Buffer Overflow Report

Contributors

Bhupesh, 2021CS10101

Naman Nirwan, 2021CS50593

1. Introduction This report documents the process of identifying and fixing buffer overflow vulnerabilities in the web server. It includes stack representations before and after overflow, exploit analysis, and implemented fixes.

2. Stack Representation

Before Overflow: In a normal function execution, the stack consists of:

- Function arguments
- Local variables (buffers, pointers, etc.)
- Saved registers
- Return address

Higher Memory
Function Arg
Local Vars
Saved Regs
Return Addr | <- Expected return location
Lower Memory

After Overflow: A buffer overflow occurs when user input exceeds the buffer size and overwrites adjacent stack elements, including the return address.

Higher Memory
Function Arg
Overflowed
Saved Regs
Overwritten | <- Return Address Hijacked
Malicious | <- Redirects Execution
Payload
Lower Memory

This results in the execution jumping to an unintended address, allowing our exploits to execute code which is malicious and deletes the file /home/student/grades.txt.

3. Identified Vulnerabilities and Fixes

Vulnerability 1: Buffer Overflow in `http_request_headers`

- **Description:** The buffer `buf` used in `http_request_headers` was vulnerable to overflow because it failed to restrict user input size when parsing headers.
- **Exploit Mechanism:** The exploit sends an HTTP request with an overly long `USER_NAME` header, which overflows `buf` and overwrites the return address.
- **Fix Implemented:**
    - Replaced unbounded string operations with `strncpy` and added bounds checks.
    - Introduced explicit length validation before copying header values.

Before Fix:

```
static char buf[8192];
if (http_read_line(fd, buf, sizeof(buf)) < 0)
    return "Socket IO error";
```

After Fix:

```
static char buf[512]; // Reduced buffer size to prevent overflow
if (http_read_line(fd, buf, sizeof(buf) - 1) < 0)
    return "Socket IO error";
buf[sizeof(buf) - 1] = '\0'; // Ensure null termination
```

```
student@65660-v23:~/lab$ sudo make check-crash
./check-bin.sh
tar xf bin.tar.gz
for f in ./exploit-2*.py; do ./check-crash.sh zookd-exstack $f; done
PASS ./exploit-2.py
```

```
student@65660-v23:~/lab$ sudo make check-exstack
./check-bin.sh
tar xf bin.tar.gz
for f in ./exploit-4*.py; do ./check-attack.sh zookd-exstack $f; done
PASS ./exploit-4.py
```

Vulnerability 2: Return-to-libc Attack (`exploit-5.py`)

- **Description:** The exploit used a return-to-libc technique by overwriting the return address with the address of `unlink()` to delete `/home/student/grades.txt`.
- **Exploit Mechanism:**
  - Overflows the buffer in `http_request_headers`.
  - Overwrites the return address with a pointer to `accidentally()`.
  - Uses `accidentally()` to move arguments into the correct registers.
  - Redirects execution to `unlink("/home/student/grades.txt")`.
- **Fix Implemented:**
  - Removed the `accidentally()` function to eliminate an easy gadget for ROP.
  - **Disabled SIGPIPE errors** to prevent unexpected crashes when writing to a closed socket.

Fixes:

```
if (strlen(buf) >= sizeof(buf) - 1)
    return "Header too long (4)";

// Ignore SIGPIPE to prevent crashes when writing to a closed socket
signal(SIGPIPE, SIG_IGN);
```

Exploit 5 is successful

```
student@65660-v23:~/lab$ sudo make check-libc
./check-bin.sh
tar xf bin.tar.gz
for f in ./exploit-5*.py; do ./check-attack.sh zookd-nxstack $f; done
PASS ./exploit-5.py
```

4. Verification The implemented fixes were verified by running `sudo make check-fixed`. The results confirm that the exploits no longer work, indicating successful mitigation of buffer overflow vulnerabilities.

```
student@65660-v23:~/lab$  sudo make check-fixed
if [ -x zookclean.py ]; then ./zookclean.py; fi
rm -f *.o *.pyc *.bin zookd zookd-exstack zookd-nxstack zookd-withssp shellcode.bin run-shellcode
cc zookd.c -c -o zookd.o -m64 -g -std=c99 -Wall -Wno-format-overflow -D_GNU_SOURCE -static -fno-stack-protector
cc http.c -c -o http.o -m64 -g -std=c99 -Wall -Wno-format-overflow -D_GNU_SOURCE -static -fno-stack-protector
cc -m64  zookd.o http.o   -o zookd
cc -m64 zookd.o http.o   -o zookd-exstack -z execstack
cc -m64 zookd.o http.o   -o zookd-nxstack
cc zookd.c -c -o zookd-withssp.o -m64 -g -std=c99 -Wall -Wno-format-overflow -D_GNU_SOURCE -static
cc http.c -c -o http-withssp.o -m64 -g -std=c99 -Wall -Wno-format-overflow -D_GNU_SOURCE -static
cc -m64  zookd-withssp.o http-withssp.o   -o zookd-withssp
cc -m64   -c -o shellcode.o shellcode.S
objcopy -S -O binary -j .text shellcode.o shellcode.bin
cc run-shellcode.c -c -o run-shellcode.o -m64 -g -std=c99 -Wall -Wno-format-overflow -D_GNU_SOURCE -static -fno-stack-pr
otector
cc -m64  run-shellcode.o   -o run-shellcode
for f in ./exploit-2*.py; do ./check-crash.sh zookd-exstack $f; done
FAIL ./exploit-2.py
for f in ./exploit-4*.py; do ./check-attack.sh zookd-exstack $f; done
FAIL ./exploit-4.py
for f in ./exploit-5*.py; do ./check-attack.sh zookd-nxstack $f; done
FAIL ./exploit-5.py
```

```
student@65660-v23:~/lab$  sudo make check-lab1
./check-zoobar.py
+ removing zoobar db
+ running make.. output in /tmp/make.out
+ running zookd in the background.. output in /tmp/zookd.out
PASS Zoobar app functionality
./check-bin.sh
tar xf bin.tar.gz
for f in ./exploit-2*.py; do ./check-crash.sh zookd-exstack $f; done
PASS ./exploit-2.py
./check-bin.sh
tar xf bin.tar.gz
for f in ./exploit-4*.py; do ./check-attack.sh zookd-exstack $f; done
PASS ./exploit-4.py
./check-bin.sh
tar xf bin.tar.gz
for f in ./exploit-5*.py; do ./check-attack.sh zookd-nxstack $f; done
PASS ./exploit-5.py
./check-bin.sh
tar xf bin.tar.gz
./check-attack.sh zookd-nxstack ./exploit-challenge.py
PASS ./exploit-challenge.py
```

5. Conclusion This report details the identified vulnerabilities, exploits, and fixes applied to the web server. By implementing proper bounds checking and security mechanisms, the buffer overflow vulnerabilities have been successfully mitigated.