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
• Not to user process directly!
OS looks up which window has “keyboard focus,”
• Not to unauthorized user process!
User process does fprintf (asks OS to write to disk)
• User process can’t access disk directly!
OS writes to disk
• 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

% a.out
What is your name?
John Smith
Thank you, John Smith.
%

#include <stdio.h>
int main(int argc, char **argv) {
    char buffer[30];
    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?

% a.out
What is your name?
adsli57asdkhj5jkllds;ahj5;klsadu5klysdkl5aujksd5ukals;5uj;akukla
Segmentation fault
%

#include <stdio.h>
int main(int argc, char **argv) {
    char buffer[30];
    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
Buffer overrun

% a.out
What is your name?
abcdefgijklmnopqrstuvwxyz
Segmentation fault
%

#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

% a.out
What is your name?
abcdefgijklm????!!
%

#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

% a.out
What is your name?
abcdefgijklm????!!
%

#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;
}

Cleverly malicious? Maliciously clever?
Buffer overrun

% a.out
What is your name?
abcdefgijklm????!!
execuable-machine-code...
How may I serve you, master?
%

#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

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
Attacking a web server

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

Attacking a web browser

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

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

```c
% 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...

```
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 \n      ...nment.\n", Name, grade);
   exit(0);
}
```

How about an 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 \n      ...nment.\n", Name, grade);
   exit(0);
}
```

A simpler solution

```
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 \n      ...nment.\n", Name, grade);
   exit(0);
}
```

The file getA

```
getA:
John Smith, movl 'A',grade; jmp wherever
```

Size of buffer

Unchanged "old EBP" (return address)

New machine code
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`

getA:
John Smith .movl 'A', grade; jmp wherever

Use gdb to find out

- Old EBP
- Old EIP
- New machine code
- Size of buffer

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

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

```
char grade = 'D';
int main(void) {
    mprotect(((unsigned)Name) & 0xfffff000,1, 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.