Computer Security

Prof. David August

COS 217

Interacting with the world

Keypress goes to OS kernel
OS looks up which window has “keyboard focus,” routes to appropriate user process’s stdin
User process does fprintf (asks OS to write to disk)
OS writes to disk

TCP packet goes to OS kernel
OS looks up which process is listening on that port, sends data to stdin
User process does fprintf (asks OS to write to disk)
OS writes to disk

Protection mechanisms

Keypress goes to OS kernel
OS looks up which window has “keyboard focus,” routes to appropriate user process’s stdin
User process does fprintf (asks OS to write to disk)
OS writes to disk

• Not to user process directly!
• Not to unauthorized user process!
• User process can’t access disk directly!
• OS writes only to files that user process has privileges to open!

TCP packet goes to OS kernel
OS looks up which process is listening on that port, sends data to stdin
User process does fprintf (asks OS to write to disk)
OS writes to disk
What prevents user process from directly accessing keyboard & disk?

- Input/output instructions are privileged instructions, attempting to execute them in unprivileged mode will result in trap to operating system
- Input/output device registers may be memory-mapped; virtual-memory system doesn’t map those pages into user space
- Virtual-memory system prevents user process from modifying OS memory (can’t fool OS into performing unauthorized services)
- Virtual-memory prevents user processes from modifying each others’ memory (can’t fool other process into writing bad data to its files on disk)

How attackers defeat protection

- Make the protection mechanism fail
  - (exploit bugs in protection software)
- Operate politely through the protection mechanism, manipulate semantics of application to obtain services
  - (exploit bad design of application)

A nice little program

```c
#include <stdio.h>
int main(int argc, char **argv) {
    char buffer[30]; int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        a[i] = c;
    }
    a[i] = '\0';
    printf("Thank you, %s\n",a);
    return 0;
}
```
Why did this program crash?

```
#include <stdio.h>
int main(int argc, char **argv) {
    char buffer[30];  int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        a[i] = c;
    }
a[i]='\0';
    printf("Thank you, %s.\n",a);
    return 0;
}
```

Stack frame layout

```
#include <stdio.h>
int main(int argc, char **argv) {
    char buffer[30];  int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        a[i] = c;
    }
a[i]='\0';
    printf("Thank you, %s.\n",a);
    return 0;
}
```

Buffer overrun

```
#include <stdio.h>
int main(int argc, char **argv) {
    char buffer[30];  int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        a[i] = c;
    }
a[i]='\0';
    printf("Thank you, %s.\n",a);
    return 0;
}
```
Innocuous? buffer overrun

```c
#include <stdio.h>
int main(int argc, char **argv) {
    char buffer[30];  int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        a[i] = c;
    }
    a[i]='\0';
    printf("Thank you, %s.\n",a);
    return 0;
}
```

Maliciously clever?

```c
#include <stdio.h>
int main(int argc, char **argv) {
    char buffer[30];  int i;
    printf("What is your name?\n");
    for (i=0; ; i++) {
        int c = getchar();
        if (c=='\n' || c ==EOF) break;
        a[i] = c;
    }
    a[i]='\0';
    printf("Thank you, %s.\n",a);
    return 0;
}
```

Buffer-overrun vulnerabilities

- Keypress goes to OS kernel
- OS looks up which window has "keyboard focus," routes to appropriate user process’s stdin
- User process does fprintf (asks OS to write to disk)
- OS writes to disk
Attacking a web server

- URLs
- Input in web forms
- Crypto keys for SSL
- etc.

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

Attacking a web browser

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

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

Attacking everything in sight

- E-mail client
- PDF viewer
- Operating-system kernel
- TCP/IP stack
- *Any* application that ever sees input directly from the outside
Your programming assignment

% a.out
What is your name?

John Smith
Thank you, John Smith.
I recommend that you get a grade of D on this assignment

char grade = 'D';
int main(void) {
    printf("What is your name?\n");
    readString(Name);
    if (strcmp(Name,"Andrew Appel")==0)
        grade='B';
    printf("Thank you, %s.\n"
          I recommend that you get a grade of %c 
          on this assignment.\n", Name, grade);
    exit(0);
}

OK, that's a B...

% a.out
What is your name?

John Smith
Thank you, John Smith.
I recommend... a grade of B...

char grade = 'D';
int main(void) {
    printf("What is your name?\n");
    readString(Name);
    if (strcmp(Name,"Andrew Appel")==0)
        grade='B';
    printf("Thank you, %s.\n"
          I recommend... grade of %c 
          on this assignment.\n", Name, grade);
    exit(0);
}

How about an A?

% a.out
What is your name?

John Smith
Thank you, John Smith.
I recommend... a grade of A...

char grade = 'D';
int main(void) {
    printf("What is your name?\n");
    readString(Name);
    if (strcmp(Name,"Andrew Appel")==0)
        grade='B';
    printf("Thank you, %s.\n"
          I recommend... grade of %c 
          on this assignment.\n", Name, grade);
    exit(0);
}
A simpler solution

```c
char grade = 'D';
int main(void) {
    printf("What is your name?\n");
    readString(Name);
    if (strcmp(Name, "Andrew Appel") == 0)
        grade = 'B';
    printf("Thank you, %s. \nI recommend ... grade of %c ...\nment.\n", Name, grade);
    exit(0);
}
```

The file getA

```c
getA:

John Smith .movl 'A', grade; jmp wherever 0000?Ak7@*%}
```

What value to use for new return address?

- Computers are deterministic
- Operating system initializes stack pointer to predictable value
- Stack grows deterministic amount from process entry to call of `readString`

```c
getA:

John Smith .movl 'A', grade; jmp wherever 00007Ak7@*%}
```
Use gdb to find out

```
(gdb) break read_string
(gdb) run
Starting program: a.out (no debugging symbols found)...
```

```
What is your name?
Breakpoint 1, 0x0804843d in read_string ()
```

```
buf     Old EBP  Old EIP
0030a898 bfffbb64 bfffbad8
bfffbad8 080484c3 08049770
00000001 00000007 0030a898
```

```
0xbfffbac0:     0x08049770      0x00000001
0x00000007      0x0030a898
0xbfffbad0:     0xbfffbb64      0x00000001
```

Defenses against this attack

- **Best:** program in languages that make array-out-of-bounds impossible (Java, C#, ML, ...)
- **Good:** use discipline in C programming always to check bounds of array subscripts
- **Better than nothing:** Operating system randomizes initial stack pointer
  - **How to attack it:**
    ```
    John Smith:......nop;nop;nop;...;nop;do_bad_things;exit(0)
    ```
    Can jump anywhere in here, so don’t have to know exact value of stack pointer

For this assignment, you don’t need such a fancy attack.

The hello.c program copies the buffer to the global bss data space (into the Name array) so you can just jump there, don’t have to know the stack height.
Defenses against this attack

- **Best**: program in languages that make array-out-of-bounds impossible (Java, C#, ML, ...)
- **Good**: use discipline in C programming always to check bounds of array subscripts
- **Better than nothing**: Operating system randomizes initial stack pointer
- **Better than nothing**: Prohibit execution of machine code from the stack and data segments
  - **Problem 1**: backward compatibility
  - **Problem 2**: need VM hardware with “exec/noexec” bit on a page by page basis; x86/Pentium family lacks this
- **Amazing hack solution**: use obsolete “segment registers” left over from 80286.

Segment register defense

- In normal (modern) usage, all segment registers point to entire range of addressable memory, 0 to 0xffffffff

  - Amazing hack is to have code segment point just to Text area
  - Problem: what if program wishes to create executable code on the fly?
  - Solution: undo protection

At your service...

- For your convenience in this programming assignment, we have turned off the segment-register defense

```c
char grade = 'D';
int main(void) {
    mprotect((unsigned)Name & 0xffffffff,1,
             PROT_READ | PROT_WRITE | PROT_EXEC);
    printf("What is your name?\n");
    readString(Name);
    if (strcmp(Name,"Andrew Appel")==0)
        grade='B';
    printf("Thank you, %s.\n\nI recommend ... grade of %c \n...nment.\n", Name, grade);
    exit(0);
}
```
How to get started

To succeed on this programming assignment,

- Use `gdb` to map out where things are
  - Stack frame of "readString"
  - Stack frame of "main" underneath it
  - Global data area containing "grade" and "Name"
  - Machine code for "main"
  
  Take notes of all these things, by address.

- Write a little assembly-language program
  - Set the "grade" variable to 'A'; jump to wherever
  - Assemble it, maybe even link it into a copy of hello.c, and examine what it looks like using gdb

- Prepare your attack data
  - I found it helpful to write a C program to print out the data string
  - Useful functions: printf, putchar, putw

Start early

- Use `gdb` to map out where things are
  - Stack frame of "readString"
  - Stack frame of "main" underneath it
  - Global data area containing "grade" and "Name"
  - Machine code for "main"
  
  Take notes of all these things, by address.

If possible, get this part done by the time your Weds/Thurs precept meets this week. Feel free to work jointly with another student on this part. Bring your notes with you to precept.