CIS-5373
Systems Security
Class 1

Bogdan Carbunar
Outline

- Administrative Issues
- Textbooks
- Security Overview
Administrative Issues

- **Staff**
  - Bogdan Carbunar, associate professor

- **Communications**
  - E-mail: carbunar@cs.fiu.edu or carbunar@gmail.com

- **Office Hours**
  - Mondays, ECS 383, 4pm – 5pm
  - Prior appointment recommended
Class Grading (subject to changes)

- 1 final worth: 35%
  - Date of exam: TBD, but May 2020
- Paper presentation: 35%
- Homework: 30%
- Extra credit: 5-10%
  - Exceptional class participation
  - Additional activities (e.g., programming project)
Class Grading: Details (cont’d)

- Student paper presentations: 35%
  - Papers will be posted on class web page
  - Let me know in time (FIFO assignment rule)
Outline

- Administrative Issues
- Textbooks
- Security Overview
Textbooks

- *Security In Computing – 4th edition*
  Pfleeger and Pfleeger
- *Cryptography and Network Security*
  William Stallings
- *Applied Cryptography – 2nd edition*
  Bruce Schneier
  Available online

- Papers assigned for reading
  - See class webpage
Textbooks (cont’d)

- You don’t need to buy the books!
- http://www.wikipedia.org/
Outline

- Administrative Issues
- Textbooks
- Security Overview
Some Topics (Subject to Change)

- Vulnerabilities
- Malware
- Access Control
- Authentication & Key exchange
- Network Security
Outline

- Administrative Issues
- Class Overview
- Security Overview
Information Security

- Protecting information and information systems from unauthorized access [Source: wikipedia]
Computer Security

- Branch of information security applied to computers
- Objective: protection of information and property
  - Theft, corruption, or natural disaster
  - Allow the information and property to remain accessible and productive to its intended users

[Source: wikipedia]
Network Security

- **Provisions and policies** adopted by a network administrator to prevent and monitor
  - Unauthorized access
  - Misuse
  - Modification
  - Denial of access of network and resources

[Source: wikipedia]
System Security

- Goals: Protect
  - Confidentiality
  - Integrity
  - Availability
Confidentiality

- Information about system or its users cannot be learned by an attacker

- Data Confidentiality:
  - Private or confidential information is not revealed to unauthorized individuals
Integrity

- The system continues to operate properly, only reaching states that would occur if there were no attacker

- Data Integrity
  - Information and programs are changed only in specified and authorized manner

- System Integrity
  - System performs intended function and nothing else
Availability

- Actions by an attacker do not prevent users from having access to use of the system
  - Enable access to data and resources
  - Timely response
  - *Fair* resource allocation
More Required Concepts

- **Authenticity**
  - Being able to be verified and trusted
  - Confidence in the validity of a message (originator)

- **Accountability**
  - Actions of an entity can be traced to it
  - Tracing a security breach to a responsible party
Security is about

- Honest user (e.g., Alice, Bob, ...)
- Dishonest Attacker
- How the Attacker
  - Disrupts honest user’s access to the system (Integrity, Availability)
  - Learns information intended for Alice only (Confidentiality)
Examples

- **Confidentiality**
  - Student grades
  - Available only to student, parents, employer

- **Integrity**
  - Patient information e.g., allergies
  - Can lead to loss of human life

- **Availability**
  - Authentication service
  - Unavailability can lead to financial loss
Program Security and Vulnerabilities

Class 2
What is Security?

- **System correctness**
  - If user supplies expected input, system generates desired output
  - Good input $\Rightarrow$ Good output
  - More features: better

- **Security**
  - If attacker supplies unexpected input, system does not fail in certain ways
  - Bad input $\Rightarrow$ Bad output
  - More features: can be worse
Why Security Vulnerabilities?

Some contributing factors

- Few courses in computer security 😊
- Programming text books do not emphasize security
- Few security audits
- *C is an unsafe language*
- Programmers have many other things to worry about
- Consumers do not care about security
- Security is expensive and takes time
In this lecture

- Buffer Overflow
- SQL Injection Attack
- Incomplete Mediation
- Time-of-Check to Time-of-Use Errors
- Malicious Code
Famous Buffer Overflow Attacks

- **Morris worm (1988):** overflow in fingerd
  - *6,000 machines infected (10% of existing Internet)*

- **CodeRed (2001):** overflow in MS-IIS web server
  - *Internet Information Services (IIS)*
  - *Web server application*
  - *The most used web server after Apache HTTP Server*
  - *300,000 machines infected in 14 hours*

- **SQL Slammer (2003):** overflow in MS-SQL server
  - *75,000 machines infected in 10 minutes (!!)*
Famous Buffer Overflow Attacks

- **Sasser (2004):** overflow in Windows LSASS
  - **Local Security Authority Subsystem Service**
    - Process in Windows OS
    - Responsible for enforcing the security policy on the system.
    - Verifies users logging on to a Windows computer or server, handles password changes, and creates access tokens
  - *Around 500,000 machines infected*

- **Conficker (2008-09):** overflow in Windows Server
  - *Around 10 million machines infected (estimates vary)*
Memory Exploits

- **Buffer** is a data storage area inside computer memory (stack or heap)
  - Intended to hold pre-defined amount of data
- If executable code is supplied as “data”, victim’s machine may be fooled into executing it
- Code will give attacker control over machine
Stack Buffers

- Suppose Web server contains this function

  ```c
  void func(char *str) {
    char buf[126];
    strcpy(buf, str);
  }
  ```

- When this function is invoked, a new frame with local variables is pushed onto the stack.
Stack Buffers (cont’d)

- When `func` returns
  - The local variables are popped from the stack
  - The old value of the stack frame pointer (sfp) is recovered
  - The return address is retrieved
  - The stack frame is popped
  - Execution continues from return address (calling function)
What If Buffer Is Overstuffed

- Memory pointed to by str is copied onto stack...
  
  ```c
  void func(char *str) {
      char buf[126];
      strcpy(buf,str);
  }
  ```

- If a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations.

  ![Diagram showing stack overflow](image)

  Stack grows this way

  - **buf**: Memory pointed to by str
  - **overflow**: Area where stack overflows
  - **str**: Pointer to string

  This will be interpreted as return address!

  ```text
  strcpy does NOT check whether the string at *str contains fewer than 126 characters
  ```
Attack 1: Smashing the Stack

- Suppose buffer contains attacker-created string
  - For example, *str contains a string received from the network as input to some network service daemon

- When function exits, code in the buffer will be executed, giving attacker a shell
  - Root shell if the victim program is setuid root
Attack 2: Variable Overflow

- Somewhere in the code `authenticated` is set only if login procedure is successful
  - Other parts of the code test `authenticated` to provide special access

```c
char buf[80];
int authenticated = 0;
void vulnerable() {
    gets(buf);
}
```
- Attacker passes 81 bytes as input to `buf`
Attack 3: Pointer Variables

- `fnptr` is invoked somewhere else in the program
  - This is only the definition

```c
void func(char *s){
    char buf[80];
    int (*fnptr)();
    gets(buf);
}
```

- Local variables
- Pointer to previous frame
- Execute code at this address after `func()` finishes
- Arguments
Attack 3: Pointer Variables (cont’d)

- Send malicious code in `s`
- Overflow `fnptr`
  - Pass more than 80 bytes in `gets`
  - `fnptr` now points to malicious code
- When `fnptr` is executed, malicious code is executed!

```c
void func(char *s){
    char buf[80];
    int (*fnptr)();
    gets(buf);
}
```
Attack 4: Frame Pointer

- One way to address overflows: zero out emptied stack locations

- Not enough!
  - Send malicious code in `s`
  - Change the caller’s saved frame pointer.
  - Pass more than 80 bytes in `gets`
  - `sfp` now points to start of malicious code
  - When `func` returns, code is still on stack!

```c
void func(char *s){
    char buf[80];
    gets(buf);
}
```
Buffer Overflow Difficulties

- Executable attack code is stored on stack, inside the buffer containing attacker’s string
  - Stack memory is supposed to contain only data, but...
- For the basic attack, overflow portion of the buffer must contain *correct address of attack code* in the RET position
  - The value in the RET position must point to the beginning of attack assembly code in the buffer
  - Otherwise application will give segmentation violation
  - Attacker must correctly guess in which stack position his buffer will be when the function is called
Problem: No Range Checking

- **strcpy does not check input size**
  - `strcpy(buf, str)` simply copies memory contents into `buf` starting from `*str` until “\0” is encountered, ignoring the size of area allocated to `buf`

- **Many C library functions are unsafe**
  - `strcpy(char *dest, const char *src)`
  - `strcat(char *dest, const char *src)`
  - `gets(char *s)`
  - `scanf(const char *format, ...)`
  - `printf(const char *format, ...)`
Does Range Checking Help?

- **strncpy**(char *dest, const char *src, size_t n)
  - If strncpy is used instead of strcpy, no more than n characters will be copied from *src to *dest
  - Programmer has to supply the right value of n

- Potential overflow in htpasswd.c (Apache 1.3):

  ```
  ... strcpy(record, user);
  strcat(record, "":");
  strcat(record, cpw);
  ...
  ```

- Published “fix” (do you see the problem?):

  ```
  ... strncpy(record, user, MAX_STRING_LEN-1);
  strcat(record,"":");
  strcat(record,cpw, MAX_STRING_LEN-1);
  ...
  ```

Copies username ("user") into buffer ("record"), then appends ":" and hashed password ("cpw")
Published “fix” for Apache htpasswdd overflow:

```c
... strncpy(record, user, MAX_STRING_LEN-1);
    strcat(record, ":");
    strcat(record, cpw, MAX_STRING_LEN-1); ... 
```

MAX_STRING_LEN bytes allocated for record buffer

- **contents of *user**
- **contents of *cpw**

- Put up to MAX_STRING_LEN-1 characters into buffer
- Put ":"
- **Again** put up to MAX_STRING_LEN-1 characters into buffer
Attack 5: Integer Overflow

static int getpeername1(p, uap, compat) {
    // In FreeBSD kernel, retrieves address of peer to which a socket is connected
    ...
    struct sockaddr *sa;
    ...
    len = MIN(len, sa->sa_len);
    ...
    copyout(sa, (caddr_t)uap->asa, (u_int)len);
    ...
}  Copies “len” bytes from kernel memory to user space

Checks that “len” is not too big
Negative “len” will always pass this check...

... interpreted as a huge unsigned integer here
... will copy up to 4G of kernel memory