Buffer Overflows: What They Are, and How to Avoid Them

Ricardo J. Rodríguez

All wrongs reversed

rj.rodriguez@unileon.es ※ @RicardoJRdez ※ www.ricardojrodriguez.es

Research Institute of Applied Sciences in Cybersecurity
University of León, Spain

April 28, 2015

Mundo Hacker Day 2015
Madrid (España)

$whoami

- Ph.D. on Comp. Sci. (Univ. of Zaragoza, Spain) (2013)
- Senior Researcher at University of León (Spain)

RIASC

$whoami

- Ph.D. on Comp. Sci. (Univ. of Zaragoza, Spain) (2013)
- Senior Researcher at University of León (Spain)
  - Performance and safety analysis on critical, complex systems
  - Model-based security analysis
  - Advanced malware analysis
  - NFC security

RIASC

$whoami

- Ph.D. on Comp. Sci. (Univ. of Zaragoza, Spain) (2013)
- Senior Researcher at University of León (Spain)
  - Performance and safety analysis on critical, complex systems
  - Model-based security analysis
  - Advanced malware analysis
  - NFC security
- Trainer at NcN, RootedCON, HIP
- Speaker at NcN, HackLU, RootedCON, STIC CCN-CERT, MalCON, HIP, HITB...

RIASC

What is a BOF? (I)

```c
void readName()
{
  char username[256];
  printf("Username: ");
  scanf("%s", username);
}
```

What is a BOF? (I)

```
void readName()
{
  char username[256];
  printf("Username: ");
  scanf("%s", username);
}
```

```
void copyBuffers(char *org, char *dst)
{
  char buffer[5000];
  strcpy(buffer, org);
  // Do some stuff into your buffer
  strcpy(dst, buffer);
}
```

What is a BOF? (I)

```c
void readName()
{
  char username[256];
  printf("Username: ");
  scanf("%s", username);
}
```

```c
void copyBuffers(char *org, char *dst)
{
  char buffer[5000];
  strcpy(buffer, org);
  // Do some stuff into your buffer
  strcpy(dst, buffer);
}
```

Buffer Overflow (BOF)

- Memory zone overflow

What is a BOF? (I)

```c
void readName()
{
  char username[256];
  printf("Username: ");
  scanf("%s", username);
}
```

```c
void copyBuffers(char *org, char *dst)
{
  char buffer[5000];
  strcpy(buffer, org);
  // Do some stuff into your buffer
  strcpy(dst, buffer);
}
```

Buffer Overflow (BOF)

- Memory zone overflow
- It has consequences: Arbitrary code execution
  - Any code can be illegitimately forced to execute by an attacker (!)

What is a BOF? (I)

```c
void readName()
{
  char username[256];
  printf("Username: ");
  scanf("%s", username);
}
```

```c
void copyBuffers(char *org, char *dst)
{
  char buffer[5000];
  strcpy(buffer, org);
  // Do some stuff into your buffer
  strcpy(dst, buffer);
}
```

Buffer Overflow (BOF)

- Memory zone overflow
- It has consequences: Arbitrary code execution
  - Any code can be illegitimately forced to execute by an attacker (!)
- Is it used?
  - Common attack vector for malware

What is a BOF? (II)

Anything else?

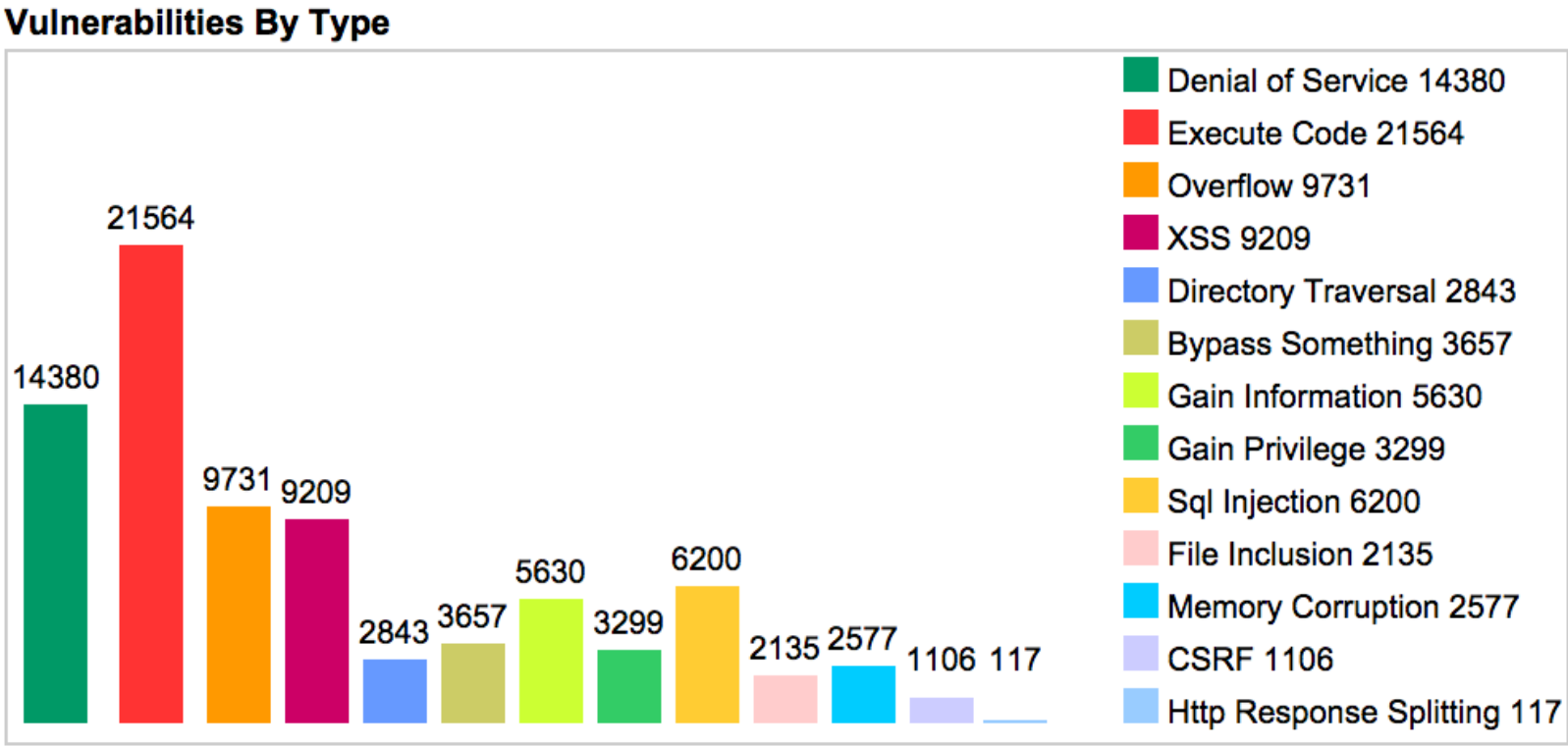
- Causes DoS
  - Application ends unexpectedly (it crashes)

What is a BOF? (II)

Anything else?

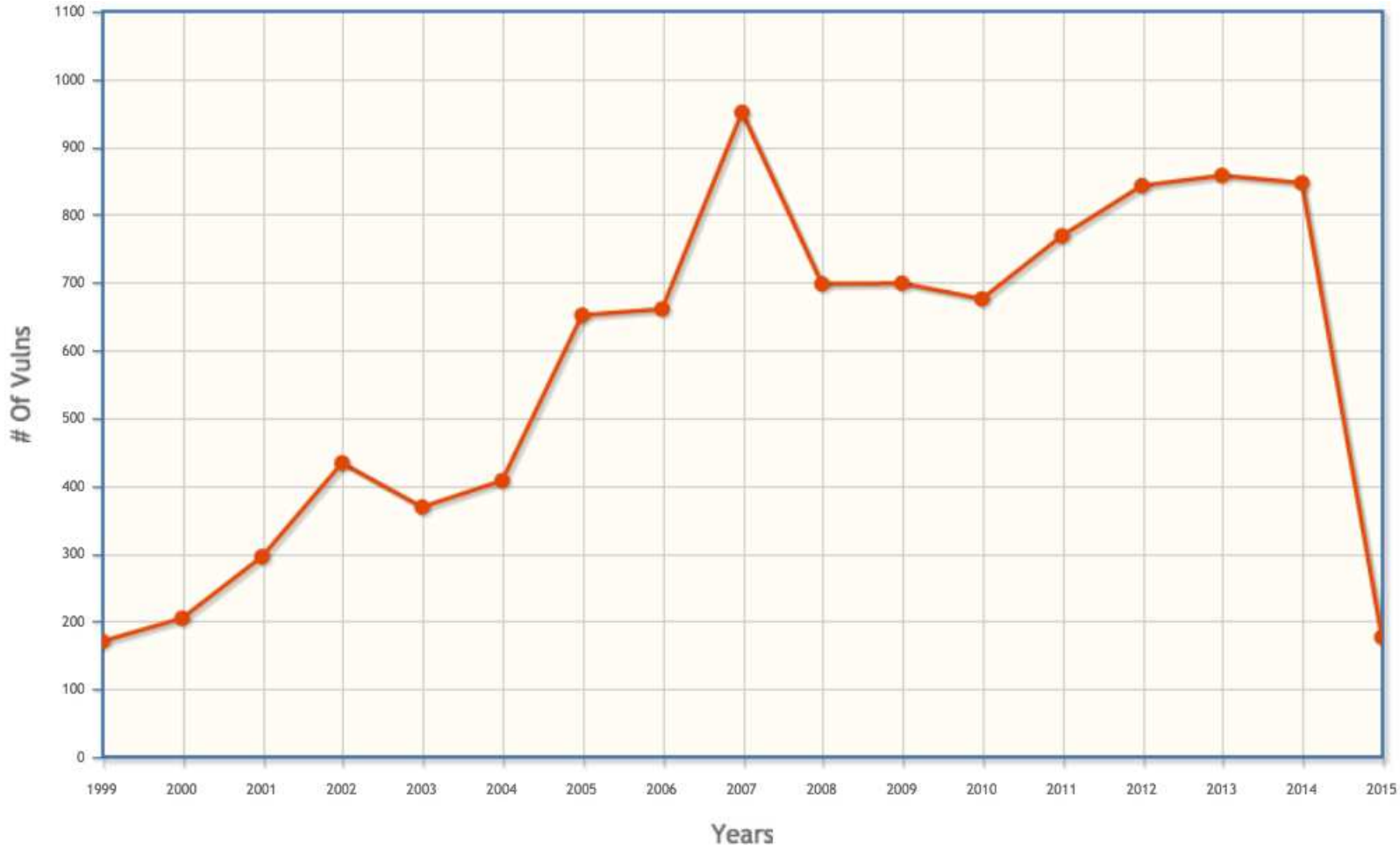
- Causes DoS
  - Application ends unexpectedly (it crashes)
- Wikipedia definition (overflow):
  - *"a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory. This is a special case of violation of memory safety' '*
- Problem trending is growing

What is a buffer overflow BOF? (III)



(Image source: `www.cvedetails.com`, date from 1999 to 2015)

What is a BOF? (IV)



(Image source: `www.cvedetails.com`, date from 1999 to 2015)

What is a BOF? (V)

Overflow types

- Stack-based BOF
  - CPU stack: Local variables storage, procedure parameters...
  - Control-flow execution data
    - Return addresses
    - Exception handlers

What is a BOF? (V)

Overflow types

- Stack-based BOF
  - CPU stack: Local variables storage, procedure parameters...
  - Control-flow execution data
    - Return addresses
    - Exception handlers
  - Consequences: Control-flow hijacking $\rightarrow$ an attacker controls what is going to be executed

What is a BOF? (V)

Overflow types

- Stack-based BOF
  - CPU stack: Local variables storage, procedure parameters...
  - Control-flow execution data
    - Return addresses
    - Exception handlers
  - Consequences: Control-flow hijacking $\rightarrow$ an attacker controls what is going to be executed
- Heap-based BOF
  - Overwriting of allocated memory (`malloc`, `allocate`)

What is a BOF? (V)

Overflow types

- Stack-based BOF
  - CPU stack: Local variables storage, procedure parameters...
  - Control-flow execution data
    - Return addresses
    - Exception handlers
  - Consequences: Control-flow hijacking $\rightarrow$ an attacker controls what is going to be executed
- Heap-based BOF
  - Overwriting of allocated memory (`malloc`, `allocate`)
  - Consequences: Memory corruption, code execution
- ...

What is a BOF? (VI)

Overflow types

- ...
- Off-by-one
  - A loop takes $(n-1)$ steps instead of n steps
  - Consequences: Control-flow register may be rewritten (1 byte)

What is a BOF? (VI)

Overflow types

- ...
- Off-by-one
  - A loop takes $(n-1)$ steps instead of n steps
  - Consequences: Control-flow register may be rewritten (1 byte)
- Buffer Overrun
  - Bottleneck on memory blocks when using CD/DVD writers
  - Buffer overflow $\rightarrow$ data is corrupted $\rightarrow$ CD/DVD useless

What is a BOF? (VI)

Overflow types

- ...
- Off-by-one
  - A loop takes $(n-1)$ steps instead of n steps
  - Consequences: Control-flow register may be rewritten (1 byte)
- Buffer Overrun
  - Bottleneck on memory blocks when using CD/DVD writers
  - Buffer overflow $\rightarrow$ data is corrupted $\rightarrow$ CD/DVD useless
- Integer OF

What is a BOF? (VI)

Overflow types

- ...
- Off-by-one
  - A loop takes $(n-1)$ steps instead of n steps
  - Consequences: Control-flow register may be rewritten (1 byte)
- Buffer Overrun
  - Bottleneck on memory blocks when using CD/DVD writers
  - Buffer overflow $\rightarrow$ data is corrupted $\rightarrow$ CD/DVD useless
- Integer OF

In this talk, we focus on Stack-based BOF

What is a BOF? (VII)

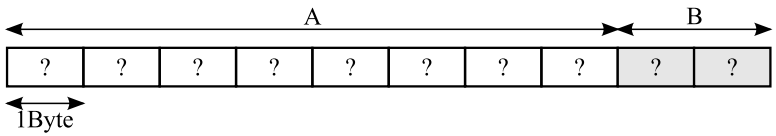
```
char           A[8];
unsigned short B;
```

- Variable A: 8B (1 char $\rightarrow$ 1B)
- Variable B: 2B
  - No initialized

What is a BOF? (VII)

```
char           A[8];
unsigned short B;
```

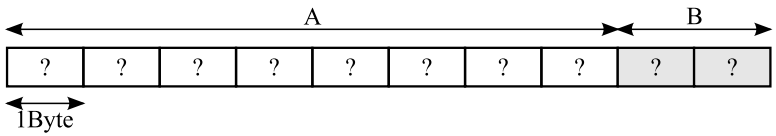
- Variable A: 8B (1 char $\rightarrow$ 1B)
- Variable B: 2B
  - No initialized



What is a BOF? (VII)

```
char           A[8];
unsigned short B;
```

- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No initialized



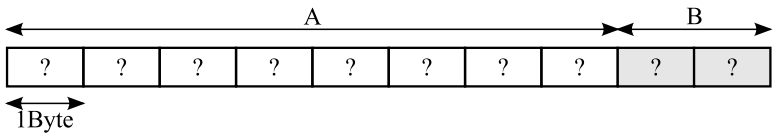
- Let's copy a string to A...

```
strcpy(A, "cadena");
```

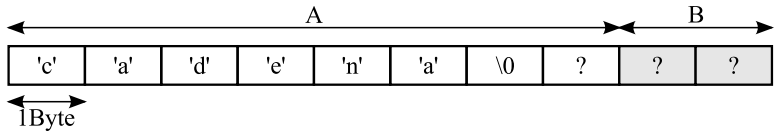
What is a BOF? (VII)

```
char           A[8];
unsigned short B;
```

- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No initialized



- What is the memory content?



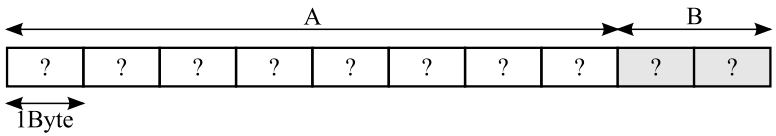
- Let's copy a string to A...

```
strcpy(A, "cadena");
```

What is a BOF? (VII)

```
char           A[8];
unsigned short B;
```

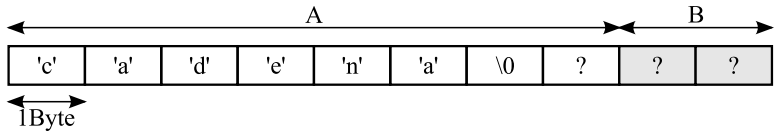
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No initialized



- Let's copy a string to A...

```
strcpy(A, "cadena");
```

- What is the memory content?



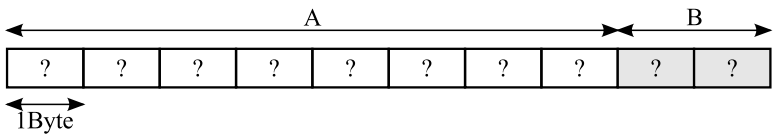
- What if we copy a longer string?

```
strcpy(A, "cadena larga");
```

What is a BOF? (VII)

```
char           A[8];
unsigned short B;
```

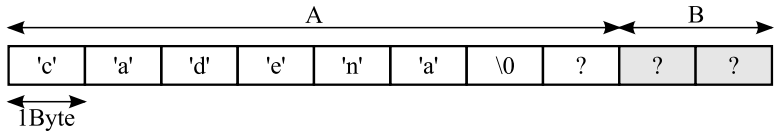
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No initialized



- Let's copy a string to A...

```
strcpy(A, "cadena");
```

- What is the memory content?



- What if we copy a longer string?

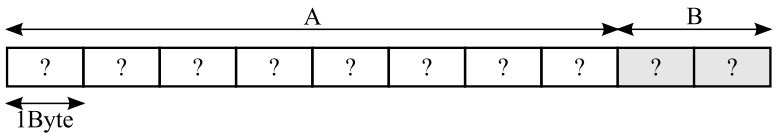
```
strcpy(A, "cadena larga");
```

- What is the memory content?

What is a BOF? (VII)

```
char           A[8];
unsigned short B;
```

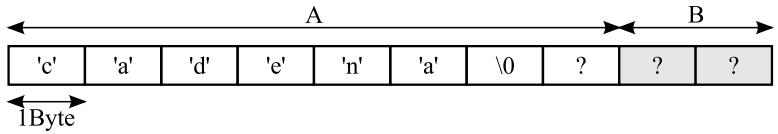
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No initialized



- Let's copy a string to A...

```
strcpy(A, "cadena");
```

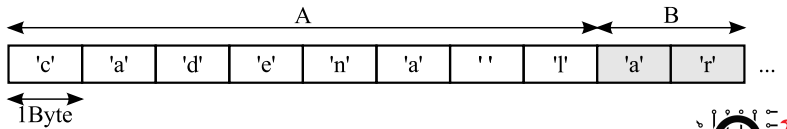
- What is the memory content?



- What if we copy a longer string?

```
strcpy(A, "cadena larga");
```

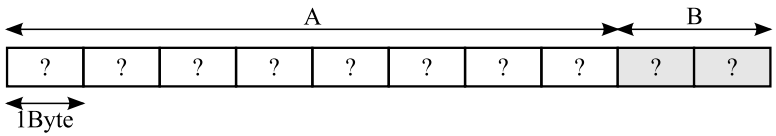
- What is the memory content?



What is a BOF? (VII)

```
char           A[8];
unsigned short B;
```

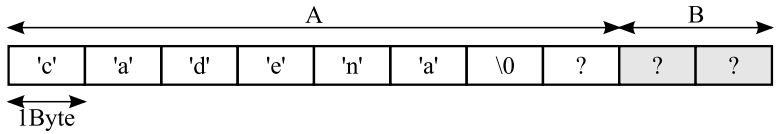
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No initialized



- Let's copy a string to A...

```
strcpy(A, "cadena");
```

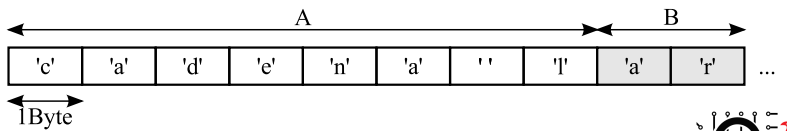
- What is the memory content?



- What if we copy a longer string?

```
strcpy(A, "cadena larga");
```

- What is the memory content?



Overwriting adjacent memory locations

Stack-based BOFs: From theory to practice (I)

Stack-based BOF

- Stack space: Local variables storage

Stack-based BOFs: From theory to practice (I)

Stack-based BOF

- Stack space: Local variables storage
- Data to control execution flow
  - Return addresses
  - Exception handlers

Stack-based BOFs: From theory to practice (I)

Stack-based BOF

- Stack space: Local variables storage
- Data to control execution flow
  - Return addresses
  - Exception handlers
- Consequences: control-flow hijacking $\rightarrow$ an attacker controls what is going to be executed

Stack-based BOFs: From theory to practice (II)

Return to the *classic BOF* (CWE-120)

- `http://cwe.mitre.org/data/definitions/120.html`
- *"the program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow."*

Stack-based BOFs: From theory to practice (II)

Return to the *classic BOF* (CWE-120)

- `http://cwe.mitre.org/data/definitions/120.html`
- *"the program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow."*
- Common exploitable functions (C language)
  - `strcpy()`, `strcat()`
  - `scanf()`, `gets()`
  - `printf()` family: `sprintf()`, `vsprintf()`, ...
  - `https://security.web.cern.ch/security/recommendations/en/codetools/c.shtml`

```
void readCredentials()
{
        /* Create an array for storing
                some dummy data */
        char username[16];
        printf("Enter your username for login, and
                    then press <Enter>: ");
        scanf("%s", username);
        printf("Hi %s, welcome back!
                    Well coding!\n", username);
        return;
}
```

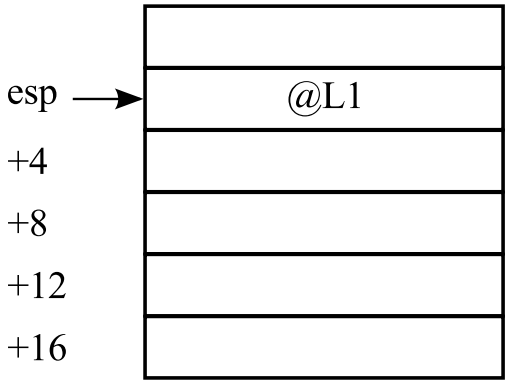
```c
void readCredentials()
{
        /* Create an array for storing
                some dummy data */
        char username[16];
        printf("Enter your username for login, and
                    then press <Enter>: ");
        scanf("%s", username);
        printf("Hi %s, welcome back!
                    Well coding!\n", username);
        return;
}
```

```
LC0: .ascii "Enter your username for login, and ... \0"
LC1: .ascii "%s\0"
LC2: .ascii "Hi %s, welcome back! Well coding!\12\0"
        .text
_readCredentials:
        push    ebp
        mov     ebp, esp
        sub     esp, 40
        mov     DWORD PTR [esp], OFFSET FLAT:LC0
        call    _printf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC1
        call    _scanf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC2
        call    _printf
        leave
        ret
L1:
```

```c
void readCredentials()
{
        /* Create an array for storing
                some dummy data */
        char username[16];
        printf("Enter your username for login, and
                    then press <Enter>: ");
        scanf("%s", username);
        printf("Hi %s, welcome back!
                    Well coding!\n", username);
        return;
}
```

```
LC0: .ascii "Enter your username for login, and ... \0"
LC1: .ascii "%s\0"
LC2: .ascii "Hi %s, welcome back! Well coding!\12\0"
        .text
_readCredentials:
        push    ebp
        mov     ebp, esp
        sub     esp, 40
        mov     DWORD PTR [esp], OFFSET FLAT:LC0
        call    _printf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC1
        call    _scanf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC2
        call    _printf
        leave
        ret
L1:
```

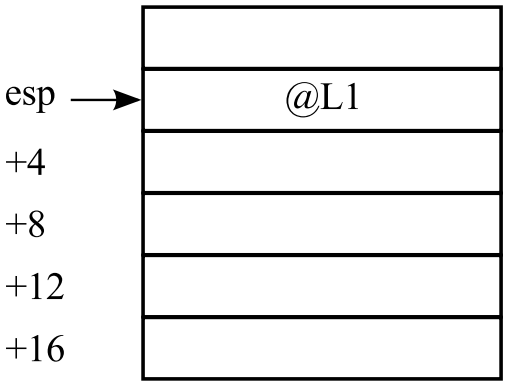
```c
void readCredentials()
{
        /* Create an array for storing
                some dummy data */
        char username[16];
        printf("Enter your username for login, and
                then press <Enter>: ");
        scanf("%s", username);
        printf("Hi %s, welcome back!
                Well coding!\n", username);
        return;
}
```

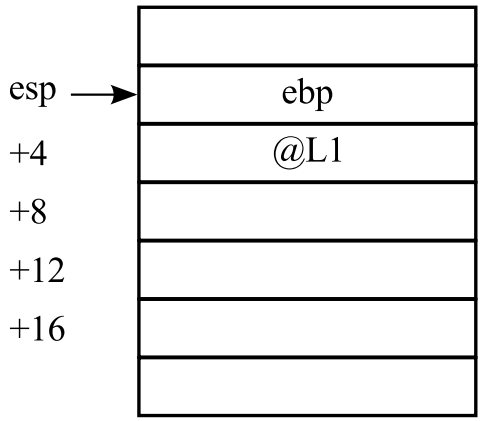



```
LC0: .ascii "Enter your username for login, and ... \0"
LC1: .ascii "%s\0"
LC2: .ascii "Hi %s, welcome back! Well coding!\12\0"
        .text
_readCredentials:
        push    ebp
        mov     ebp, esp
        sub     esp, 40
        mov     DWORD PTR [esp], OFFSET FLAT:LC0
        call    _printf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC1
        call    _scanf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC2
        call    _printf
        leave
        ret
L1:
```

```c
void readCredentials()
{
        /* Create an array for storing
                some dummy data */
        char username[16];
        printf("Enter your username for login, and
                then press <Enter>: ");
        scanf("%s", username);
        printf("Hi %s, welcome back!
                Well coding!\n", username);
        return;
}
```

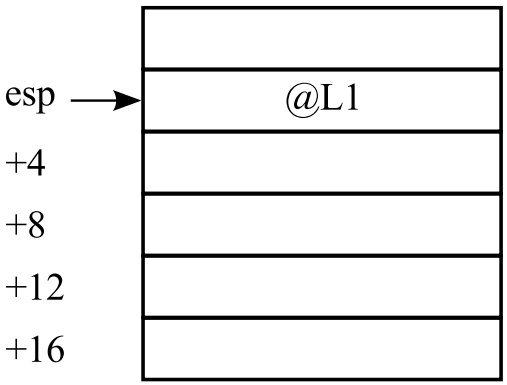
```
LC0: .ascii "Enter your username for login, and ... \0"
LC1: .ascii "%s\0"
LC2: .ascii "Hi %s, welcome back! Well coding!\12\0"
        .text
_readCredentials:
        push    ebp
        mov     ebp, esp
        sub     esp, 40
        mov     DWORD PTR [esp], OFFSET FLAT:LC0
        call    _printf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC1
        call    _scanf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC2
        call    _printf
        leave
        ret
L1:
```

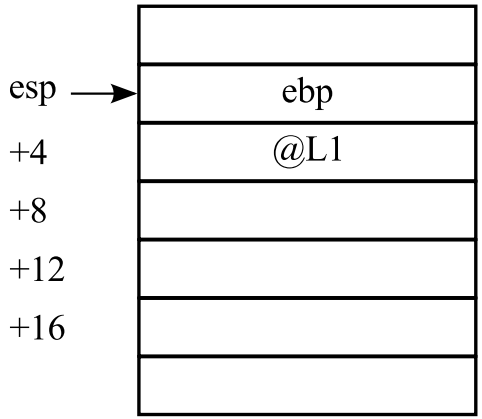


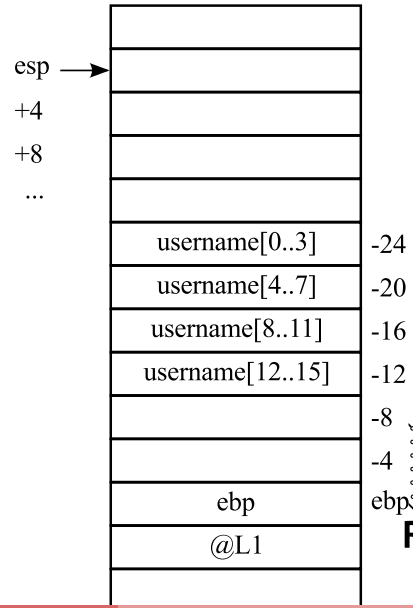


```c
void readCredentials()
{
        /* Create an array for storing
                some dummy data */
        char username[16];
        printf("Enter your username for login, and
                then press <Enter>: ");
        scanf("%s", username);
        printf("Hi %s, welcome back!
                Well coding!\n", username);
        return;
}
```

```
LC0: .ascii "Enter your username for login, and ... \0"
LC1: .ascii "%s\0"
LC2: .ascii "Hi %s, welcome back! Well coding!\12\0"
        .text
_readCredentials:
        push    ebp
        mov     ebp, esp
        sub     esp, 40
        mov     DWORD PTR [esp], OFFSET FLAT:LC0
        call    _printf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC1
        call    _scanf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC2
        call    _printf
        leave
        ret
L1:
```







It's demo time!

Mechanisms to Avoid Stack-based BOFs: Brief Summary

- Stack Cookies (aka Stack Canaries)
  - Compiler flag (`/GSswitch`)

Mechanisms to Avoid Stack-based BOFs: Brief Summary

- Stack Cookies (aka Stack Canaries)
  - Compiler flag (`/GSswitch`)
- SafeSEH / SEHOP (Structured Exception Handler Overwrite Protection)
  - Compiler flag (`/safeSEH`) (out of scope in this talk!)

Mechanisms to Avoid Stack-based BOFs: Brief Summary

- Stack Cookies (aka Stack Canaries)
  - Compiler flag (`/GSswitch`)
- SafeSEH / SEHOP (Structured Exception Handler Overwrite Protection)
  - Compiler flag (`/safeSEH`) (out of scope in this talk!)
- Data Execution Prevention (DEP) (aka Write or eXecute only mode, $W \oplus X$)
  - Operating System / Architecture supported
- Address Space Layout Randomization (ASLR)
  - Operating System / Compiler flag `/DYNAMICBASE`

Conclusions (I)

- Programming bugs may lead in exploitable BOFs
- Several protection mechanisms exist:
  - Compiler flags (/GS, /SafeSEH, /NXCOMPAT, /DYNAMICBASE)
  - Operating System/Architecture (SEHOP, Hardware-DEP, ASLR)
  - Commercial/Free third-party libraries
    (`http://en.wikipedia.org/wiki/Buffer_overflow_protection`)
- Evasion techniques for these protections are well-known
  - Isolated: Makes the exploit process more difficult to achieve
  - Combined: Better protection is guaranteed

Conclusiones (II)

Take-Home Message

Code (and compile) safely!

Conclusiones (II)

Take-Home Message

Code (and compile) safely!

Final recommendations

- Make use of safe functions
- Compile with all available protection flags activated
- In all files!

Further Readings

- Corelan EWT, https://www.corelan.be/index.php/category/security/exploit-writing-tutorials/
- Wikipedia, http://en.wikipedia.org/wiki/Buffer_overflow
- CVE details, http://www.cvedetails.com

- *Practical Malware Analysis*, M. Sikorski, A. Honig, NoStarch, 2012
- *Malware Analyst's Cookbook*, M.H. Ligh, S. Adair, B. Hartstein, M. Richard, Wiley, 2011
- *A Guide to Kernel Exploitation: Attacking the Core*, E. Perla, M. Oldani, Elsevier, 2011
- *Software Security: Building Security In*, G. McGraw, Addison Wesley, 2006
- *Reversing: Secrets of Reverse Engineering*, E. Eilam, Wiley, 2005
- *The Art of Computer Virus Research and Defense*, P. Szor, Addison Wesley, 2005

Agenda

1. What is a buffer overflow (BOF)?
2. Stack-based BOFs: From theory to practice
3. Mechanisms to Avoid Stack-based BOFs
4. Conclusions
5. Further Readings

Buffer Overflows: What They Are, and How to Avoid Them

Ricardo J. Rodríguez

All wrongs reversed

rj.rodriguez@unileon.es ※ @RicardoJRdez ※ www.ricardojrodriguez.es

Research Institute of Applied Sciences in Cybersecurity
University of León, Spain

April 28, 2015

Mundo Hacker Day 2015
Madrid (España)