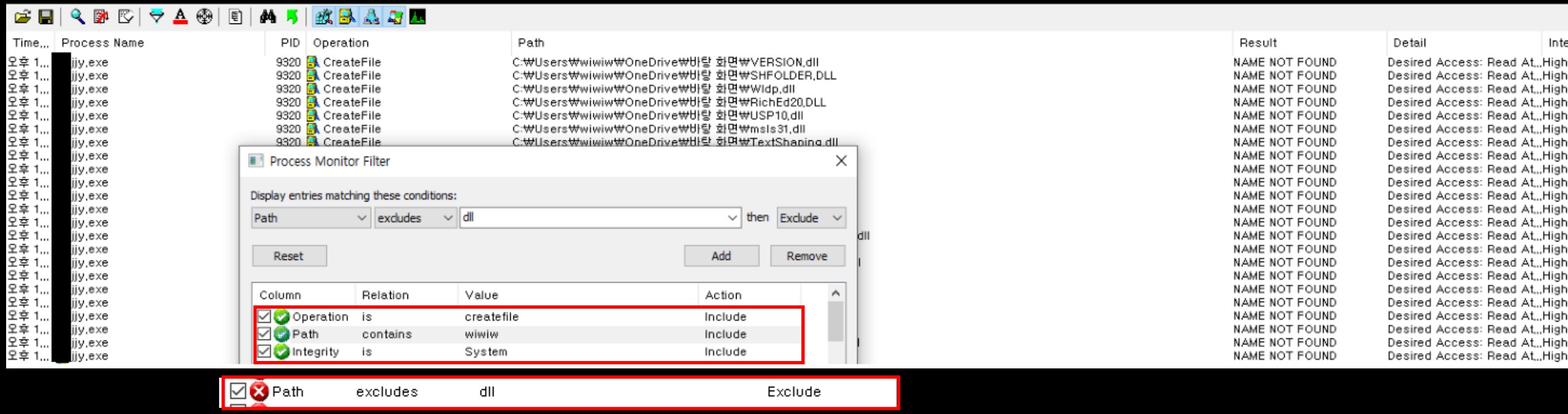
Write file at writeable folder

- Symbolic link attack
- Root 권한으로 파일을 쓸 수 있게 됨.
- Link to ‘/etc/passwd’, ‘/root/.bashrc’

```
➜ test ls -al
total 12
drwxrwxrwx 2 root root 4096 Jun  6 01:40 .
drwxrwxr-x 3 jjy  jjy  4096 Jun  6 01:36 ..
-rw-r--r-- 1 root root    3 Jun  6 01:40 read_only
➜ test rm read_only
rm: remove write-protected regular file 'read_only'? y
➜ test ls -al
total 8
drwxrwxrwx 2 root root 4096 Jun  6 01:40 .
drwxrwxr-x 3 jjy  jjy  4096 Jun  6 01:36 ..
➜ test
```

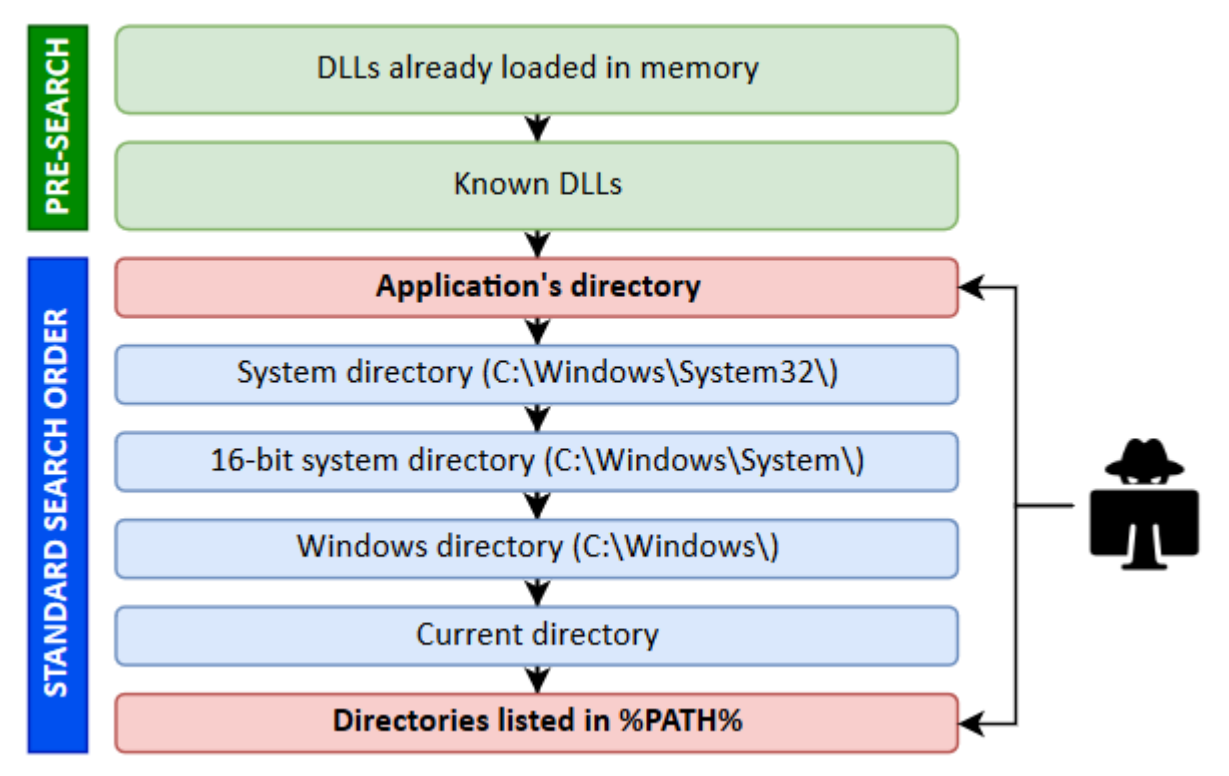
Monitoring tools

- **Load Library from writeable path**
  - **DLL Hijacking!!! - windows**

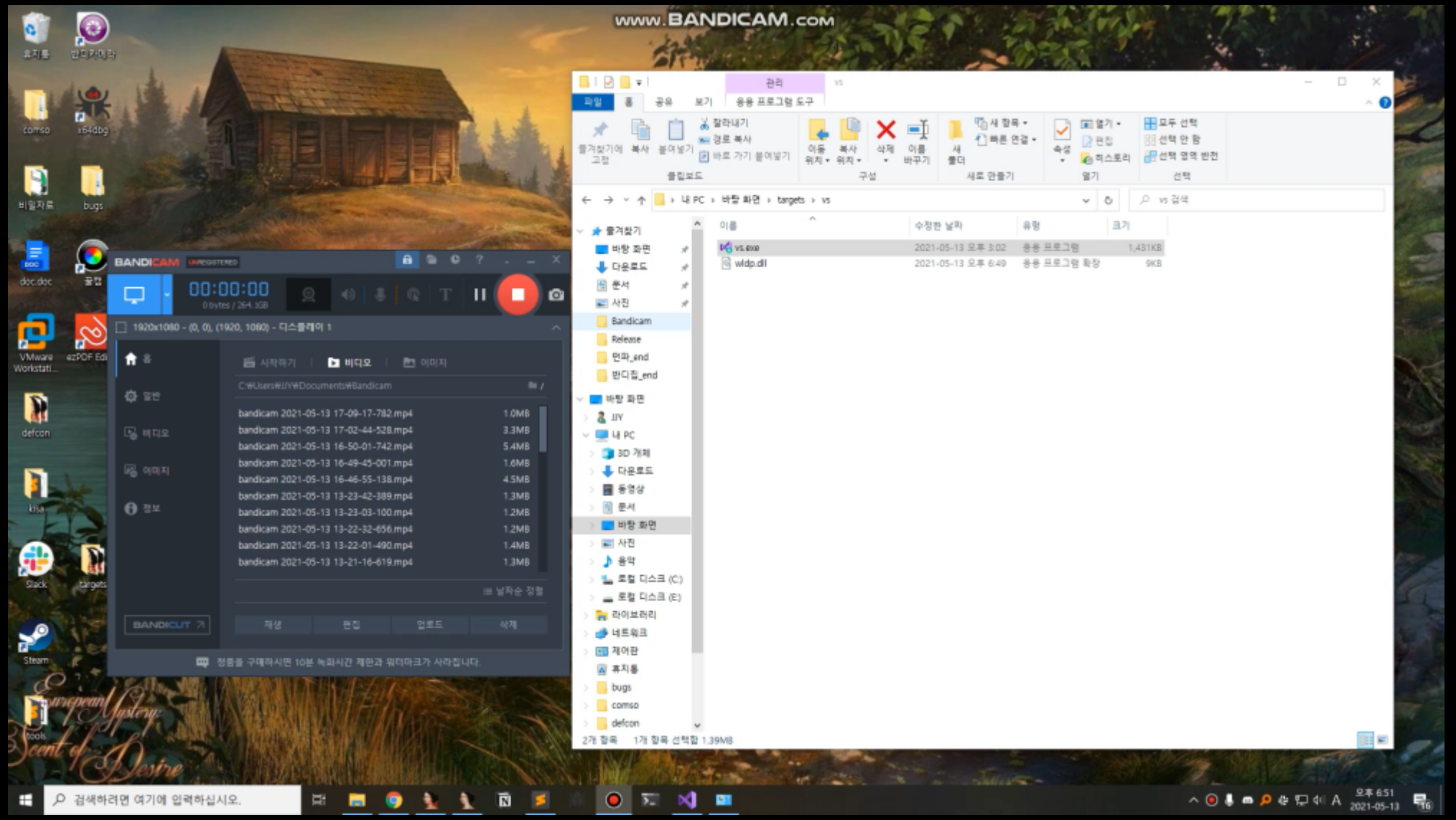


Monitoring tools

- **Load Library from writeable path**
  - **DLL Hijacking!!! - windows**



DEMO


www.BANDICAM.com
BANDICAM
00:00:00
bandicam 2021-05-13 17-09-17-782.mp4 1.0MB
bandicam 2021-05-13 17-02-44-528.mp4 3.3MB
bandicam 2021-05-13 16-50-01-742.mp4 5.4MB
bandicam 2021-05-13 16-49-45-001.mp4 1.6MB
bandicam 2021-05-13 16-46-55-138.mp4 4.5MB
bandicam 2021-05-13 13-23-42-389.mp4 1.3MB
bandicam 2021-05-13 13-23-03-100.mp4 1.2MB
bandicam 2021-05-13 13-22-32-656.mp4 1.2MB
bandicam 2021-05-13 13-22-01-490.mp4 1.4MB
bandicam 2021-05-13 13-21-16-619.mp4 1.3MB
vs.exe 2021-05-13 오후 3:02 응용 프로그램 1,431KB
wldp.dll 2021-05-13 오후 6:49 응용 프로그램 확장 9KB

Find SW vulnerability
via automatic method

CodeEngn