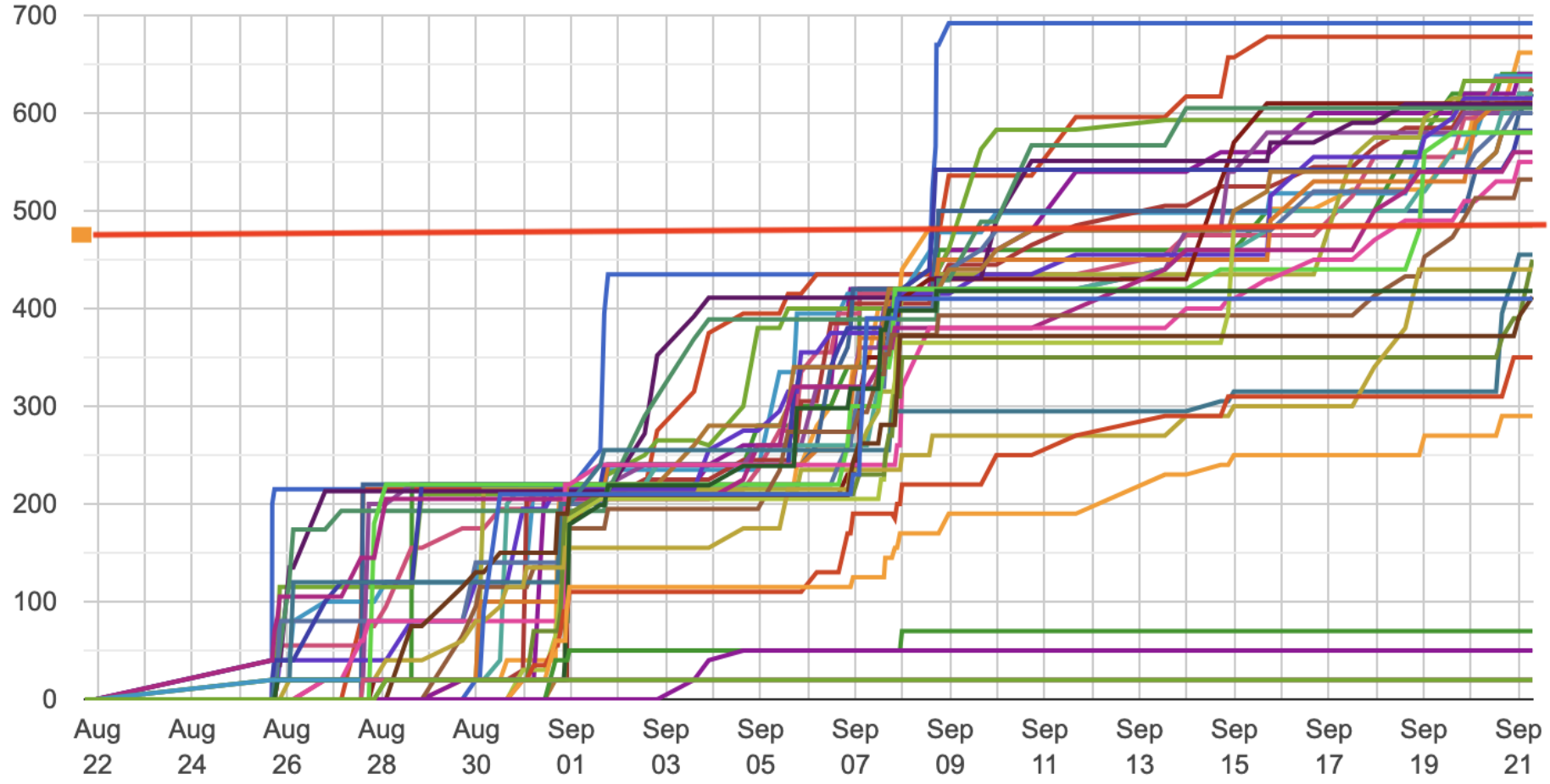
Lec05: Stack Protections

Taesoo Kim

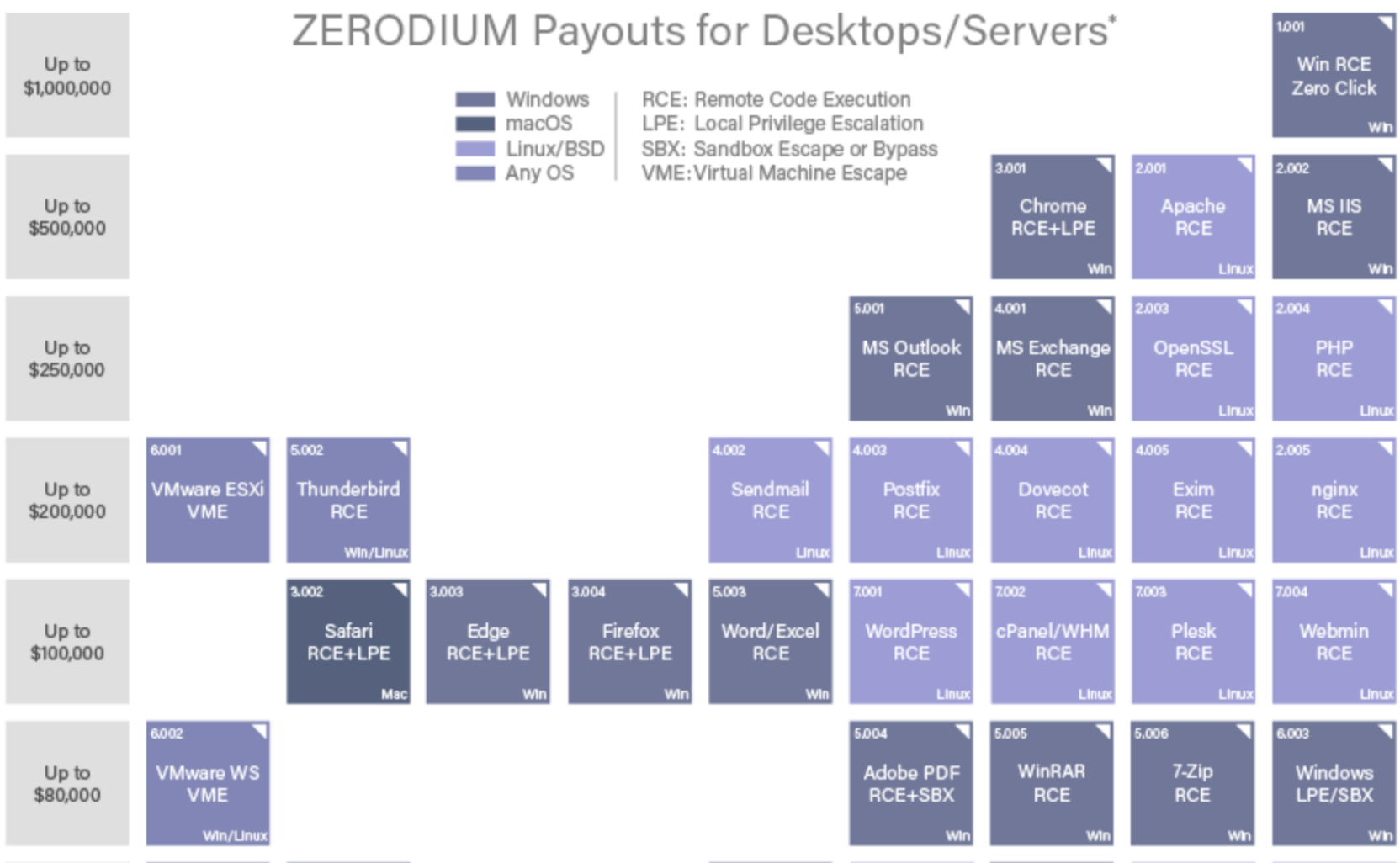
Administrivia

- Please submit your write-ups on time!
- Please write down your collaborators' names on the write-ups
- Due: Lab04 is out, and its due on Sept 28 at midnight

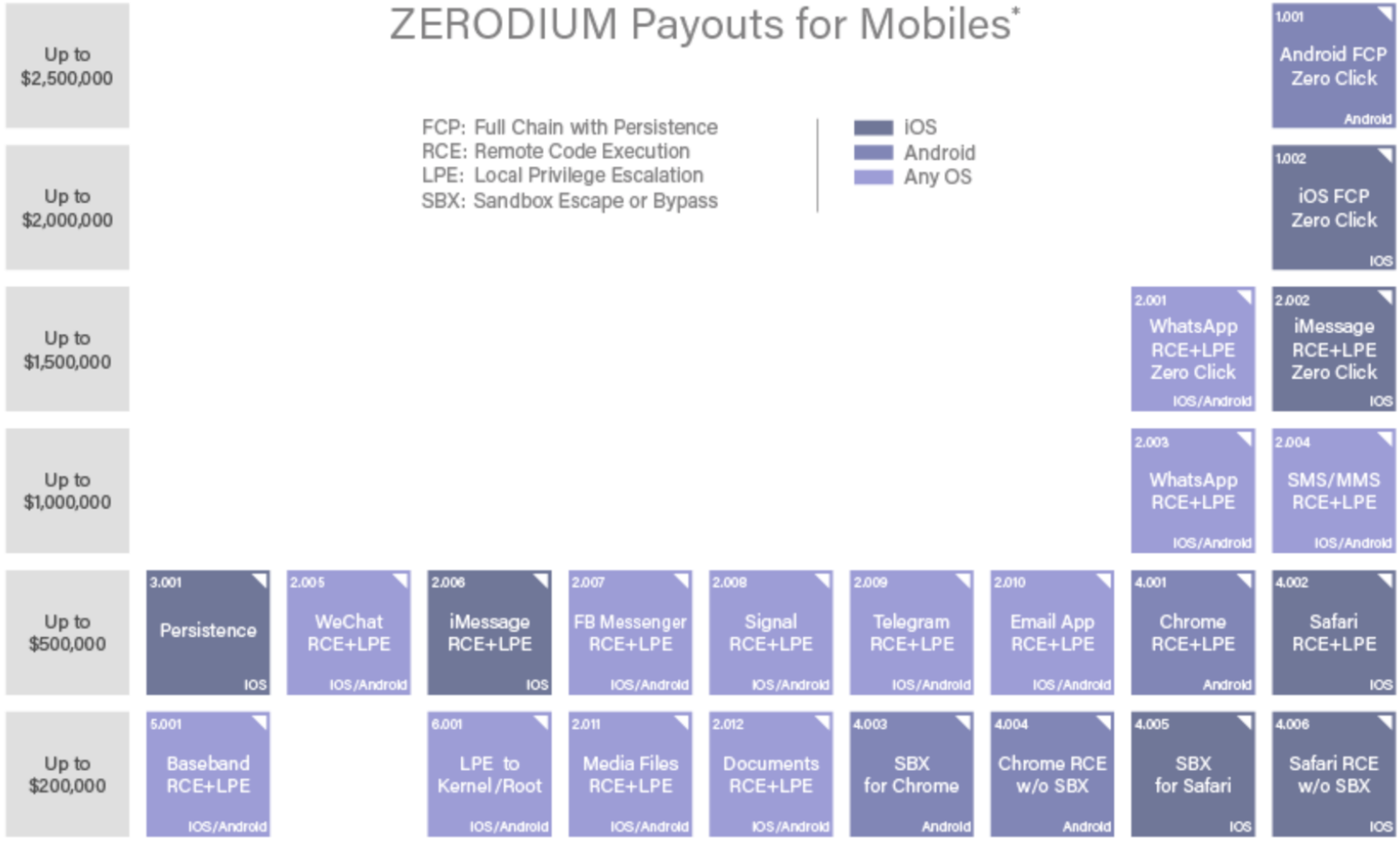
Lab03: Stack Overflow



Discussion: Not Yet Motivated?



Discussion: Not Yet Motivated?



Discussion: jmp-to-where

- What's the bug?
- What's special about this challenge?

```
  (x86-32)
0x08048000
 | code   .text section
 | data   .data section
 | bss    .bss section
 | ...
 | stack  stack
 | args   program args
 | envs   program envs
 | prog   filename of program
 | null   final dword of zero
0xBFFFFFFF
```

Discussion: jmp-to-where


```
 (stack)
| argc
| argv[0]
| ...
| argv[argc-1]
| NULL
| env[0]
| ...
| env[n]
| NULL
| ...
| [/your-program-name] <= is this something you can control?
```

→ A user-controlled input can be used in a surprising way!

Discussion: unusual-main

- What's the bug?
- What's special about this challenge?

Discussion: unusual-main

- vuln()

```
     0x080484d5 <+63>:    mov    ebx,DWORD PTR [ebp-0x4]
==>  0x080484d8 <+66>:    leave
     0x080484d9 <+67>:    ret
```

→ Stack pivoting: control `eip` by modifying `ebp`

Discussion: man-strncpy

- What's the bug?
- What's special about this challenge?

Discussion: man-strncpy

```
1  char *
2  strncpy(char *dest, const char *src, size_t n) {
3    size_t i;
4    for (i = 0; i < n && src[i] != '\0'; i++)
5      dest[i] = src[i];
6    for ( ; i < n; i++)
7      dest[i] = '\0';
8    return dest;
9  }
```

- NULL on the remaining bytes
- Corrupting the lowest byte of `ebp`
- And, any other problems?

Discussion: man-strncpy

- What's your lesson?
  - API misuses introduce subtle bugs (but mostly designed for reasons)
  - By the way, check out a better language like Rust
- How to prevent this?

Discussion: man-strncpy


```
    char buf[BUFSIZ];
    strncpy(buf, input, ???);

    1. BUFSIZ   (sizeof(buf))?
    2. BUFSIZ-1 (sizeof(buf)-1)?

NOTE.
    snprintf(buf, sizeof(buf), "...")
```

Discussion: man-strncpy (safe usage)


```
1       char buf[BUFSIZ];
2       strncpy(buf, input, sizeof(buf) - 1);
3       buf[sizeof(buf) - 1] = '\0';
```

Discussion: alternative strlcpy()

```
1       strlcpy(buf, s, sizeof(buf));
```

strlcpy(3) - Linux man page

Name

strlcpy, **strlcat** - size-bounded string copying and concatenation

Library

Utility functions from BSD systems (libbsd, -lbsd)

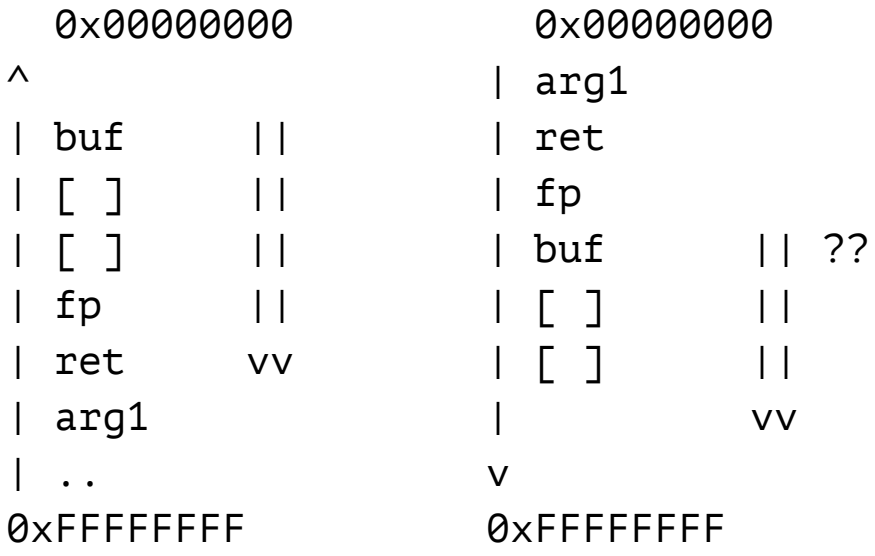
Synopsis

#include <bsd/string.h>

size_t

strlcpy(*char *dst*, *const char *src*, *size_t size*);

Discussion: upside-down



- Overwriting function arguments saved on the stack

Discussion: upside-down

- More secure? less? in terms of security?

How to Prevent Stack Overflow?

- Two approaches:
  - Bug prevention
  - Exploitation mitigation
- Protect “integrity” of ra, funcptr, etc (code pointers)
  - (e.g., exploitation mitigation → NX, canary)
- Prevent the buffer overflow at the first place
  - (e.g., code analysis, better APIs)

Today's Tutorial

- In-class tutorial
  - Let's understand the implementation of the stack protector.
  - Let's exploit the (insecurely) protected crackme0x00 to get a flag!

Reminder: crackme0x00


```
$ objdump -M intel -d crackme0x00
...
80485c4:   lea    eax,[ebp-0x1c]
80485c7:   push   eax
80485c8:   lea    eax,[ebx-0x191e]
80485ce:   push   eax
80485cf:   call   8048440 <__isoc99_scanf@plt>

           |<=- 0x1c-=>|+--- ebp
top                    v
[          [~~~~>  ]   ][fp][ra]
```

Reminder: Exploiting crackme0x00

```
                |<=- 0x1c -=>|+--- ebp
top                          v
[           [~~~~>  ]   ][fp][ra]
            AAAABBBB.....GGGGHHHH
```

crackme0x00 in C


```c
int main(int argc, char *argv[])
{
  char buf[16];
  printf("IOLI Crackme Level 0x00\n");
  printf("Password:");

  // Q. How to fix this?
  scanf("%s", buf);

  if (!strcmp(buf, "250382"))
    printf("Password OK :)\n");
  else
    printf("Invalid Password!\n");
  return 0;
}
```

How to fix crackme0x00's bug?


```
// NOTE. 15 not 16
scanf("%15s", buf);

// NOTE. GCC allocates memory for char *buf,
// which requires a manual free()
scanf("%as", &buf);
```

StackGuard ('1998) vs Phrack ('1996)

StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks*

Crispin Cowan, Calton Pu, Dave Maier, Heather Hinton,† Jonathan Walpole,
Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle and Qian Zhang
Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology
immunix-request@cse.ogi.edu, http://cse.ogi.edu/DISC/projects/immunix

Abstract

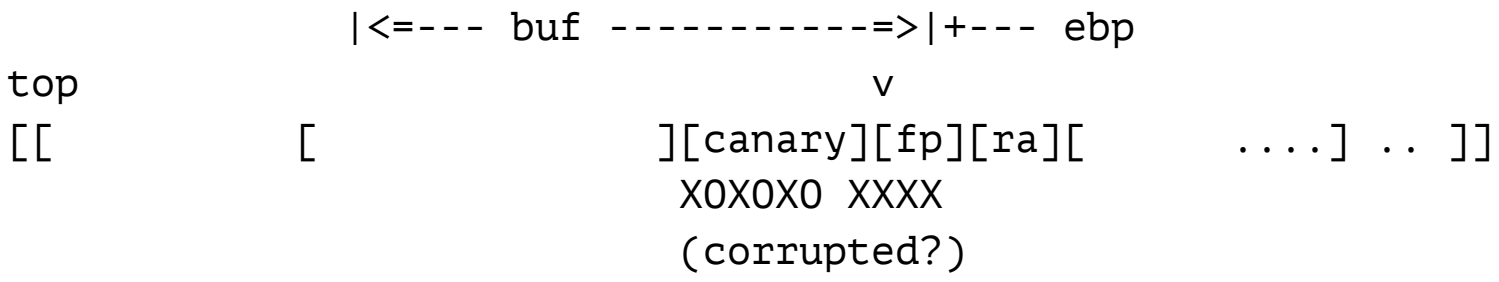
This paper presents a systematic solution to the persistent problem of buffer overflow attacks. Buffer overflow attacks gained notoriety in 1988 as part of the Morris Worm incident on the Internet. While it is fairly simple to fix individual buffer overflow vulnerabilities, buffer

1 Introduction

This paper presents a systematic solution to the persistent problem of buffer overflow attacks. Buffer overflow attack gained notoriety in 1988 as part of the Morris Worm incident on the Internet [23]. Despite the fact that fixing individual buffer overflow vulnerabilities is fairly simple, buffer overflow attacks continue to this day, as reported in the SANS Network Security Digest:

Core Idea of Stack Protector

- Use a “canary” value as an indicator of the integrity of fp/ra



Where does Stack Canary come from?

- `gs` : thread local storage
  - Check value in pwndbg: `gsbase`
- Disarming stack canary by overwriting it

```
            11fd:       55                      push   ebp
            11fe:       89 e5                   mov    ebp,esp
            1200:       56                      push   esi
            1201:       53                      push   ebx
            1202:       83 ec 18                sub    esp,0x18
            1205:       e8 f6 fe ff ff          call   1100
<__x86.get_pc_thunk.bx>
            120a:       81 c3 f6 2d 00 00       add    ebx,0x2df6
            1210:       8b 45 0c                mov    eax,DWORD PTR [ebp+0xc]
            1213:       89 45 e0                mov    DWORD PTR [ebp-
0x20],eax
            1216:       65 a1 14 00 00 00       mov    eax,gs:0x14
```

Subtle Design Choices for the Stack Canary

- Where to put? (e.g., right above ra? fp? local vars?)
- Which value should I use? (e.g., secrete? random? per exec? per func?)
- How to check its integrity? (e.g., xor? cmp?)
- What to do after you find corrupted? (e.g., crash? report?)

DEMO: GCC's Stack Protector

- Check `Makefile`
- Check compilation options
- Run `diff.sh`

In-class Tutorial

- Step 1: Understanding GCC's Stack Protector
- Step 2: Let's exploit 0xdeadbeef canary!

```
$ ssh lab04@54.88.195.85
Password: xxxxxxxx

$ cd tut04-ssp
$ cat README
```

References

- Bypassing StackShield