Buffer Overrun Vulnerabilities and Assignment 6 (The ‘B’ Attack)
Yet another character reading loop program ...

```c
#include <stdio.h>
int main(void)
{
    char name[12], c;
    int i = 0, magic = 42;
    printf("What is your name?\n");
    while ((c = getchar()) != '\n')
        name[i++] = c;
    name[i] = '\0';
    printf("Thank you, %s.\n", name);
    printf("The answer to life, the universe, "
    "and everything is %d\n", magic);
    return 0;
}
```

```
$ ./a.out
What is your name?
John Smith
Thank you, John Smith.
The answer to life, the universe, and everything is 42
```
Why People With Long Names Have Issues with Computer Systems

#include <stdio.h>
int main(void)
{
    char name[12], c;
    int i = 0, magic = 42;
    printf("What is your name?\n");
    while ((c = getchar()) != '\n')
        name[i++] = c;
    name[i] = '\0';
    printf("Thank you, %s.\n", name);
    printf("The answer to life, the universe, "
            "and everything is %d\n", magic);
    return 0;
}

$ ./a.out
What is your name?
Szymon Rusinkiewicz
Thank you, Szymon Rusinkiewicz.
The answer to life, the universe, and everything is 8020841

(Note: this is just the number that’s actually printed when you actually run the code. It’s not an attempt to Easter egg a phone number or anything like that. Please don’t try to call it. kthx)
When there are too many characters, program carelessly writes beyond space "belonging" to name.

- Overwrites other variables
- This is a buffer overrun, or stack smash
- The program has a security bug!

```
#include <stdio.h>
int main(void)
{
    char name[12], c;
    int i = 0, magic = 42;
    printf("What is your name?\n");
    while ((c = getchar()) != '\n')
    {
        name[i++] = c;
        name[i] = '\0';
    }
    name[i] = ' ';
    printf("Thank you, %s.\n", name);
    printf("The answer to life, the universe, "
            "and everything is %d\n", magic);
    return 0;
}
```
Buffer overrun can overwrite onto its caller function's stack frame!

#include <stdio.h>
int callee(void) {
    char name[12], c;
    int i = 0, magic = 42;
    printf("What is your name?\n");
    while ((c = getchar()) != '\n')
        name[i++] = c;
    name[i] = '\0';
    printf("Thank you, %s.\n", name);
    printf("The answer to life, the universe, "
        "and everything is %d\n", magic);
    return 0;
}
It Gets Worse...

And somewhere on caller's stack frame is the saved return address for that function ...

Buffer overrun can overwrite caller's return address!
  - Replacement value can be an invalid address, leading to a segfault, or ...

```c
#include <stdio.h>
int callee(void)
{
    char name[12], c;
    int i = 0, magic = 42;
    printf("What is your name?\n");
    while ((c = getchar()) != '\n')
        name[i++] = c;
    name[i] = '\0';
    printf("Thank you, %s.\n", name);
    printf("The answer to life, the universe, "
          "and everything is %d\n", magic);
    return 0;
}
```

SP

Return addr

name...

c

magic

i

old x30

overwritten with, e.g., NULL

Old SP

0
And somewhere on caller's stack frame is the saved return address for that function ...

Buffer overrun can overwrite caller's return address!

- Replacement value can be an invalid address, leading to a segfault, or it can cleverly cause unintended control flow!

#include <stdio.h>

int callee(void)
{
    char name[12], c;
    int i = 0, magic = 42;
    printf("What is your name?\n");
    while ((c = getchar()) != '\n')
    {
        name[i++] = c;
        name[i] = '\0';
    }
    printf("Thank you, %s.\n", name);
    printf("The answer to life, the universe, "
"and everything is %d\n", magic);
    return 0;
}
And somewhere on caller's stack frame is the saved return address for that function …

Buffer overrun can overwrite caller's return address!

- Replacement value can be an invalid address, leading to a segfault, or it can cleverly cause unintended control flow, or even cause arbitrary malicious code to run!

```
#include <stdio.h>
int callee(void)
{
    char name[12], c;
    int i = 0, magic = 42;
    printf("What is your name?\n");
    while ((c = getchar()) != '\n')
        name[i++] = c;
    name[i] = '\0';
    printf("Thank you, %s.\n", name);
    printf("The answer to life, the universe, "
        "and everything is %d\n", magic);
    return 0;
}
```
Attacking a Web Server

URLs
Input in web forms
Crypto keys for SSL
etc.

Client PC
Web Server

```c
for(i=0;p[i];i++)
    search[i]=p[i];
```

this is a really long search term that overflows a buffer
Attacking Everything in Sight

for (i=0; p[i]; i++)
important[i] = p[i];

Any application that ever sees input directly from the outside!

- E-mail clients
- PDF viewers
- Operating-system kernels
- TCP/IP Stack

- webp image library (9/2023)
- C/C++ MP4 video library (4/2023)
- OpenSSL crypto library (11/2022)
- Smart UPS devices (3/2022)
- Zoom (11/2021)
- VLC media player (1/2019)
- Nintendo Switch (4/2018)
- ...
Defenses Against This Attack

Best: program in languages that make array-out-of-bounds impossible (Java, python, C#, ML, ...)

But if you need to use C...
Defenses Against This Attack

In C: use discipline and software analysis tools to check bounds of array subscripts

**DESCRIPTION**
The `strcpy()` function copies the string pointed to by `src`, including the terminating null byte (`\0`), to the buffer pointed to by `dest`. The strings may not overlap, and the destination string `dest` must be large enough to receive the copy. *Beware of buffer overruns!* (See BUGS.)

**BUGS**
Never use `gets()`. Because it is impossible to tell without knowing the data in advance how many characters `gets()` will read, and because `gets()` will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use `fgets()` instead.

Augmented by OS- or compiler-level mitigations:

- Randomize initial stack pointer
- "No-execute" memory permission for sections other than `.text`
- "Canaries" at end of stack frames

None of these would have prevented the "Heartbleed" attack
Half a billion dollars worth of heartburn ...

Heartbeat – Normal usage

Client

Server, send me this 4 letter word if you are there: "bird"

bird

Heartbeat – Malicious usage

Client

Server, send me this 500 letter word if you are there: "bird"

Server

bird. Server master key is 31431498531054. User Bob has connected. User Alice wants 4 letters: bird. Server master key is 31431498531054. User Carol wants to change password to "password 123"...

Server

Wikipedian FenixFeather - Creative Commons Attribution-Share Alike 3.0 Unported
Assignment 6: Attack the “Grader” Program

```
enum {BUFSIZE = 48};
char grade = 'D';
char name[BUFSIZE];
...
int main(void)
{
    mprotect(...);
    getname();
    if (strcmp(name, "Andrew Appel") == 0)
        grade = 'B';
    printf("%c is your grade.\n", grade);
    printf("Thank you, %s.\n", name);
    return 0;
}
```

$ ./grader
What is your name?
Joe Student
D is your grade.
Thank you, Joe Student.
$ ./grader
What is your name?
Andrew Appel
B is your grade.
Thank you, Andrew Appel.
Assignment 6: Attack the “Grader” Program

/* Prompt for name and read it */
void getName() {
    printf("What is your name?\n");
    readString();
}

/* Read a string into name */
void readString() {
    char buf[BUFSIZE];
    int i = 0;
    int c;

    /* Read string into buf[] */
    for (;;) {
        c = fgetc(stdin);
        if (c == EOF || c == '\n')
            break;
        buf[i] = c;
        i++;
    }
    buf[i] = '\0';

    /* Copy buf[] to name[] */
    for (i = 0; i < BUFSIZE; i++)
        name[i] = buf[i];
}

Unchecked write to buffer!
Assignment 6: Attack the “Grader” Program

```c
enum {BUFSIZE = 48};
char grade = 'D';
char name[BUFSIZE];
...
int main(void)
{
    mprotect(...);
    getname();
    if (strcmp(name, "Andrew Appel") == 0)
        grade = 'B';
    printf("%c is your grade.\n", grade);
    printf("Thank you, %s.\n", name);
    return 0;
}
```

$ ./grader
What is your name?
Joe Student\0(#@&%*#&(*^!@%*!(&$
B is your grade.
Thank you, Joe Student.

Smash the stack!
Memory Map of STACK Section

SP

readString’s stackframe

getName’s stackframe

main’s stackframe

Keep writing past end of buf

Get to getName’s stackframe

What’s there?

getName’s saved x30! (somewhere on stack)

Overwrite it!

With what?
Assignment 6: Attack the “Grader” Program

```c
enum {BUFSIZE = 48};
char grade = 'D';
char name[BUFSIZE];
...
int main(void)
{
    mprotect(...);
    getname();
    if (strcmp(name, "Andrew Appel") == 0)
        grade = 'B';
    printf("%c is your grade.\n", grade);
    printf("Thank you, %s.\n", name);
    return 0;
}
```

$ ./grader
What is your name?
Joe Student
B is your grade.
Thank you, Joe Student.
Memory Map of TEXT Section

(readString)

<table>
<thead>
<tr>
<th>prolog</th>
<th>instrs...</th>
<th>instrs...</th>
</tr>
</thead>
<tbody>
<tr>
<td>instrs...</td>
<td>epilog</td>
<td>instrs...</td>
</tr>
<tr>
<td>instrs...</td>
<td>epilog</td>
<td>return</td>
</tr>
</tbody>
</table>

(getName)

<table>
<thead>
<tr>
<th>prolog</th>
<th>instrs...</th>
<th>instrs...</th>
</tr>
</thead>
<tbody>
<tr>
<td>instrs...</td>
<td>epilog</td>
<td>instrs...</td>
</tr>
<tr>
<td>instrs...</td>
<td>epilog</td>
<td>return</td>
</tr>
</tbody>
</table>

(main)

<table>
<thead>
<tr>
<th>prolog</th>
<th>instrs...</th>
<th>instrs...</th>
</tr>
</thead>
<tbody>
<tr>
<td>instrs...</td>
<td>epilog</td>
<td>instrs...</td>
</tr>
<tr>
<td>instrs...</td>
<td>epilog</td>
<td>return</td>
</tr>
</tbody>
</table>

(All of these instructions are actually machine code, not flattened C, of course!)

... checkappel:
  if (strcmp(name, "Andrew Appel") != 0)
    goto afterb
    grade = 'B' ← HERE!
  afterb:
    print ...
...

(22) (All of these instructions are actually machine code, not flattened C, of course!)
Construct Your Exploit String (createdatab.c)

1. Your name.
   • After all, the grader program’s last line of output must be: “Thank you, [your name].”

2. A null byte.
   • Otherwise, the grader program’s last line of output will be corrupted.

3. Filler to overrun until x30.
   • Presumably more null bytes are easiest, but easter eggs are fine.

4. The address of the target
   • The statement grade = 'B'.

fopen the file "dataB" and write your name into that file (e.g. with fprintf)

See “Writing Binary Data” precept handout. '\0' is just a single byte of binary data.

Address is a 64-bit (little-endian) unsigned integer: C unsigned long
Let's Not Get Thrown in Jail, Please

18 U.S. Code § 1030 - Fraud and related activity in connection with computers

(a) Whoever—

(1) having knowingly accessed a computer without authorization or exceeding authorized access, and by means of such conduct having obtained information that has been determined by the United States Government pursuant to an Executive order or statute to require protection against unauthorized disclosure for reasons of national defense or foreign relations, or any restricted data, as defined in paragraph y. of section 11 of the Atomic Energy Act of 1954, with reason to believe that such information so obtained could be used to the injury of the United States, or to the advantage of any foreign nation willfully publishes, transmits, or causes to be communicated, delivered, or transmitted, or attempts to communicate, deliver, or transmit, or publishes, transmits, or causes to be communicated, delivered, or transmitted, or attempts to communicate, deliver, or transmit, any such information, with the intent to damage the United States or to benefit any foreign nation, shall be fined under this title, or imprisoned not more than 10 years, or both.
Summary

• This lecture:
  • Buffer overrun attacks in general
  • Assignment 6 “B Attack” principles of operation

• Next precept:
  • Assignment 6 “B Attack” recap
  • Memory map using gdb
  • Writing binary data

• Final 2 lectures:
  • Assignment 6 “A Attack” overview
  • Machine language details needed for “A Attack”
  • Finally finishing the 4-stage build process: the Linker!

• Final precept:
  • MiniAssembler and ”A Attack” details
Final Exam Info

What: Final Exam!

When: ~3 weeks from today
Thursday, Dec 21
7:30pm – 10:30 pm

Where: McDonnell A01/A02


Why: Cumulative assessment. You've learned a lot, so show us!