Buffer Overrun Vulnerabilities and Assignment 6 (The ‘B’ Attack)
A 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?
Aarti Gupta
Thank you, Aarti Gupta.
The answer to life, the universe, and everything is 42
A Reason Why People With Long Names Can’t Have Nice Things

#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?
Christopher Moretti
Thank you, Christopher Mor
tti.
The answer to life, the universe, and everything is 6911092

#15334 (depending on the area code, this might be an interesting phone number, but probably not one you should call for the answer to life, the universe, and everything)
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!

```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;
}
```
`#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;
}

Christopher's (not \0 terminated) in name[0]–name[11]
More in 3 padding bytes before c

Each letter from getchar overwrites c (it is also
overwritten once by name[i++] = c, when i is 15 and c is
‘e’) until c becomes ‘\n’ and the loop ends.
First t overwrites 42 with 0x74 (‘t’) – little endian!
Second t makes magic 29812 (2 high-order bytes still 0)
Final i makes magic 6911092 (1 high-order byte still 0)
Buffer overrun can overwrite return address of a previous stack frame!

```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;
}
```
Buffer overrun can overwrite return address of a previous stack frame!

- Value can be an invalid address, leading to a segfault, or ...

```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;
}
```
Buffer overrun can overwrite return address of a previous stack frame!

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

```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;
}
```
It Gets Much, Much Worse…

Buffer overrun can overwrite return address of a previous stack frame!

- 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 execute!

```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;
}
```
Attacking a Web Server

URLs
Input in web forms
Crypto keys for SSL
etc.

Client PC
Web Server

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

this is a really long search term that overflows a buffer
Attacking a Web Browser

- HTML keywords
- Images
- Image names
- URLs
- etc.

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

Client PC

Web Server @ badguy.com

Earn $$$ Thousands working at home!
Attacking Everything in Sight

Client PC

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

The Internet
@ badguy.com

E-mail client
PDF viewer
Operating-system kernel
TCP/IP stack

Any application that ever sees input directly from the outside
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

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
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?
Aarti
D is your grade.
Thank you, Aarti.
$ ./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?
Aarti\0(#@$%^&(*^!@%*!!(&#$
B is your grade.
Thank you, Aarti.

Smash the stack!
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?

- SP
- readString’s stackframe
- getName’s stackframe
- main’s stackframe

???
buf
buf
...
buf
???

???
...
???

???
...
???
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?
Aarti
B is your grade.
Thank you, Aarti.
checkappel:
    if (strcmp(name, "Andrew Appel") != 0)
        goto afterb
        grade = 'B'
        afterb:
            print ...

...
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 (which in C is spelled unsigned long).
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 communicates, delivers, transmits, or causes to be communicated, delivered, or transmitted, or attempts to communicate,
Summary

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

- **This week’s 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 next week:**
  - MiniAssembler and ”A Attack” details