System Security - Attack and Defense for Binaries

CS 4390/5390, Spring *2026*

Instructor: MD Armanuzzaman (***Arman***)

Real-world Examples

Morris Worm

The vulnerability was in fingerd from 4.3BSD Unix, the version of the Berkeley Software Distribution (BSD) released in 1986.

```
/*
 * Finger server.
 */
#include <sys/types.h>
#include <netinet/in.h>

#include <stdio.h>
#include <ctype.h>

main(argc, argv)
        char *argv[];
{
        register char *sp;
        char line[512];
        struct sockaddr_in sin;
        int i, p[2], pid, status;
        FILE *fp;
        char *av[4];

        i = sizeof (sin);
        if (getpeername(0, &sin, &i) < 0)
                fatal(argv[0], "getpeername");
        line[0] = '\0';
        gets(line);
        sp = line;
        av[0] = "finger";
        i = 1;
```

https://www.tuhs.org/cgi-bin/utree.pl?file=4.3BSD/usr/src/etc/fingerd.c

OpenBSD 2.8 ftpd Off-by-One

In 2000 a buffer overflow was discovered in the piece of code handling directory names in the FTP daemon included in OpenBSD distribution. The vulnerable piece of code is shown here (/src/libexec/ftpd/ftpd.c):

MAXPATHLEN is 1024

```
replydirname(name, message)
    const char *name, *message;
{
    char npath[MAXPATHLEN];
    int i;

    for (i = 0; *name != '\0' && i < sizeof(npath) - 1; i++, name++) {
        npath[i] = *name;
        if (*name == '"')
                npath[++i] = '"';
    }
    npath[i] = '\0';
    reply(257, "\"%s\" %s", npath, message);
}
```

A Recent Example:
JuiceBox 40 Smart EV Charging Station

A classic stack-based buffer overflow

The Gecko OS provides a template for setting log formats, including tags such as timestamp, SSID, host, port, and MAC address. The template has a 32-character limit, including a NULL byte for termination. Each tag, such as @t for the timestamp, uses two characters, allowing a maximum of 15 tags per template. When the @t timestamp tag is used, it outputs 23 bytes into the message buffer, meaning 15 timestamp tags would generate 345 bytes. However, the buffer is only 192 bytes long. This vulnerability was uncovered through firmware analysis, which helped the team locate the function responsible for handling the message format.

https://vicone.com/blog/from-pwn2own-automotive-a-stack-based-buffer-overflow-vulnerability-in-juic

A Recent Example:
JuiceBox 40 Smart EV Charging Station

JuiceBox 40 (CVE-2024-23938)

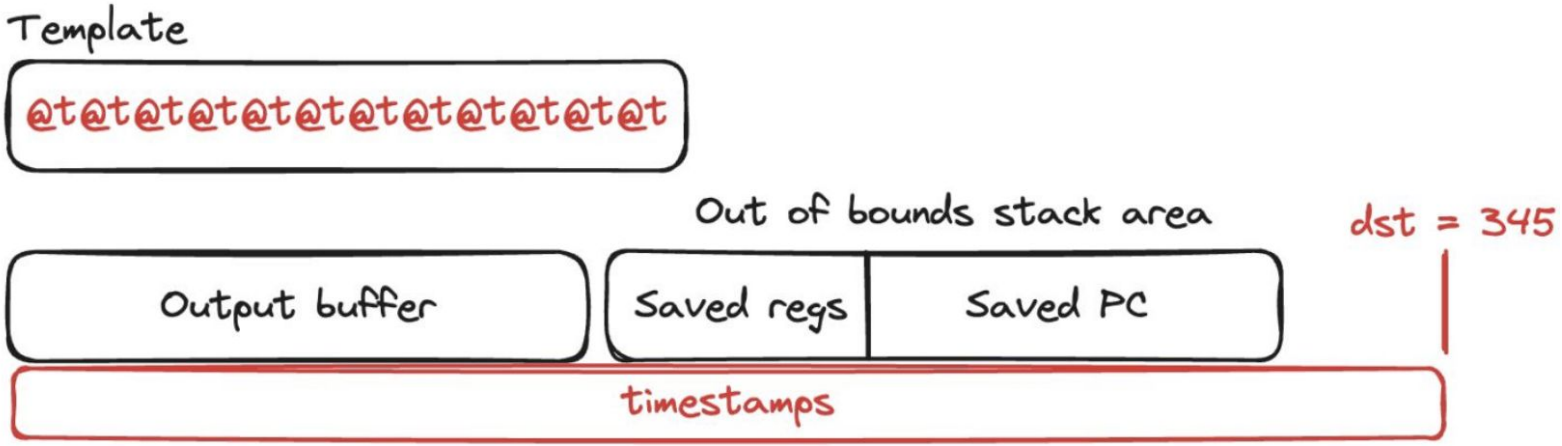
```
char scratch_buffer[132];
char formatted_msg_buffer[192];
char * dst = formatted_msg_buffer;
// ...
if ((format_tag == 't') &&
   (print_timestamp_to_string(scratch_buffer, 1) == SUCCESS))
{
  memcpy(dst, scratch_buffer, 10);
  dst[10] = ' ';
  dst[11] = '|';
  dst[12] = ' ';
  memcpy(dst + 13, scratch_buffer + 11, 8);
  dst[21] = ':';
  dst[22] = ' ';
  dst = dst + 23;
  *dst = '\0';
}
```

*https://i.blackhat.com/BH-US-24/Presentations/US24-Alkemade-Low-Energy-to-High-Energy-Hacking-Nearby-EV-Chargers-Over-Bluetooth-Wednesday.pdf?_gl=1*1s6dkoi*_gcl_au*NTY2MjE0MjI2LjE3Mjk1NTc4Mjg.*_ga*MTIxOTgyOTExMy4xNzI5NTU3ODI5*_ga_K4JK67TFYV*MTcyOTU1NzgyOC4xLjAuMTcyOTU1NzgyOC4wLjAuMA..&_ga=2.169853153.1304097414.1729557829-1219829113.1729557829*

A Recent Example: JuiceBox 40 Smart EV Charging Station

JuiceBox 40 (CVE-2024-23938)

- **What if we provide multiple @t tags?**
  - At most 15 times, each using up **23** bytes
  - **15 * 23 = 345** bytes, while the stack allocated buffer is **192** bytes long
- No canaries, no ASLR, but some limitations on allowed byte values



Tesla hacked, 24 zero-days demoed at Pwn2Own Automotive 2024

Finding Buffer Overflow in Source Code

Possible Approaches

- Lexical static code analysis
- Semantic static code analysis
- Dynamic program analysis, e.g., Valgrind
- Formal methods based approaches, e.g., using Coq

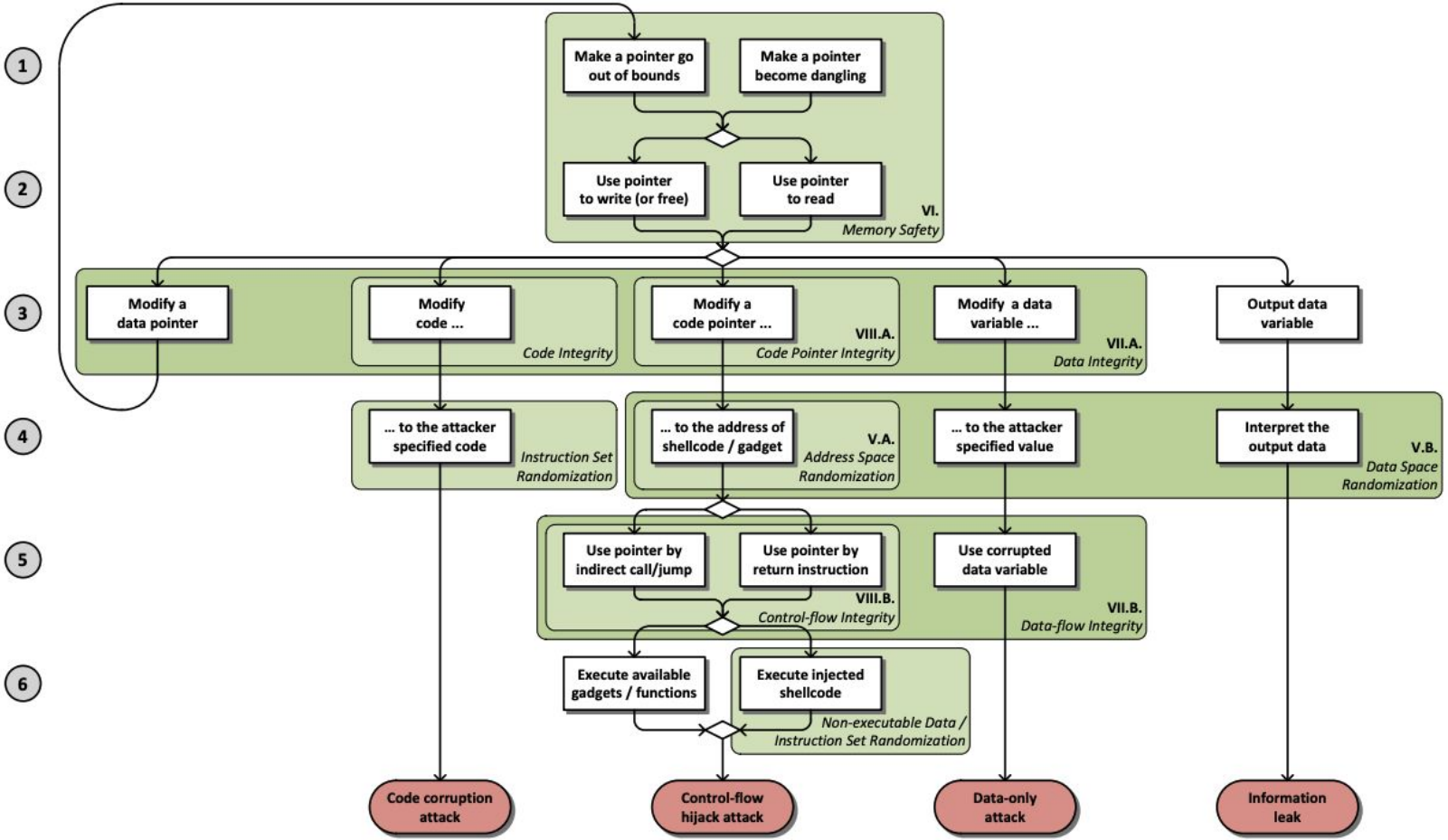
Possible Approaches

- Fuzzing: breaking software/hardware with random inputs
  - Blackbox vs. whitebox
  - Coverage-based, mutation-based, grammar-based
  - Symbolic execution, concolic execution
  - Re-hosting
- AI and Large Language Models

https://www.fuzzingbook.org/

LLM-guided Fizzing

- Utilize the LLM's reasoning and generation capability to improve fuzzing
- Recent research:
  - *Fuzzing BusyBox: Leveraging LLM and Crash Reuse for Embedded Bug Unearthing*
    - *USENIX'2024*
  - *Large Language Model guided Protocol Fuzzing*
    - *NDSS'2024*



SoK: Eternal War in Memory. IEEE S&P 2013