(In)secure software

Lecture plan

- Introduction and Attacker MO
- **Software** and web app security
- System security
- Crypto
- Network security
- Reactive security
- Outro

Today's agenda

- Part 1 – Some common and notable bugs
- Part 2 – Fuzzing, or how to automate bug finding

Definitions

- A **bug** is a fault in a program that produces an incorrect or unexpected result, or causes it to behave in unintended ways
  - Bugs are plentiful, but not all bugs put the program at risk
- A **vulnerability** is bug that manifests as an opportunity for malicious use of the program
  - Not all vulnerabilities can be exploited for evil - **exploitable** vulnerabilities are what matters
- Bottom line, **vulnerabilities** gets **exploited** to run code that downloads and installs **malware**, more or less

Case study

An Adobe Flash 0day is being actively exploited in the wild

Adobe plans to have a fix for the critical flaw next week.

DAN GOODIN - 2/3/2018, 12:34 AM

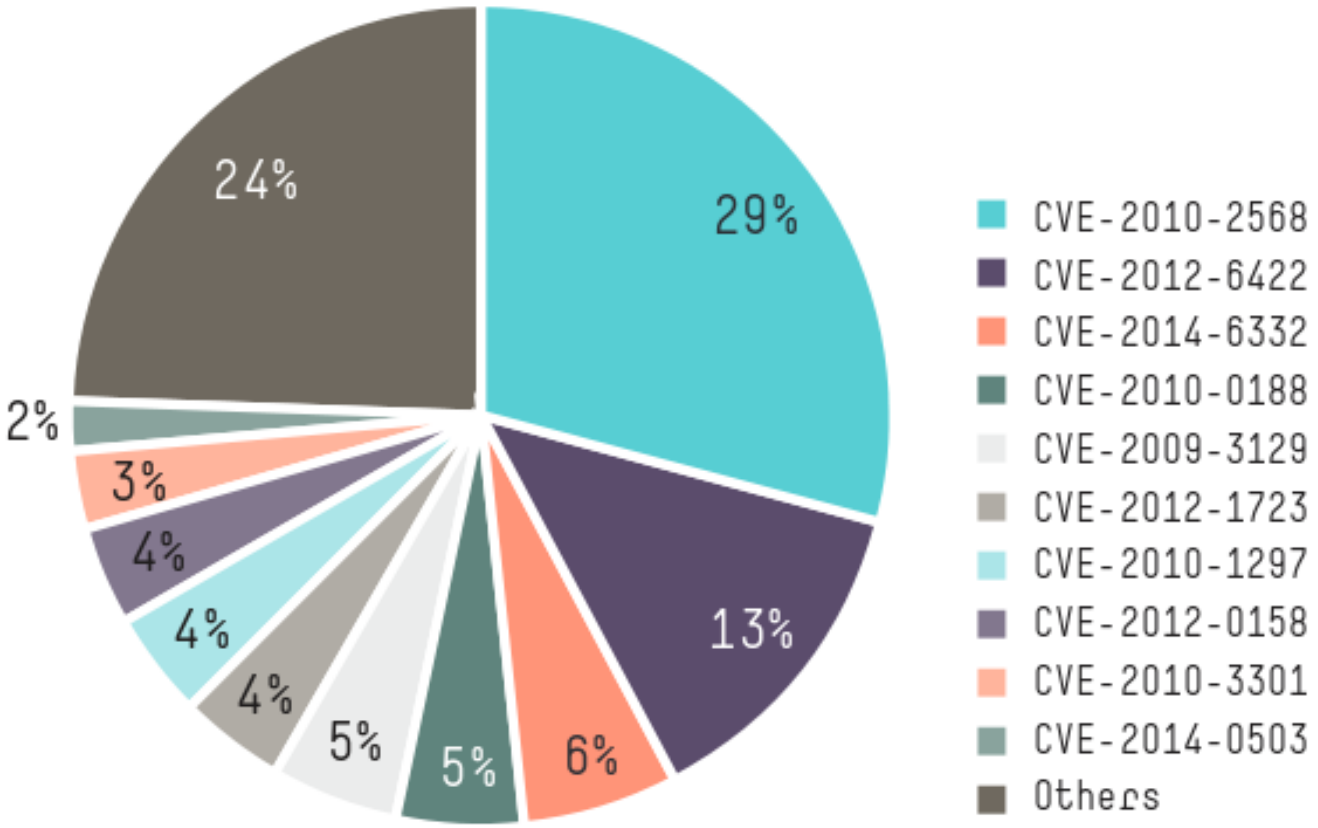
- https://arstechnica.com/information-technology/2018/02/theres-a-new-adobe-flash-0day-and-up-and-coming-hackers-are-exploiting-it/
- https://nvd.nist.gov/vuln/detail/CVE-2018-4877
- https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?name=CVE-2018-4877&vector=AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
- https://cwe.mitre.org/data/definitions/416.html
- https://en.wikipedia.org/wiki/Common_Vulnerability_Scoring_System#Impact_metrics

Most CVEs in 2017

	Product Name	Vendor Name	Product Type	Number of Vulnerabilities
1	Android	Google	OS	692
2	Linux Kernel	Linux	OS	407
3	Iphone Os	Apple	OS	344
4	Imagemagick	Imagemagick	Application	341
5	Windows 10	Microsoft	OS	255
6	Windows Server 2016	Microsoft	OS	239
7	Mac Os X	Apple	OS	236
8	Windows Server 2008	Microsoft	OS	231
9	Windows Server 2012	Microsoft	OS	223
10	Windows 7	Microsoft	OS	217

https://www.cvedetails.com/top-50-products.php?year=2017

Top CVEs exploited (2015)



Source: HP Cyber Risk Report 2016

Where’s the bug?

1.c

```
#include <stdio.h>

int main () {

    int i;

    printf("Enter a value: ");
    scanf("%d", &i);

    if (i < 0)
        goto fail;
    if (i > 100)
        goto fail;
        goto fail;
    if (i%2 == 0)
        goto fail;

    return;

fail:
    printf("Fail\n");
    return;
}
```

Problem

```
$ ./1.out
Enter a value: 2
Fail

$ ./1.out
Enter a value: 3
Fail
```

1fix.c

```
#include <stdio.h>

int main () {

    int i;

    printf("Enter a value: ");
    scanf("%d", &i);

    if (i < 0)
        goto fail;
    if (i > 100)
        goto fail;
        //goto fail;
    if (i%2 == 0)
        goto fail;

    return;

fail:
    printf("Fail\n");
    return;
}
```

Apple iOS Goto Fail

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

2.c

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

int main ()
{

   char buf[20] = "http://www.diku.dk";
   char shh[30] = "mumstheword";
   char out[64];
   int chars;

   printf("Buffer contents: %s\n", buf);

   printf("Chars to copy: ");
   scanf("%d", &chars);

   memcpy(out, buf, chars);

   printf("Copied: ");
   fwrite(out, chars, 1, stdout);
   printf("\n");

   return(0);
}
```

Problem

```
$ ./2.out
Buffer contents: http://www.diku.dk
Chars to copy: 12
Copied: http://www.d

$ ./2.out
Buffer contents: http://www.diku.dk
Chars to copy: 50
Copied: http://www.diku.dk?¨0L?H?¨mumstheword
```

Explanation

- `void * memcpy ( void * destination, const void * source, size_t num );`

  Copies num *bytes from the location pointed by* source *to the location pointed by* destination

- `memcpy(out, buf, chars);`

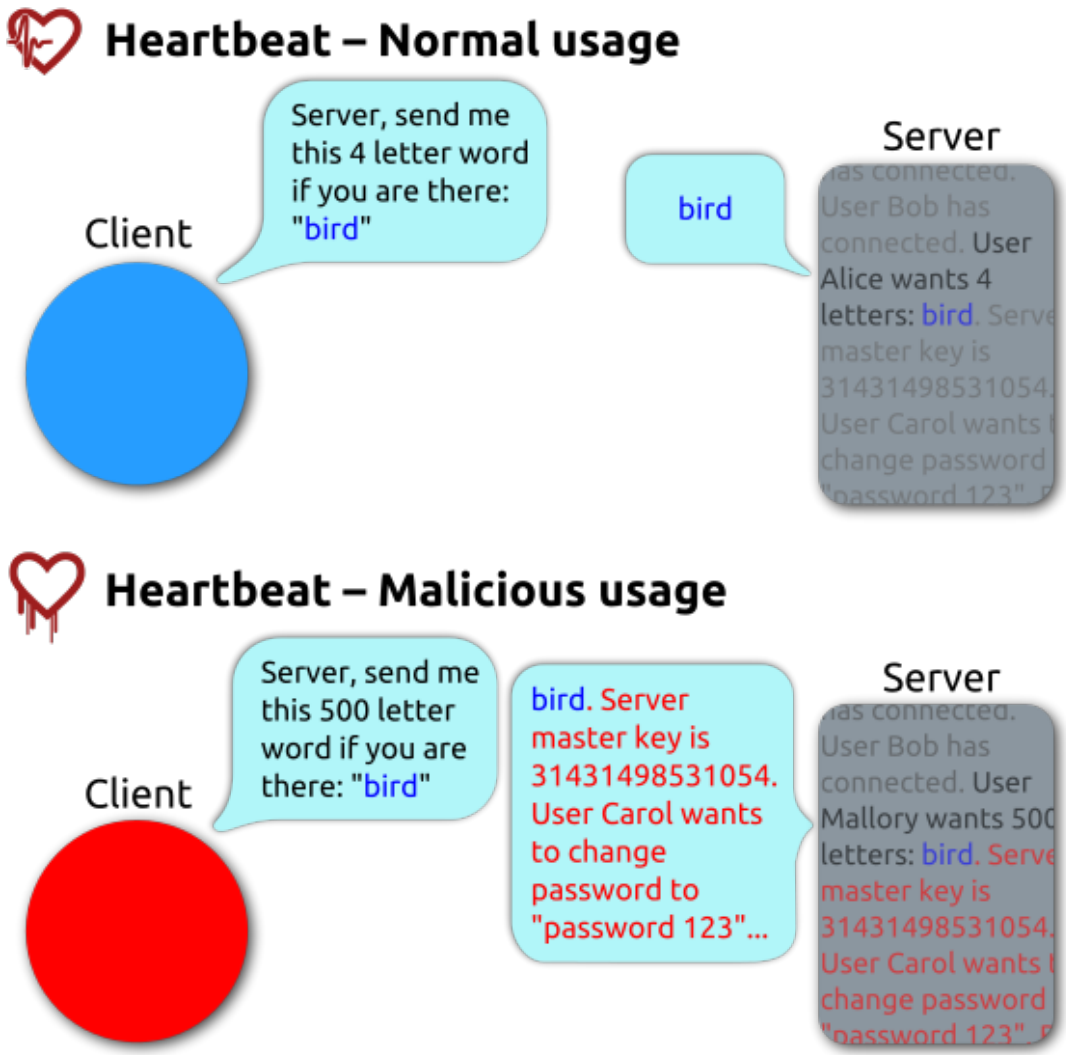
User inputted

2fix.c

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

int main ()
{

   char buf[20] = "http://www.diku.dk";
   char shh[30] = "mumstheword";
   char out[64];
   int chars;

   printf("Buffer contents: %s\n", buf);

   printf("Chars to copy: ");
   scanf("%d", &chars);
   if (chars > sizeof(buf)) chars = sizeof(buf);
   memcpy(out, buf, chars);

   printf("Copied: ");
   fwrite(out, chars, 1, stdout);
   printf("\n");

   return(0);
}
```

The Heartbleed Bug

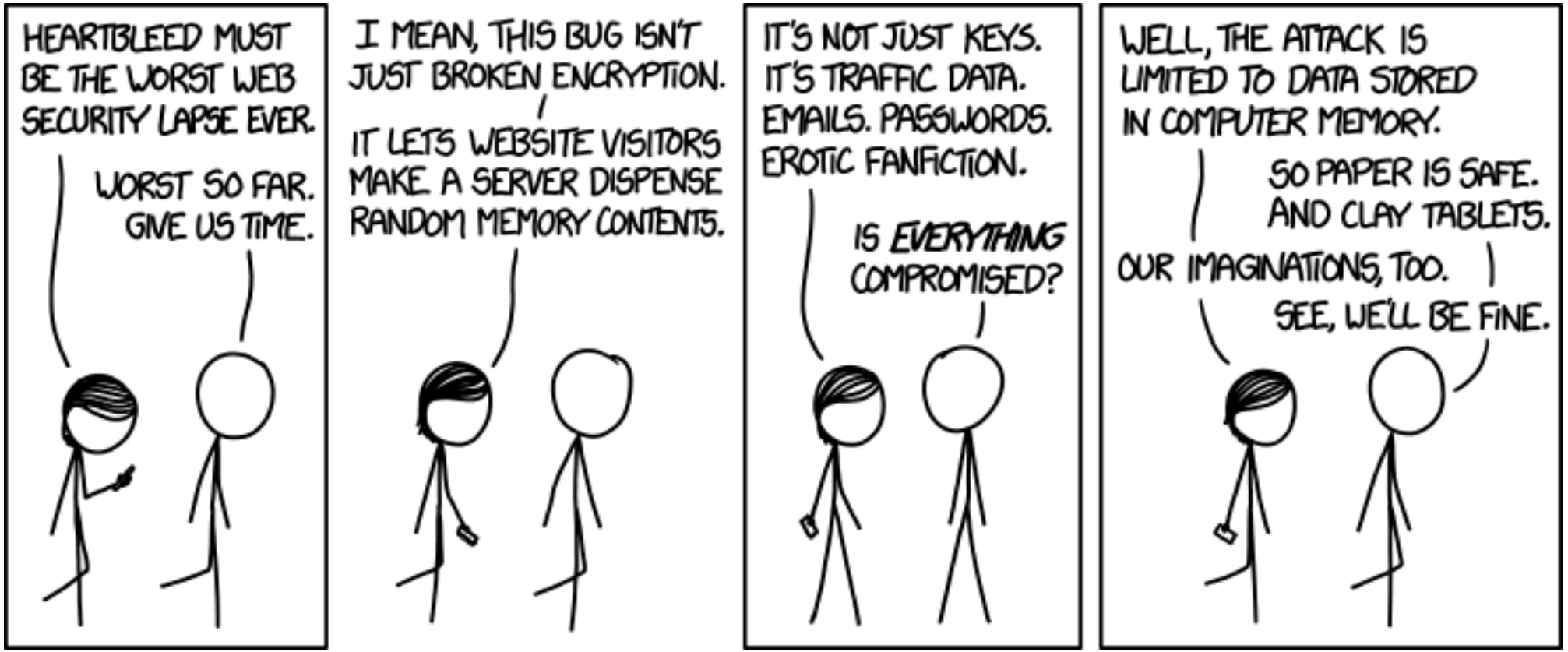
- Vulnerability in the popular OpenSSL cryptographic software library



What to do

- Simple fix
  - Add bounds check
- Knock-on effects
  - Change passwords
  - Revoke certificate
  - Generate new keys
  - Issue new certificate
  - Compromised?

You know it's bad when there's a XKCD



Shellshock

- A family of bugs in the Unix Bash shell

Shellshock

Command to set environment variable

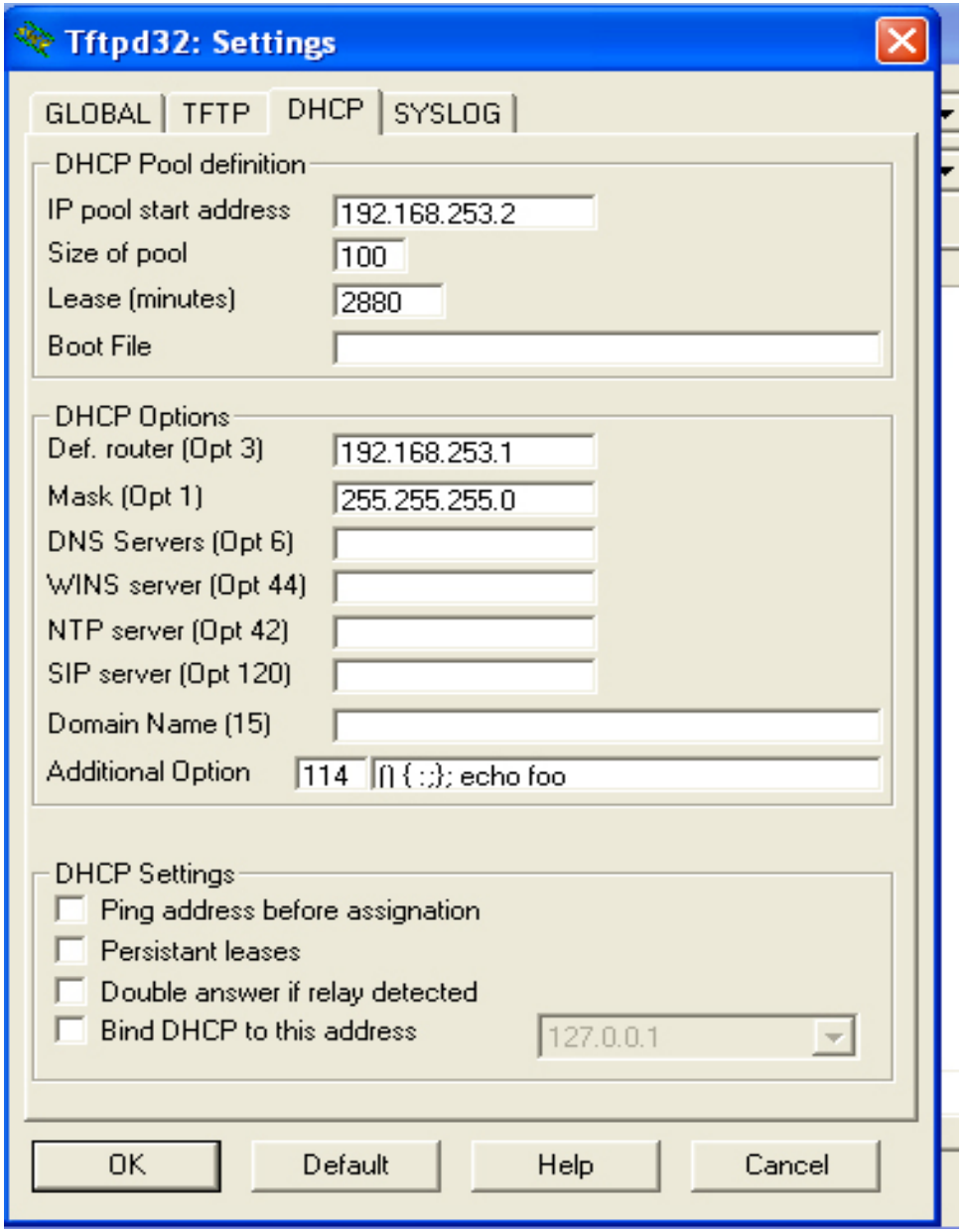
Real command

```
env x='() { :;}; echo vulnerable' bash -c "echo testing"
```

Tacked-on arbitrary command that will be executed by Bash

How to set the environment variable?

- Attack vectors include
  - CGI-based web server
  - OpenSSH server
  - Qmail server
  - DHCP clients



Heartbleed vs Shellshock

- Introduced 2011
- Found 2014
- Affects millions
- Easily exploitable
- Random reads

- Introduced 1981
- Found 2014
- Even more millions
- Easily exploitable
- Complete control

3.c

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

int main(int argc, char **argv)
{

   printf("Current time: ");
   fflush(stdout);
   system("date");
   return 0;

}
```

Problem

```
$ ./3.out
Current time: Sun May  1 23:47:47 CEST 2016

$ export PATH=`pwd`:$PATH
$ echo -e '#!/bin/sh\necho "Hello"' > date
$ chmod 700 date

$ ./3.out
Current time: Hello
```

3fix.c

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

int main(int argc, char **argv)
{

   printf("Current time: ");
   fflush(stdout);
   system("/bin/date");
   return 0;

}
```

Real-world example: PlugX

- PlugX drops
  - A legitimate NVIDIA file (NvSmart.exe)
  - A malicious DLL (NvSmartMax.dll)
- Normally, NvSmart.exe would load a legitimate NvSmartMax.dll
- But, if a (malicious) version the DLL file is located in the same directory, this will load instead

4.pl

```
#!/usr/bin/perl

open(FH, $ARGV[0]);

while(<FH>)
{
   print $_;
}

close(FH);
```

Problem

```
$ ./4.pl 4.pl
#!/usr/bin/perl

open(FH, $ARGV[0]);

while(<FH>)
{
   print $_;
}

close(FH);

$ ./4.pl 'ls -l 4.pl|'
-rwx------ 1 user user 79 May  1 10:45 4.pl
```

Explanaition

- According to the Perl documentation
  - If filename ends with a "|", filename is interpreted as a command which pipes output

4fix.pl

```
#!/usr/bin/perl

open(FH, "< ".$ARGV[0]); #force read open with '<'

while(<FH>)
{
   print $_;
}

close(FH);
```

5.c

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

int main (int argc, char **argv)
{

   char buffer[64];
   strncpy(buffer, argv[1], sizeof(buffer));

   printf("You entered: ");
   printf(buffer);
   printf("\n");
}
```

Problem

```
$ ./5.out A
You entered: A

$ ./5.out %s
You entered: You entered:

$ ./5.out %x
You entered: 510a2000

$ ./5.out %x%x
You entered: 437a00041569e0
```

5fix.c

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

int main (int argc, char **argv)
{

   char buffer[64];
   strncpy(buffer, argv[1], sizeof(buffer));

   printf("You entered: ");
   printf("%s", buffer);
   printf("\n");
}
```

Format string in sort.exe

C:\Windows\system32\cmd.exe

```
C:\>sort %x
291fe6Den angivne fil blev ikke fundet.

C:\>sort %x%x%x%x%x%x%x%x
41e372fffc064a2041e120016ae130104c0045Den angivne fil blev ikke fundet.

C:\>sort %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
4120eafffc064a5840e1b00161e1c02c4c00457825782578257825782578257825782578257825782578
25782564002e40e9c0016131217d5e98b726d918Den angivne fil blev ikke fundet.

C:\>_
```

6.c

```
#include <string.h>

void foo (char *bar)
{
   char  c[12];
   strcpy(c, bar);
}

int main (int argc, char **argv)
{
   foo(argv[1]);
}
```

Problem

```
$ ./6.out A

$ ./6.out AAAAAAAAAAAAAAA

$ ./6.out AAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
```

6fix.c

```
#include <string.h>

void foo (char *bar)
{
   char  c[12];
   strncpy(c, bar, sizeof(c));
}

int main (int argc, char **argv)
{
   foo(argv[1]);
}
```

Stack-based buffer overflow

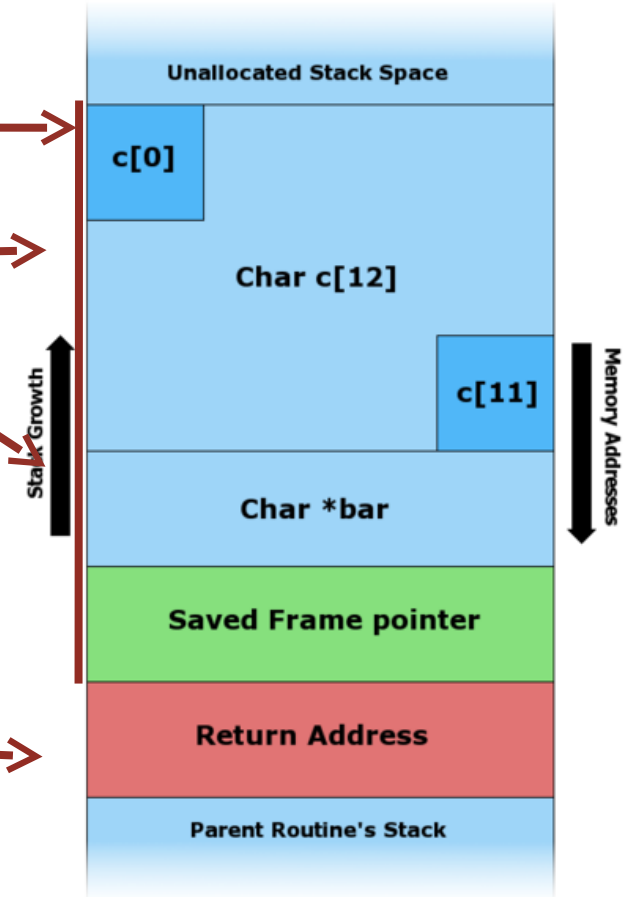
```
#include <string.h>

void foo (char *bar) {

    char  c[12];

    strcpy(c, bar);

}

int main (int argc, char
   **argv) {

    foo(argv[1]);

}
```

Entire stack frame for foo()

Local variable

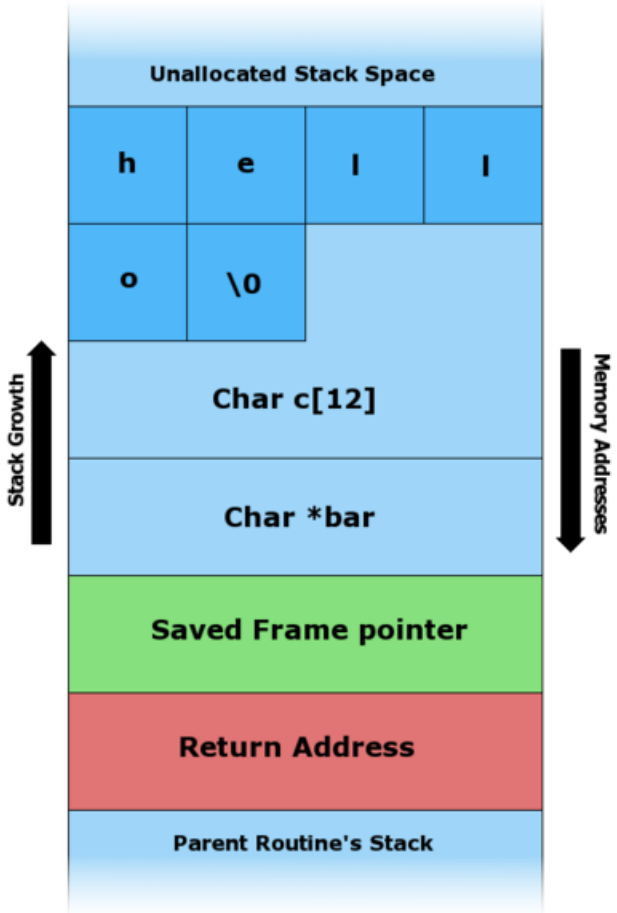
Function call args

Return addr to main()



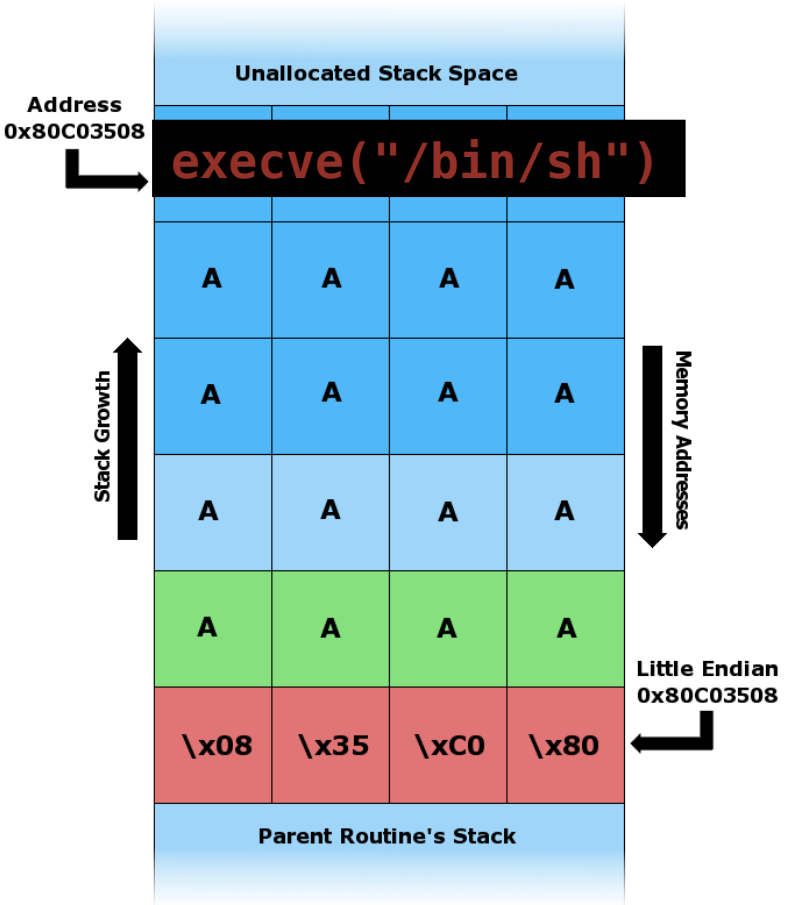
Stack-based buffer overflow

```
$ ./buffer1 hello
```



Stack-based buffer overflow

```
$ python -c 'print "A"
  * 24' | ./buffer1
Segmentation fault

$ python -c 'print "A"
  * 20+"\x08\x35\xc0\x80"'
 | ./buffer1
Segmentation fault
```



Shellcode

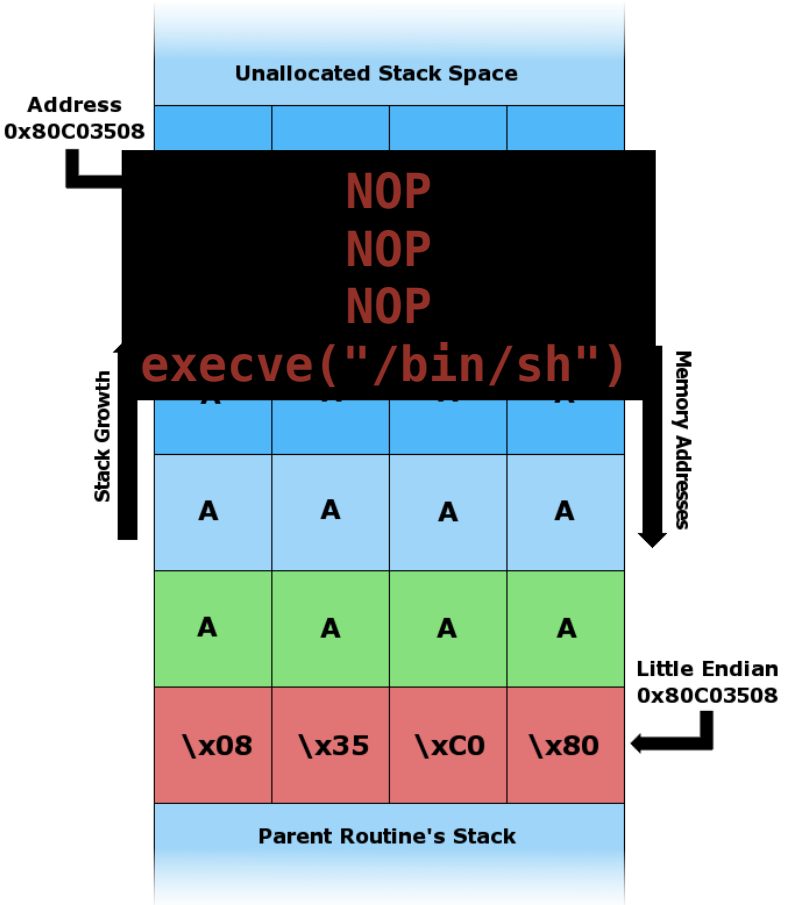
- `execve("/bin/sh")` is shellcode

- Shellcode is machine code used as the payload in the exploitation of a vulnerability to get a shell

- Processor architecture specific: x86, PowerPC, ARM, x64, etc.

NOP sled

- Improving the odds with a No Operation (NOP) slep



Some counter-measures

- Stack canaries
  - Check stack not altered when function returns
- Data execution prevention (DEP)
  - Prevent the execution of data on the stack or heap
- Address space layout randomization (ASLR)
  - Rearrange memory positions to make successful exploitation more difficult

Okay so you've found a bug

Vulnerability marketplace options