CS 493
Secure Software Systems

Midterm review

Dr. Williams
Central Connecticut State University

Agenda

- Guidelines
- Content
- Sample problems/questions

Test guidelines

- Midterm Wednesday(10/18)
- Monday we will go through more examples in class and review homework
- Allowed one letter size piece of paper of notes (single side) – it **<u>must</u>** be handwritten
- Partial credit – make your best effort!!

Exam Content

- Note some aspects the textbook covers more depth than covered explicitly in lecture and some aspects covered more in lecture than in textbook
- You are expected to know **<u>both</u>** and could have questions on both, but focus will be on that covered in class

Textbook content

Textbook chapters 1-9

1. Intro
2. Current and emerging threats
3. The network environment
4. The operating system
5. The database environment
6. Programming languages
7. Security requirements planning
8. Vulnerability mapping
9. Development and implementation

Intro objectives

- Identify threat agents
- Identify mitigation techniques
- Identify how security affects the software design process.
- Understand common techniques for improving system security.
- Understand the context

Key terms

- A **vulnerability** is simply a design flaw or an implementation bug that allows a potential attack on the software in some way.
- A **threat** is a possible exploit of a vulnerability
- An **attack** is the actual use of such an exploit.
- A **countermeasure** is a means to eliminate the possibility of an attack or at least to mitigate the amount of damage caused if it occurs
- A **threat agent** is anything or anyone who could potentially harm your software.

Common Vulnerabilities

- Lack of input validation
- Insecure configuration management
- Lack of bounds checking on arrays and buffers
- Unintentional disclosure

The Usual Suspects

- A **virus** is a segment of code that attaches to a host file.
- A **worm** is a stand-alone executable that operates like a virus, with the exception that it does not need a host file on which to reside.
- A **Trojan** offers a service that a user would find desirable, such as photo editing or even, ironically, virus protection. It may or may not deliver that service, but as it does so, it is also performing malicious activity on the host machine.

The Usual Suspects

- A **vulnerability scanner** applies to any software that looks for weaknesses on a system.
- **Backdoor** is a method of circumventing normal authentication procedures and allowing unwanted access into a computer system.
- **Rootkits** are pieces of software that actually subvert the legitimate control of a software system by its operators, operating at the highest level of permission on the system.

The Usual Suspects

- **The Human Element**
  - Social engineering is a dangerous tactic to even the most hardened security. Because the human element is often the weakest link in security, it only makes sense that attackers would choose that as their angle of attack.
  - Also just human error in general

The CIA Triad

- **Confidentiality** in an application means that the private and sensitive data handled by the application cannot be read by anyone who is not explicitly authorized to view it.
- **Integrity** means that the data processed by an application is not modified by any unauthorized channels or any unauthorized persons.
- **Availability** is defined as system’s ability to remain operational even in the face of failure or attack.

Cryptography

- **Cryptography** is a mathematical approach to transforming data such that, without the necessary piece of information, a key to unlock it, the information cannot be read.
- **Cleartext** or **plaintext** is information that is not encrypted is said to be in the clear.
- **Ciphertext** is information that has been encrypted.

Symmetric key cryptography

- **Symmetric cryptography** same key is needed for both the encryption and decryption process.
- Implication is both the sender and receiver must have the same key which had to be exchanged at some point
- Thus the safety of a message is only as protected as the method the key is distributed

Public Key Cryptography

- **Public key cryptography** states that two keys are needed for the process; one key is used to encrypt and one is used to decrypt.
- Result <u>secret key never needs to be distributed</u>, public key can be seen by anyone without compromising security
- Requires someway to know you can trust that public key actually is for that entity – typically certificates from certificate authority - public key infrastructure (PKI)
- Public key cryptography also makes signatures possible to allow non-repudiation

Integrity

- Integrity is used to prevent data modification, insertion or deletion by unauthorized parties.
- A **cryptographic hash** ensures that the message hasn’t changed since the hash was created, but does not guarantee the hash was produced by that user
  - To ensure hash created by a trusted source must either be encrypted by a shared symmetric key or signed using public key cryptography

Availability

- **Availability** is the more difficult attribute of security to ensure.
- **Redundancy** can be used to provide availability.
- **Off-site backup** may work in the midst of an attack, but it is costly to keep it running when there is no emergency situation requiring it.

Fundamental strategies for secure software

Prevention
Avoidance
Detection
Recovery

- **Prevention** - assertion that an attack absolutely cannot happen to or through your system.
- **Avoidance** - best attempt at making sure that attacks do not affect your system.
- **Detection** - Checkpoints to verify everything is still working correctly otherwise raise an alert
- **Recovery** - Restarting the application at the last safe state

Changing the Design

- The term **scope creep** comes up a lot in software engineering; this is the term for a software. system taking on more and more functionality as it is developed. Specifically, this is functionality that was not intended when the software system was originally conceived.
- If you keep going and keep adding, you are adding holes in your software and increasing its complexity beyond its scope.

Current And Emerging Threats

- Common organization security threats
- The mentality that allows system compromise
- Impedance mismatch in system development
- Risks associated with personnel
- Risks to the network environment
- Risks to the operating system environment
- Risks to the database environment
- **Recognize all involved must work together to achieve security**

The Human Factor

- Most dangerous types of attacks use a combination of social engineering tactics and technical tools.
- A **social engineering attack** is one in which the attacker uses easily available company information, which a company thinks is innocuous, to disguise him- or herself as someone who is authorized to receive protected information.
- I**nformation assurance training program (IATP)** – goal is to train employees not to fall for social engineering

Social engineering in action

- New employee
- Tech support call
- The human factor – build trust
- Security testing of ***information assurance***

The Network

- Nearly all current systems to be relevant are connected to the network internally at a minimum and the majority have some connections externally as well
  – Result lots of avenues for attacks
  – Attention needed to limit unnecessary avenues
- Web 2.0 technologies include social networking sites, such as Facebook and Twitter
  – Prime target for social engineering and leaking information
  – Also haven for malicious code and phishing attacks.

The Operating System Environment

- Weak Passwords
- Open network ports
- Old software versions
- Insecure and poorly configured programs
- Stale and unnecessary accounts
- Procrastination of updates

Data-centric threats

- When all is said and done, system security is all about protecting the data, from corporate secrets to your own personal address.

- A **DBMS** establishes relationships between tables of data.
- **Data** is simply defined as “raw facts.”
- A **database** is a collection of data that has an established relation.
- A database also contains a **data dictionary** known as the **metadata** or “data about data.”
- **Information** is defined as data that has been organized into a format that is useful and actionable.

The Network Environment Objectives

- Identify threats to network communication.
- Identify proper applications of cryptosystems to the protection of network traffic.
- Identify the risks associated with different layers of connectivity.
- Assess the needs of a message in transit.
- Plan the network communication structure

CIA – related to networking

- **Confidentiality –** communication is sent unencrypted
- **Integrity** – communication can be altered and not detected
  - **Authentication –** In addition to not being altered that you know it is from party you think you are communicating with
  - **Nonrepudiation** means the person or entity to whom you are speaking cannot deny speaking to you and is therefore held accountable for what was said.
- **Availability** – can needed messages get through

Cryptography types

- **Symmetric Encryption**
  - <u>Single key</u>, which has to be kept secret, is used to encrypt the plaintext and the same key was used to decrypt the ciphertext.
- **Asymmetric encryption**
  - <u>Two separate keys</u>
    - <u>Private key</u> – must be kept secret, used to decrypt and sign
    - <u>Public key</u> – does not need to be kept secret, used to encrypt and verify signature

The Quest for Perfect Secrecy

- Only one algorithm has been found so far that establishes perfect secrecy: the onetime pad.
- A **one-time pad** uses simple substitution with a key length equal to the length of the message so that each letter is substituted by a different alphabet without repetition.
- This cipher is immune to brute force because a brute force attack will yield every possible result of equal length to the message without any way to determine which one is the actual message.
- <u>Impractical because of key distribution problem</u>

Eve

- The simplest scenario is where Alice sends Bob a message (M) without any encryption. Eve can intercept this message, often without the detection of either party.
- Without any encryption, Eve can read the message and store the information for use at her leisure.

Eve Unleashed

- Eve can take encrypted messages offline and start working to break the encryption scheme
- Passwords/keys critical
  - The lowest order of attack **brute force** – try all possibilities
  - A **dictionary attack** is the use of common words to form possible keys or passwords; a better variant of this is to hybridize words and numbers.

Eve Unleashed

- **Security by obscurity** means you rely on an attacker not knowing how the internal mechanism of your security operates as a means of securing the system.
- <u>The strength of your security must be in held even if you assume that an attacker knows your security schema</u>

Trudy
Malicious Modifications and Insidious Insertions

- Eavesdropping is generally a passive activity and the participants are unaware of the extra presence.
- Also must consider active attacker
  - Attacks on integrity – cryptographic hash and its limits
  - How to ensure integrity
  - Difference between integrity and non-repudiation

Making the Connection

- A **protocol** is a set structure for a message that allows network hardware to determine what information is being sent and what to expect; a protocol can include a single pattern for all communication or multiple patterns for continued communication between parties.

OSI Model

- **Open Systems Interconnection (OSI) model.**
  1. **Physical layer**
  2. **Data link layer** - physical address of a device (MAC) address
  3. **Network layer** - Routing between machines
  4. **Transport layer** - End-to-end transfer of data
  5. **Session layer** - organizing connections between a network node and a remote entity or service.
  6. **Presentation layer** - translates between application and network formats; this layer is primarily concerned with the representation of data and any possible structure of the data
  7. **Application layer** - software is directly involved in directing network communications.

The OS Environment Goals

- Define computer operating system security.
- Describe common security flaws applicable to operating systems.
- Mitigate the security vulnerabilities in common operating systems.
- Apply disaster and recovery techniques to operating systems.

What Is Operating System Security?

Operating system

- Manager for hardware and software resources on a computer
- Controls resource usage and access and provides a means for the user to interact with the computing system
- Where your applications live and access the network if required.
- Connects all communication with the network

Operating System Threats

- **Boot sector virus-** an attack which occurs when the first sector is loaded into memory when the computer is started.
- **Macro virus-** a virus written as a macro to execute malicious code.
- **Polymorphic virus-** a virus type that constantly morphs or changes its footprint to fool virus-detecting software.
- **Worm-** Similar to a virus that has the ability to self replicate while working its way through the network.
- **Rootkits** are purposely designed to hide in your operating system by hiding fragments of the executable and deleting detectable fragments after it executes

Operating System Defense Tactics

Multilayered approach to your security strategy.

- Keeping all security patches up to date
- Finding services you don’t use, shutting them down, and making them unviable to an attacker.
- Avoid bypassing OS security measures, encrypted HD
- Use levels of control available to limit what an account is allowed to do
- Strong passwords
- Do not work off of the administrator or root account to conduct non-administrator tasks.
- Review your system logs and store them on a different server.
- Disable booting from external devices and set a password for the boot loader.

Auditing and Monitoring

- Monitoring and auditing changes that occur in an information system are crucial to effectively managing security in an organization’s infrastructure.
- Auditing requires that a written policy be established that determines and dictates how and which events will be archived.
- Auditing and monitoring help you to identify a baseline for resource usage.

Backup and Redundancy

There are three types of data backups that should be noted:

- **Full backup:** Make an entire copy of all of the data from the target drive.
- **Differential backup:** Make copies of all files that have changed since the last full backup.
- **Incremental backup:** Copy only changed files since the last full backup.

Full backup strategy

- Full backup of all files from the target drive
- Pro
  - Easy to recover quickly from any specific day
- Con
  - Typically very slow with large systems sometimes taking longer than 24 hours
  - Large amount of data storage needed
    - If take full backup 7 days a week need storage for 7x the size of the active system for just a week's worth of back ups!

Differential backup strategy

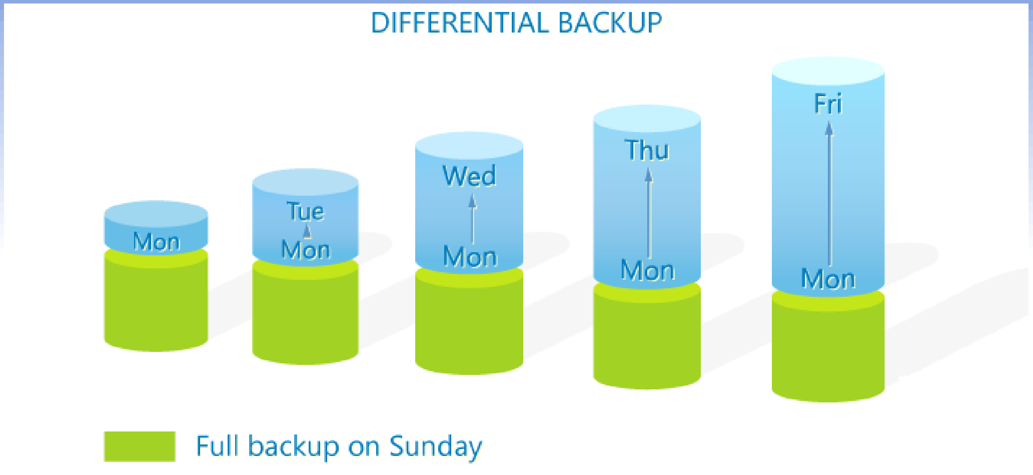


Image from codetwo.com

- Full back up taken periodically
- Each subsequent back up until next full back up stores all files that changed **<u>since the last full</u>** back up
- Pros
  - Much faster than full backup
  - Backup giving daily recovery per week much smaller than full backup approach
- Cons
  - If time between full backups is long difference for day can get large potentially as large as full back up
- Recovery applied by taking last full then applying difference file for day to restore to

Increment backup strategy

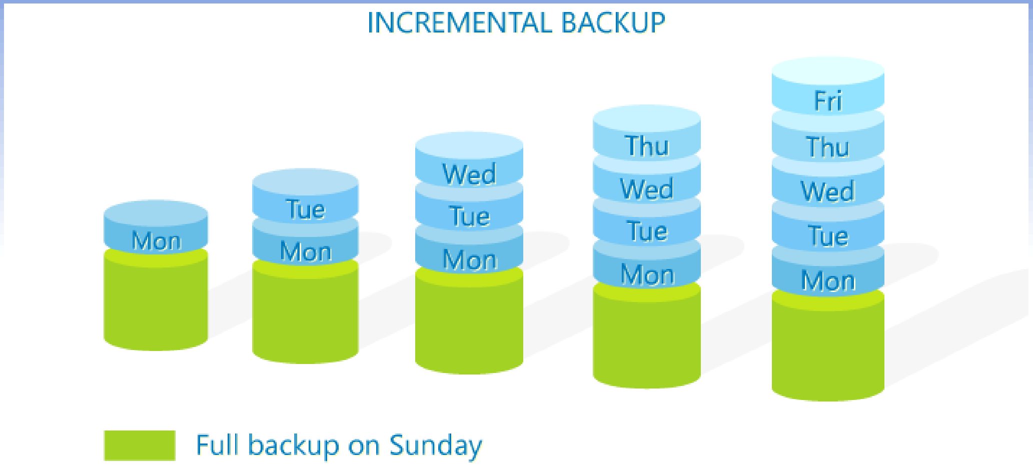


Image from codetwo.com

- Full back up taken periodically
- Each subsequent back up until next full back up stores files that changed **since the last incremental** back up
- Pros
  - Much faster than full backup, typically much faster than differential – Can allow smaller timeframes such as down to hourly backups
  - Backup giving daily recovery per week much smaller than full backup approach, typically much smaller than differential as well as only small amount each time
- Cons
  - Recovery applied by taking last full then applying differences to files in sum of back ups forward to restore point – so can be more lengthy recovery
  - In some cases if single increment is lost impossible to restore (incremental) back up after that point

Backup and Redundancy

Many operating systems have the ability to use a process for fault tolerance protection known as **Redundant Array of Independent Drives (RAID).**

- **RAID Level 0:** Does not provide fault tolerance, but simply improves read and write performance by using **disk striping,** which spreads blocks of data across the disk array.
- **RAID Level 1:** Uses disk mirroring to provide fault tolerance.
- **Disk mirroring** is the use of several drives connected to the same RAID disk controller, which sends the duplicate data to the other disks in the RAID to create a mirror of the data. This gives the administrator the ability to remove a faulty disk and continue operations without loss of data. Level 1 is slower on write operations.
- **RAID Level 5:** Uses multiple drives, but stores error-checking data, known as parity data, on all drives in the array instead of just one, providing an additional layer of protection. This level is one of the most popular for new system implementations.
- **Raid Level 10:** Uses both mirroring and striping to provide a higher degree of read and write performance than the other RAID levels, but it is more expensive because of the use of additional drives.

Other key ideas

- **Remote Access Services (RAS)** are deployed using servers and software to allow remote workers the same access used by employees working at the physical organization’s facility.
- **Virtualization** allows system administrators the opportunity to consolidate system resources using a logical view of system resources rather than a physical environment.

Database Fundamentals

Modern organizations rely heavily on the transformation of data to information in order to make critical business decisions.

- **Data** is simply defined as “raw facts.”
- **Information** is data that has been organized into some useful format to help the entity make critical business decisions.

Database Fundamentals

- The **database management system (DBMS)** is the software used to manage, create, and maintain the database.
- An **attribute** or **field** is the property used to describe a characteristic of the entity.
- The **primary key** is a unique identifying attribute or a combination of two attributes that identify the row in a relation.

Database Fundamentals

- A **foreign key** is a field in the database that serves as a primary key in a different table or entity in the same database.
- The **schema** identifies the logical mapping of the entire database.
- The **subschema** is the visualization of the database as seen by the database user.

Database Fundamentals

- **Data Definition Language (DDL),** which is used to develop the schema.
- **Data Manipulation Language (DML)** is used to manipulate the data in the database.
- Objectives of relational model:
  - Consistency
  - Eliminate redundancy
  - Data independence - application separate from the data representation

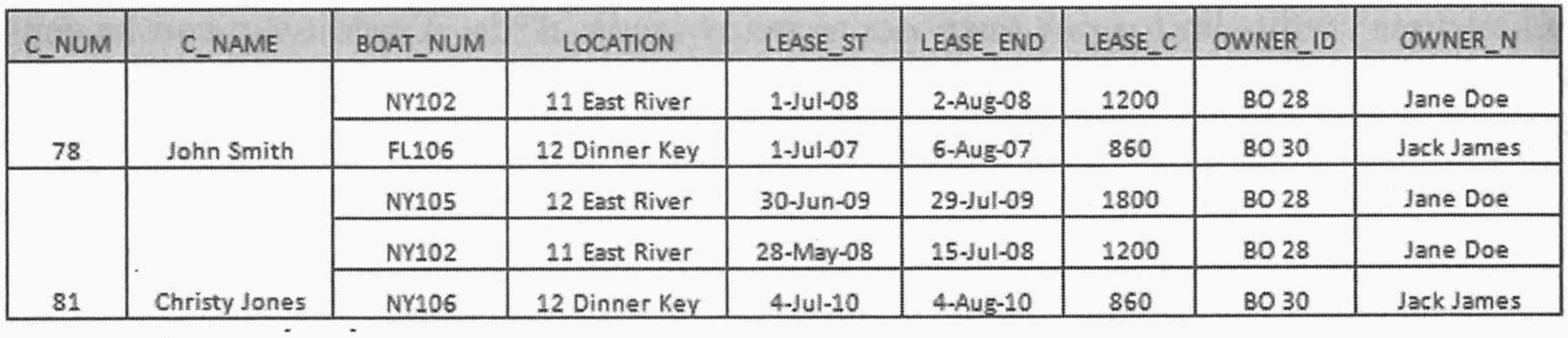
Database Design

- **Conceptual modeling** is the process used to construct the architectural components of the database. Data requirements are collected, detailed definitions are developed, and diagrams produced.
- During the **logical design,** the developer takes conceptual design and implements the database into a logical data model.
- The **entity-relationship model** is the conceptual representation of the data in an organization.

Database Normalization

- **Normalization** is defined as a methodology used to develop well-organized entities based on the organization's information needs.
- This method is used to eliminate redundancy, anomalies, and inconsistency in the database.

Unnormalized data



C_NUM	C_NAME	BOAT_NUM	LOCATION	LEASE_ST	LEASE_END	LEASE_C	OWNER_ID	OWNER_N
		NY102	11 East River	1-Jul-08	2-Aug-08	1200	BO 28	Jane Doe
78	John Smith	FL106	12 Dinner Key	1-Jul-07	6-Aug-07	860	BO 30	Jack James
		NY105	12 East River	30-Jun-09	29-Jul-09	1800	BO 28	Jane Doe
		NY102	11 East River	28-May-08	15-Jul-08	1200	BO 28	Jane Doe
81	Christy Jones	NY106	12 Dinner Key	4-Jul-10	4-Aug-10	860	BO 30	Jack James

Figure 5.3

Unnormalized data in the `BoatRental` table.

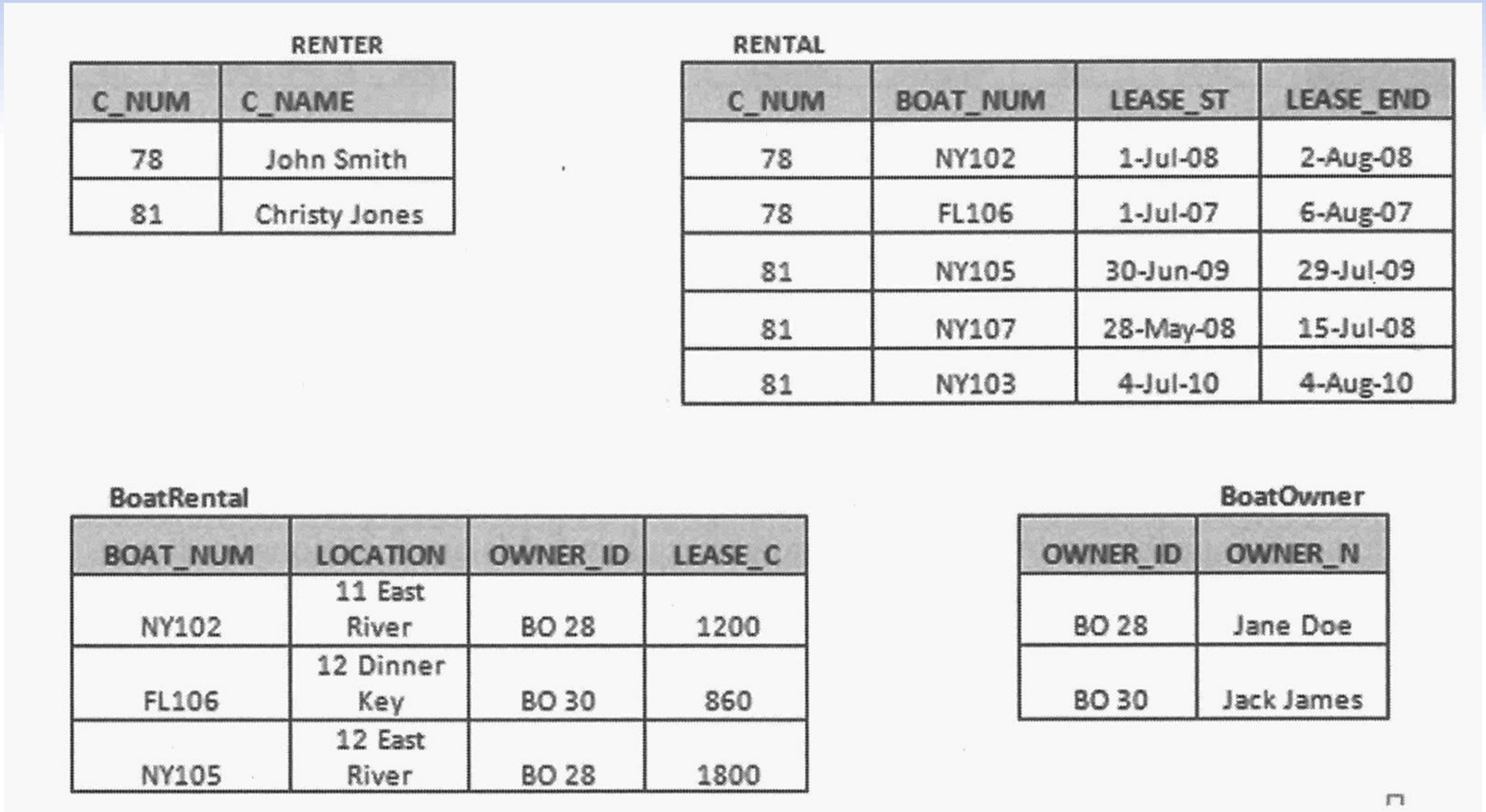
Database Normalization

Normalization:

- Eliminate repeating groups.
- Identify each entity with a primary key.
- Eliminate partial dependencies
- 3NF All transitive dependencies have been removed.

3NF

All transitive dependencies have been removed.



RENTER

C_NUM	C_NAME
78	John Smith
81	Christy Jones

RENTAL

C_NUM	BOAT_NUM	LEASE_ST	LEASE_END
78	NY102	1-Jul-08	2-Aug-08
78	FL106	1-Jul-07	6-Aug-07
81	NY105	30-Jun-09	29-Jul-09
81	NY107	28-May-08	15-Jul-08
81	NY103	4-Jul-10	4-Aug-10

BoatRental

BOAT_NUM	LOCATION	OWNER_ID	LEASE_C
NY102	11 East River	BO 28	1200
FL106	12 Dinner Key	BO 30	860
NY105	12 East River	BO 28	1800

BoatOwner

OWNER_ID	OWNER_N
BO 28	Jane Doe
BO 30	Jack James

Figure 5.7

(3NF) New normalized relations.

The Physical Design

- **Physical design**
  - Hardware needed
  - DBMS platform
  - Indexing and file organization
  - Transformation of the entity relationship diagrams into relations.
- A performance and tuning plan would be developed at this stage.

SQL

SQL (Structured Query Language)

- **nonprocedural language** powerful tasks using relatively simple commands
- Data Definition Language (DDL)
  - Creation, modification, deletion of tables and keys
- Data Manipulation Language (DML)
  - Creation, modification, deletion of records
- Data Control Language (DCL)
  - Manipulation of of access/authorization

Data manipulation

Select SQL cheat sheet

```
select * from PERSON

select id,firstname from PERSON

select * from PERSON where id=?

select * from PERSON where
  firstname like ?
```

? - "Bob%“

Join tables:

```
select P.firstname, A.city from
  PERSON P, ADDRESS A where P.ID =
  A.PERSONID
```

Insert statement cheatsheet

```
insert into PERSON values
  (?,?,?)
```

- Note very problematic as depends on order of database columns if DBA changes these it breaks all of your code or worse

```
insert into PERSON
  (id,firstname) values(?,?)
```

Update statement cheatsheet

```
update PERSON set firstname=?,
  lastname=? WHERE id=?
```

```
update PERSON set firstname=?
  WHERE lastname=? AND age=?
```

Note like delete, very dangerous if WHERE condition isn’t specific enough

Delete statement cheatsheet

```
delete from person where id=?

delete from person where
  firstname=? AND lastname=?
```

Very dangerous if WHERE condition isn’t specific enough

Web Applications and the Internet

- A web application is an application that end users access by using the Internet.
- One of the most common threats to a web application is an SQL injection attack.
- **SQL injection attack** is a serious threat to any application that translates raw user input into database communications.

Programming languages Goals

- Identify common attacks against programming languages.
- Identify mitigation techniques to prevent malicious input.
- Determine the highest risks to the use of an API or library.
- Identify the threats to the most common programming languages.
- Conduct an investigation into the programming languages used in your system.

Programming and Security

- **Programming Securely** To develop code in a secure manner so that the code itself is not a vulnerability that can be exploited by an attacker.
- **Programming Security** To develop code for security-specific functions such as encryption, digital signatures, firewalls, etc.

- In this lecture focus on programming securely:
  - Programming securely – language security and interaction security
  - programming security: security APIs and trust models.

Software Issues

Alice and Bob

- Find bugs and flaws by accident
- Hate bad software…
- …but must learn to live with it
- Must make bad software work

Trudy

- Actively looks for bugs and flaws
- Likes bad software…
- …and tries to make it misbehave
- Attacks systems via bad software

Language Barriers

- **Programming languages** are convenience structures that keep a programmer from needing to speak the native language of the machine
- Most vulnerabilities arise from differences in how assumptions in one level of language translate (or don't translate) to a different level/language

Language Barriers

- **Application programming interfaces (APIs)** have been created to allow a system to call existing functionality in another module or system through a specified interface.
- **Compiling** is the process of translating the high-level language into native machine Code.
- An **interpreted language** is one that has a lower layer of machine code that dynamically reads and interprets commands from a higher-level language.

Program Flaws

- An **error** is a programming mistake
  - To err is human
- An error may lead to incorrect state: **fault**
  - A fault is internal to the program
- A fault may lead to a **failure**, where a system departs from its expected behavior
  - A failure is externally observable

$$\begin{tikzcd}\text{error} \arrow[r] & \text{fault} \arrow[r] & \text{failure}\end{tikzcd}$$

Example

```
char array[10];
for(i = 0; i < 10; ++i)
      array[i] = `A`;
array[10] = `B`;
```

- This program has an **error**
- This error might cause a **fault**
  - Incorrect internal state
- If a fault occurs, it might lead to a **failure**
  - Program behaves incorrectly (external)
- We use the term **flaw** for all of the above

Secure Software

- In software engineering, try to ensure that a program does what is intended
- ***Secure*** software engineering **requires** that software **does what is intended…**
- **…and nothing more**
- Absolutely secure software is impossible
  - But, absolute security ***anywhere*** is impossible
- **How can we manage software risks?**

Buffer Bashing

- A **buffer overflow**, in its most general sense, is when more information is written to a location than the location can hold.
- Most buffer overflows in programming result from a lack of constraint on the amount of input that is allowed or the mishandling of pointer variables.

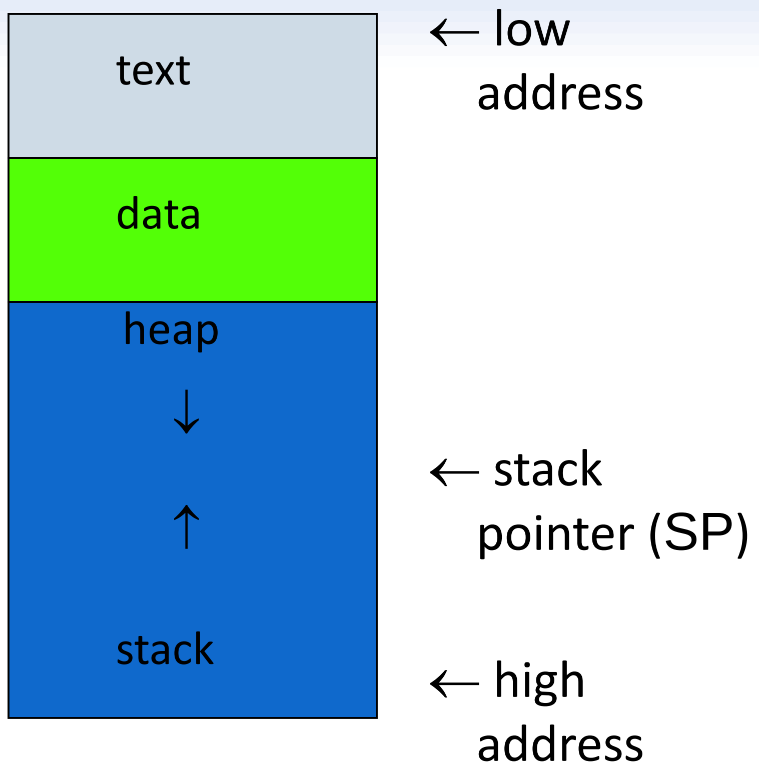
Buffer Overflow

```
int main(){
     int buffer[10];
     buffer[20] = 37;}
```

- **Q:** What happens when code is executed?
- **A:** Depending on what resides in memory at location “buffer[20]”
  - Might overwrite **user** data or code
  - Might overwrite **system** data or code
  - Or program could work just fine

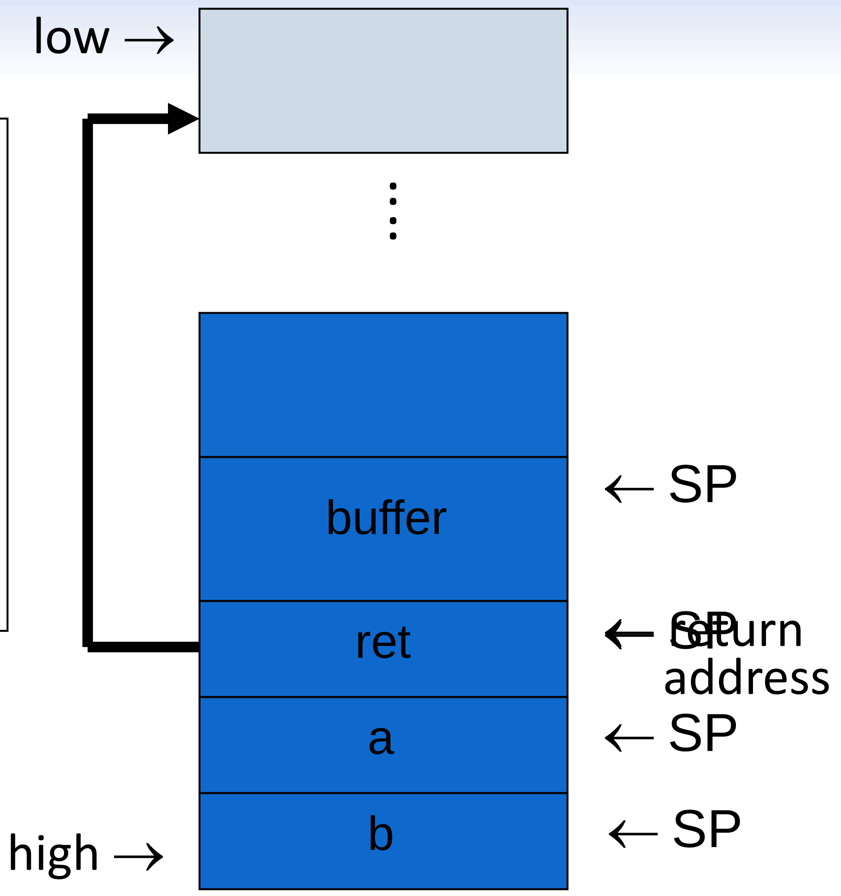
Memory Organization

- **Text** == code
- **Data** == static variables
- **Heap** == dynamic data
- **Stack** == “scratch paper”
  - Dynamic local variables
  - Parameters to functions
  - Return address



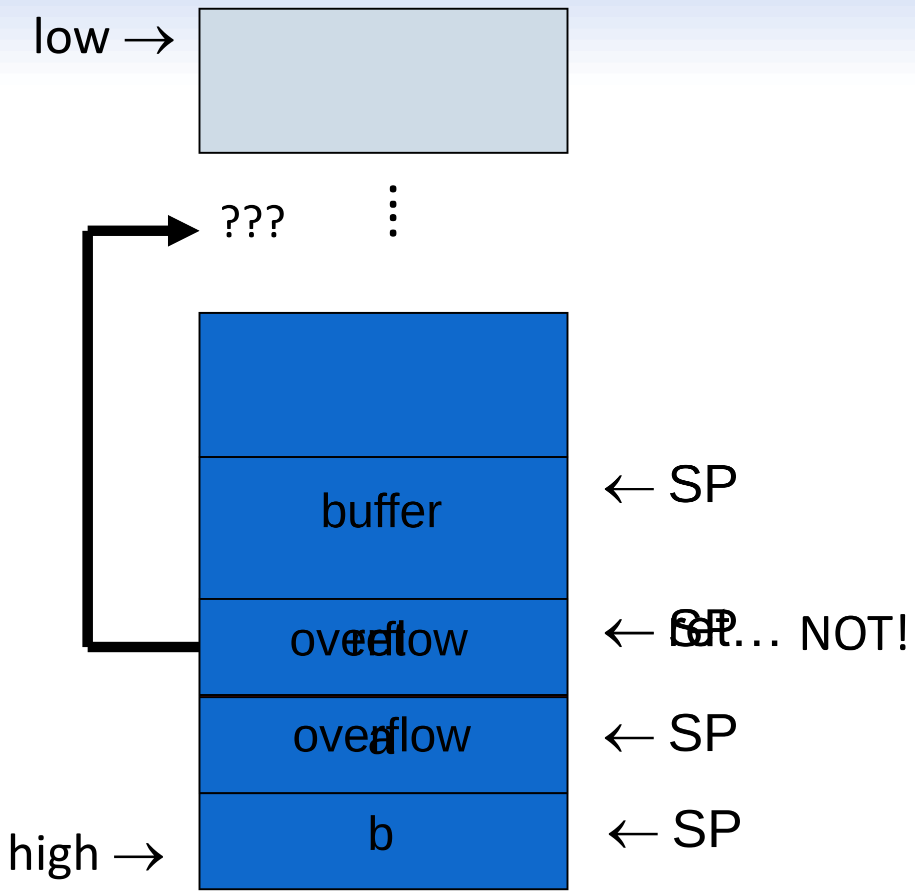
Simplified Stack Example

```
void func(int a, int b){
   char buffer[10];
}
void main(){
   func(1, 2);
}
```



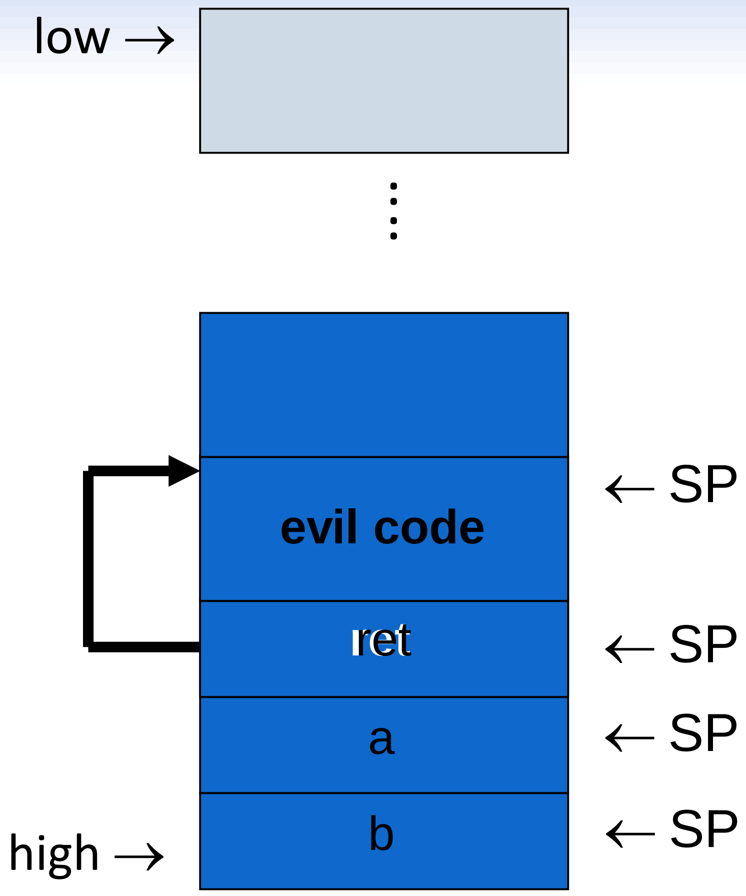
Smashing the Stack

- What happens if buffer overflows?
- Program "returns" to wrong location
- A crash is likely



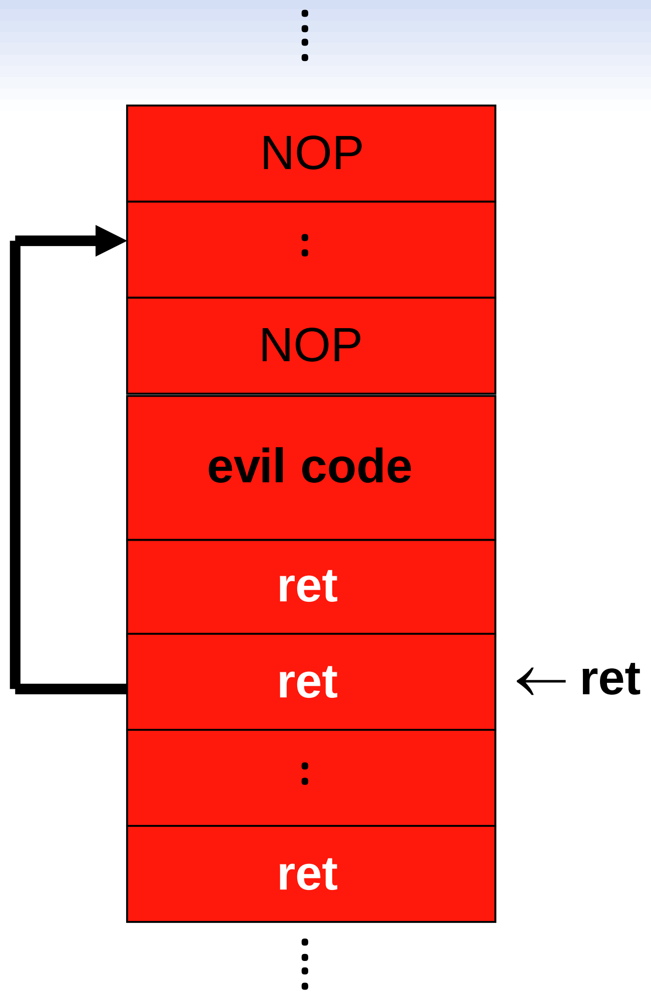
Smashing the Stack

- Trudy has a better idea…
- **Code injection**
- Trudy can run code of her choosing…
  - …on your machine



Smashing the Stack

- Trudy may not know…
  1) Address of evil code
  2) Location of **ret** on stack
- Solutions
  1) Precede evil code with NOP "landing pad"
  2) Insert **ret** many times



Stack Smashing Summary

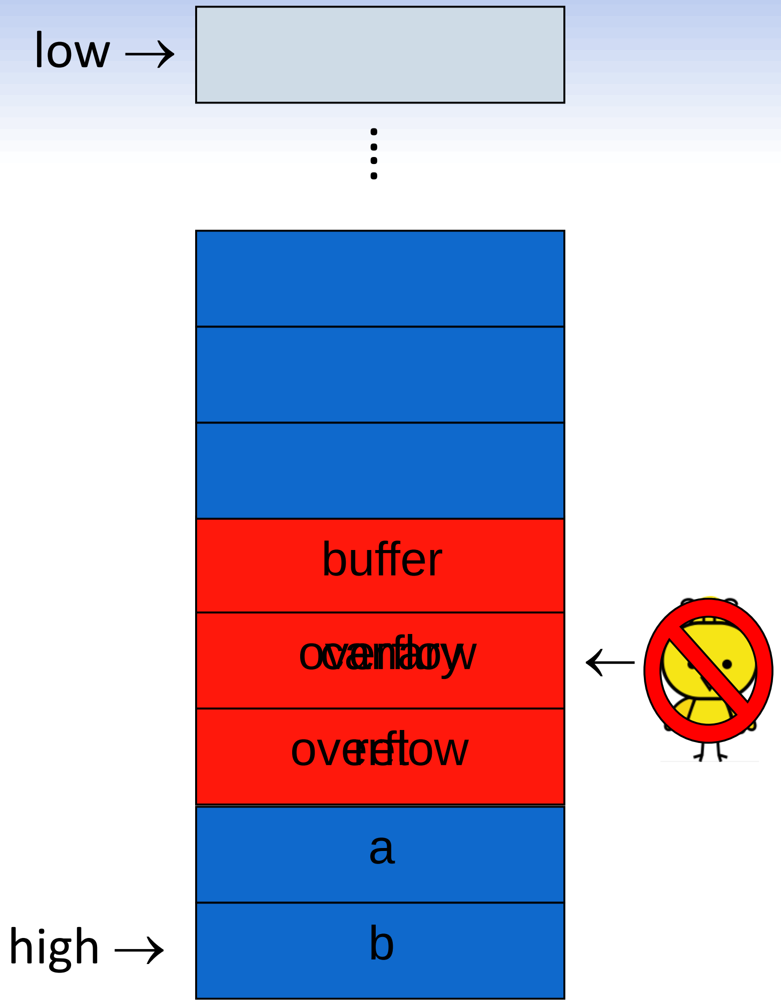
- A buffer overflow must exist in the code
- Not all buffer overflows are exploitable
  - Things must align properly
- If exploitable, attacker can **inject code**
- Trial and error is likely required
  - Fear not, lots of help is available online
  - Smashing the Stack for Fun and Profit, Aleph One
- Stack smashing is “attack of the decade”
  - Regardless of the current decade
  - Also heap overflow, integer overflow, …

Stack Smashing Defenses

- Employ **non-executable stack**
  - "No execute" **NX bit** (if available)
  - Seems like the logical thing to do, but some real code executes on the stack (Java, for example)
- Use a **canary**
- Address space layout randomization (**ASLR**)
- Use **safe languages** (Java, C#)
- Use **safer C functions**
  - For unsafe functions, safer versions exist
  - For example, strncpy instead of strcpy

Stack Smashing Defenses

- **Canary**
  - Run-time stack check
  - Push canary onto stack
  - Canary value:
    - Constant 0x000aff0d
    - Or may depends on **ret**



ASLR

- Address Space Layout Randomization
  - Randomize place where code loaded in memory
- Makes most buffer overflow attacks probabilistic
- Windows Vista uses 256 random layouts
  - So about 1/256 chance buffer overflow works?
- Similar thing in Mac OS X and other OSs
- Attacks against Microsoft's ASLR do exist
  - Possible to "de-randomize"

Buffer Overflow

- A major security threat yesterday, today, and tomorrow
- The good news?
- It <u>is</u> possible to reduced overflow attacks
  - Safe languages, NX bit, ASLR, education, etc.
- The bad news?
- Buffer overflows will exist for a long time
  - Legacy code, bad development practices, etc.

Buffer Bashing

There are defenses against buffer overflow, though none of them are perfect:

- **Array bounding**
- **Pointer handler indirection**
- **Data canaries**
- **Strict read and write limits**

Input Validation

- Consider: `strcpy(buffer, argv[1])`
- A buffer overflow occurs if

  `len(buffer) < len(argv[1])`
- Software must **validate** the input by checking the length of `argv[1]`
- Failure to do so is an example of a more general problem: **incomplete mediation**

Good Input

- **Input validation** is a common tool used in developing a system. This asserts that the information entered by a user is of proper format for the system to process.
- **Input validation** at the front end of the application is wonderful for getting legitimate users to line up correctly, but the attacker can circumvent the front end. If the same validation is not done on the back end of the system, your initial input validation is meaningless in terms of attack.

JIT Systems

There are several steadfast rules that should be employed with any system, but most of all with JIT systems:

- **Never execute your input.**
- **Keep a layer between your code and your input.**
- **Map your exceptions.**

Good Output

Output scrubbing is different than input scrubbing because you do not need to worry about format and injection unless you are passing user input through your system into another external component.

- **Encrypt information that is secret.**
- **Include only what is necessary for the external process.**
- **Do not assume trust for the external system.**

Inherent Inheritance and Overdoing Overloads

Two consideration that need to be made when dealing with secure coding:

1. **Inheritance** is the use of a base parent class and defining specific extensions of it.
2. An **overload** is a redefinition of an existing operation for a new class.

The Threatdown

The following list identifies some of the highest-reported vulnerabilities for these languages and what you can do to mitigate their effects.

C: has its main vulnerabilities in susceptibility to buffer overflow and data type mismatch.

C++: One of the main languages used in infrastructure along with C, C++ falls victim to buffer overflow as easily as C. No inherent bound checking occurs in either language. One of the other significant areas of vulnerability for C++ is the boundary conditions on floating point values.

The Threatdown

Java: One of the most popular and powerful languages, Java was designed to have the highest interoperability of any language in existence. The highest risk to Java is the ability to call out external executions and inject OS commands.

C#: is Microsoft's answer to Java. It exists on the .NET platform from Microsoft, which insulates it from most of the usual suspects when it comes to exploits. The most significant risk in using any .NET application is specifying an unsafe block of code.

The Threatdown

Visual Basic (VB): Although it has lost out for most thick client development, VB has found a new life in Active Server Pages (ASP). Remote code execution is the big item on the list for VB.

PERL: PERL is often called the "Duct Tape of the Internet" because of its unequaled power and ability to compile on demand within a server environment. It is vulnerable to command injection.

The Threatdown

- **PHP:** PHP is another common language used for web applications. In particular, sending malicious-form data could actually corrupt the internal data of a PHP application and run arbitrary code or simply cause a system crash.

Web security: server-side threats

- **Access control**: should prevent certain files being served.
- Complex or malicious URLs
- Denial of service attacks
- Remote authoring and administration tools
- Buggy servers, with attendant security risks
- Server-side scripting languages: C or shell CGI, PHP, ASP, JSP, Python, Ruby, all have serious security implications in configuration and execution. File systems and permissions have to be carefully designed. *That's before any implemented web application is even considered. . .*

Web programming: application security

Many issues

- **Input validation**: to prevent SQL injection, command injection, other confidentiality attacks.
- **Ajax**: beware client-side validation! Understand metacharacters at every point. Use labels/indexes for hidden values, not values themselves.
- **Output filtering**: Beware passing informative error messages.
- **Careful cryptography**: encryption/hashing to protect server state in client, use of appropriate authentication mechanisms for web accounts

Cross site scripting (XSS)

- Inserting code to be run on target server or pages returned by target server
- Common way unprotected database inserts
- Steal cookies, key logging, passwords, credit card, phishing , etc

- See code demo
  - Simple test
  - Key logging
  - Steal authentication cookie…and so much worse

SQL injection

- Inserting SQL to be run in existing SQL calls to database
- Common way unprotected database selects, inserts, updates, deletes
- Insert/update/delete records, potentially drop tables
- Return information you shouldn’t be able to access

- See code demo

Security Requirements Planning Goals

- Identify security requirements for a proposed system.
- Identify how to secure existing requirements.
- Identify and prioritize stakeholders in the system.
- Apply accountability of stakeholders to the system scope.
- Determine elements of requirements that document and assert security.

Establishing Stakeholders

- A **stakeholder** is anyone with an interest in the project or anyone affected by the project.
- Secondary stakeholders are those who are indirectly affected by the project or those who may indirectly affect the project.
- The process of **stakeholder analysis** is used to determine the members of each group.

Example Stakeholder Analysis

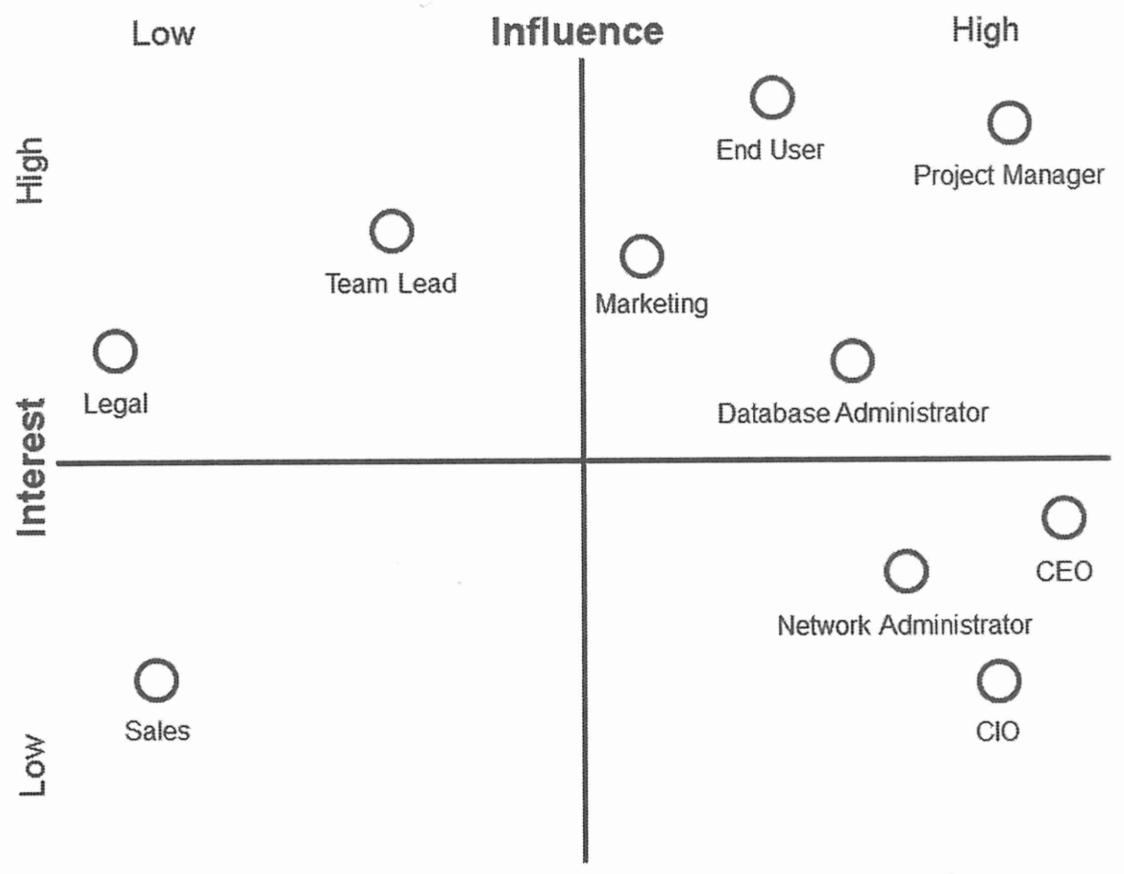


Figure 7.2

Scatter analysis of stakeholders for case project.

Point to Remember….

The earlier security is considered, the more likely it is to be implemented well. While the baseline of security considerations will be confidentiality, integrity, and availability, you can get more granular with the list you present to each of the stakeholders by including items such as the following suggestions:

1. Data privacy
2. Strict authentication and access control
3. Uptime/reliability
4. Failing safely
5. Nonrepudiation

Gathering Requirements

- A **requirement** is an outcome for the proposed system, something that it must perform or a quality it must have.

Two types of requirements:

1. A **functional requirement** is something that the system must do; it is an outcome that the system must produce as part of its useful operation.
2. A **nonfunctional requirement** is a quality or constraint for the system; this is something that must be upheld as the system operates.

Functional and Nonfunctional Security

Asking and answering the following questions will create a well-written requirement:

1. **Why should this be part of the system?**
2. **What are the constraints on this requirement?**
3. **What are the dependencies for this requirement?**
4. **Who are the stakeholders for this requirement?**

Example requirement types

- Functional

  7. "Payment options should be presented for automatic monthly renewal.

  Something the system must do

- Nonfunctional

  25. "Users can search for questions or other user profiles with a single search interface."

  Constraint that can't be more than one search interface

Gathering Requirements

- A **security requirement** is an associated protection that must be placed on some part of the system as a contingency to normal operation or a guarantee of some constraint that would otherwise violate the conditions of safe operation.

Functional and Nonfunctional Security

- Security at the requirements level is mainly a consideration of all outcomes for a functional requirement **<u>and an assessment of what happens if the constraint fails for a nonfunctional requirement</u>**.
- Requirements are written in the language of business, but that does not mean there is no room for including technical components, especially in considering constraints.

Security Requirement

- **Fail case:** This is what will happen if the requirement is not fulfilled during operation.
- **Consequence of failure:** This is the result of the fail case. When the fail case is hit for the requirement, this is where the potential outcome should be documented.
- **Associated risks:** The associated risks include sensitive information that could be compromised or revealed, domino effects to the failure of dependent requirements, or violation of laws or system specifications.

Security Requirement

- What are the exceptions to the normal case for this requirement?
- What sensitive information is included in this requirement?
- What are the consequences if the conditions of this requirement are violated?
- What happens if this requirement is intentionally violated?

Note the answer to each of these is in the context of the **specific requirement**

Establishing Scope

- **Product scope** is the collection of functional and nonfunctional requirements that will be included in the final system.
- **Project scope** refers to the work that is to be completed and is more concerned with how the project itself will be governed, such as personnel, timelines, and so on.

Establishing Scope

- **Validation testing**: asserting that the needs of the system and the needs of the stakeholders are being met with the requirements gathered.
- **Validation**: is the process of making sure the right system is being built.
- **Tradeoff-analysis**: is where both competing needs are analyzed and the best outcome for the project is decided.

Vulnerability mapping
Goals

- Construct use case and misuse case diagrams.
- Identify overlapping security concerns in a use case overview diagram.
- Construct supporting documents in UML with the addition of security concerns.
- Identify and prioritize system vulnerabilities.
- Manage the required documentation to provide a complete business specification of the system.

Use Case Construction and Extension

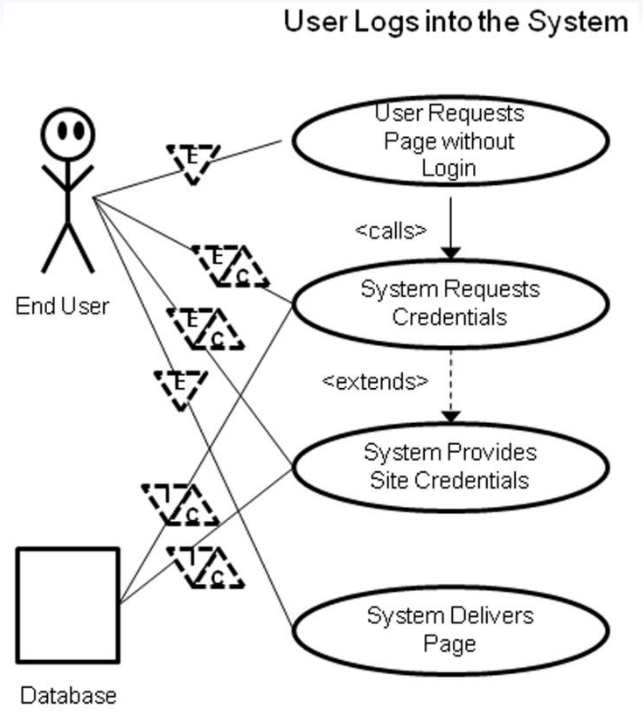
- The first step in moving from a listing of system requirements to an actualized and deployed system is the process of use case mapping.
- A **use case** is a translation of functional requirements into a visual map of activity that details the steps of arriving at a measurable system outcome in more granular and explicit fashion.

Use Case Construction and Extension

Use cases involve three primary components:

1. An **actor** is a person, external system, or entity that plays a role in the performance of the functional task described in the use case.
2. A **procedure** is a step performed to achieve the outcome of the system specified by the functional requirement.
3. An **association** is a relationship between actors and procedures. For actors and procedures, this is represented by a directional arrow specifying the instantiation of the next step in the process for the system.

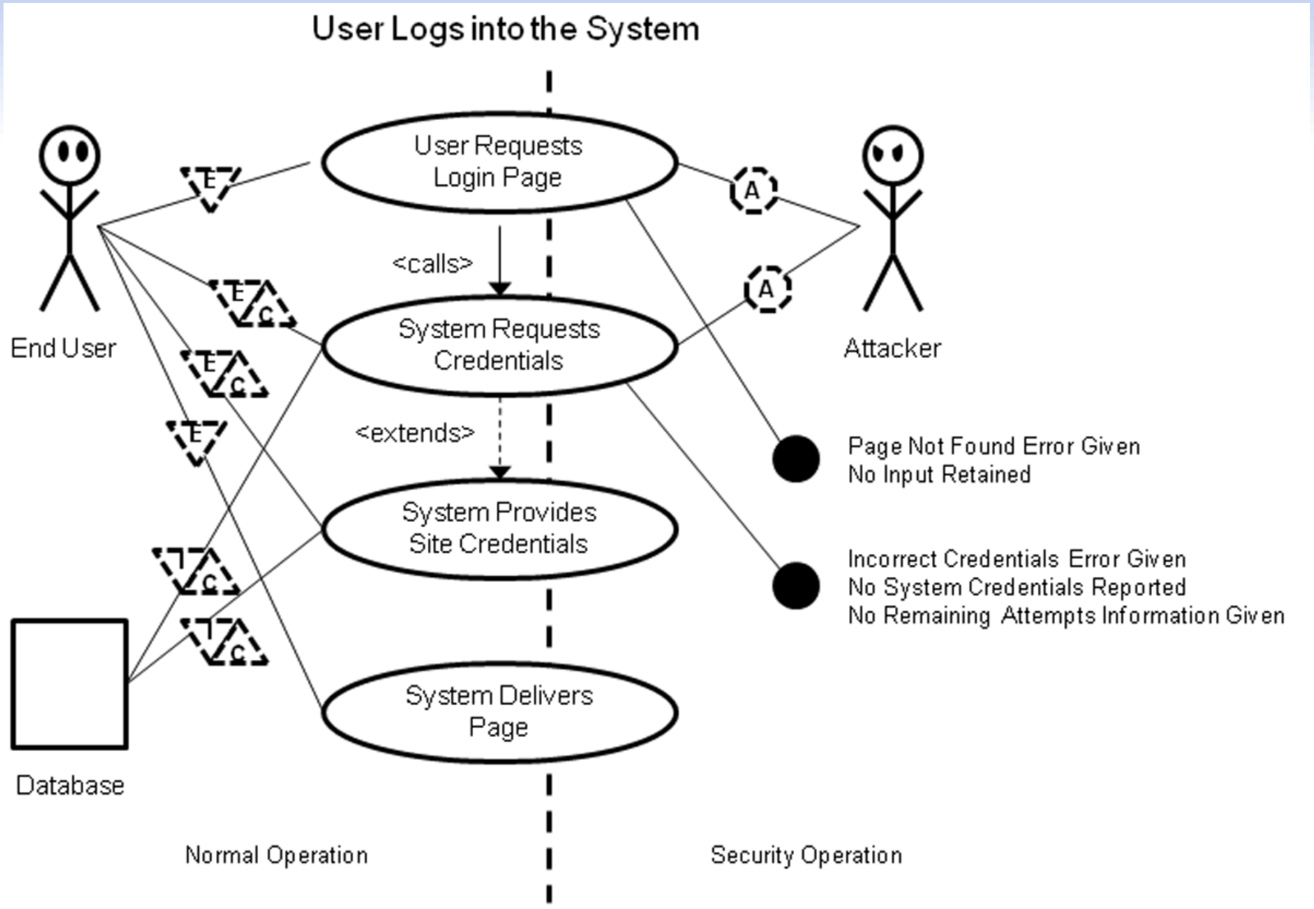
Sample Use Case


User Logs into the System
End User
Database
User Requests Page without Login
<calls>
System Requests Credentials
<extends>
System Provides Site Credentials
System Delivers Page
E
C
I

Managing Misuse

- This is an attempt to elicit security requirements by considering what a malicious actor could do within the context of the system.
- **Misuse Management Method (MMM)** is done by first, keeping all of the actors in your use case to the left of the procedures and keeping the procedures in the middle of the diagram. Draw a dotted line down through the middle of your procedures to separate the normal operating case from the diagram for malicious attacks on the system. The left side will retain the use case properties and information. The right side will contain the analysis of security needs and the procedural extensions necessary to round out the functionality in the use case to manage attacks. (The next slide has a sample)

Misuse Management Method (MMM)


User Logs into the System
User Requests Login Page
<calls>
System Requests Credentials
<extends>
System Provides Site Credentials
System Delivers Page
End User
Database
Attacker
E
C
A
Page Not Found Error Given
No Input Retained
Incorrect Credentials Error Given
No System Credentials Reported
No Remaining Attempts Information Given
Normal Operation
Security Operation

Off the Map

- After the individual use cases have been completed and the misuse information has been added to the diagrams, it is possible to start mapping the entire system.
- The easiest way to do this is to create a new **use case overview diagram** and add the central functionality defined in each of the individual use cases.

Sequence Diagrams and Class Analysis

- A **sequence diagram** is a detailed breakdown of the communication that will occur between actors and system objects or components.
- A sequence diagram is most compatible with object-oriented systems, because it allows relatively straightforward mapping from use cases by applying object models to the instantiation.
- A **class** is a template for behavior and variable usage; an **object** is an instantiation of this template.

Sequence Diagrams and Class Analysis

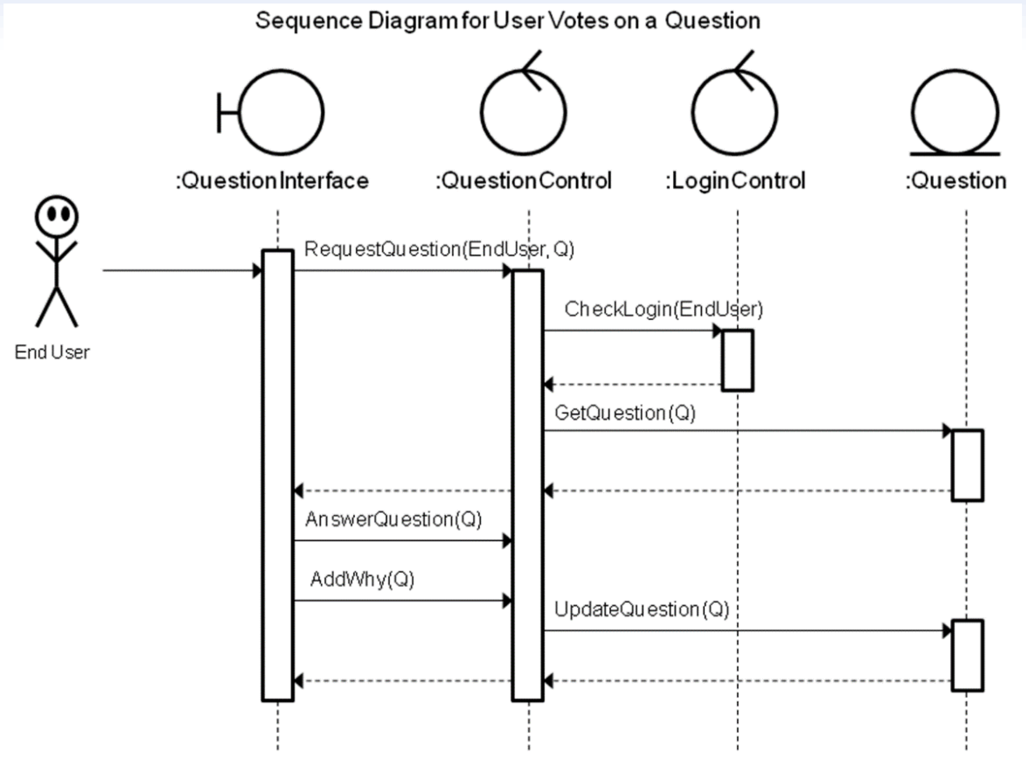
There are several types of classes that will be determined in a sequence diagram:

- An **entity class** is, broadly, a storage class. The objects it generates are the housing for most of the data in a system.
- A **boundary class** is primarily responsible for handling interactions between the actors and the system.
- A **control class** is a coordinator for the system. These classes insulate entity classes from changing business rules and policies, effectively modularizing the system to a degree.

Sequence Diagrams Process Steps

1. Choose a single use case.
2. Write out each process of the use case in detail.
3. Specify a boundary class between the system and each actor.
4. Specify a single control class for the use case.
5. Specify an entity class for each object referenced in the use case.
6. Specify a control class for any generalized use case referenced.
7. Align the use case steps next to the row of actors and classes.
8. Determine lifetimes for the objects based on a complete sequence of messages to and from the boundary class indicating a single overall transaction in the system.
9. Refine your classes for any functionality that is too complex for a single class.
10. Refine your classes for any functionality that is too complex for a single class. (Sample next slide)

Sample Sequence Diagram



Data Planning

- The goal of data planning is to establish the database construct that will be needed to support the system being developed.
- An **entity-relationship model** (**E-R model**) is a relational diagram used to establish tables, table attributes, and relationships within a system.
- The cardinality is the number of relationships that can occur from one table to another.

Knowing Your Boundaries

There are several good restrictions for boundary classes:

- A boundary class is not allowed to directly execute any input.
- A boundary class on the client machine is not allowed to divulge or contain privacy data that is not entered by the client or sent by the internal control class.
- A boundary class must authenticate the communicating control class to which it is connected.

Knowing Your Boundaries

- A control class on the outside of the trust boundary should never be allowed to interface directly with a control class on the inside of the trust boundary.
- A control class must authenticate the boundary classes from which it receives a message.
- A control class must provide authentication to the boundary class to which it is communicating.
- A control class must evaluate or process input from a boundary class before directly executing any information coming from a boundary class.
- A control class must provide authentication to an entity class from which it is requesting privacy data or mission-critical data.
- A control class that is not trusted cannot directly communicate to an entity class that is trusted.
- All data members of a control class must be private or protected.

Entity Classes

- An entity class can divulge information only to a control class.
- An entity class on a client machine may not directly access information inside the system trust boundary.
- An entity class may not communicate with a boundary class.
- An entity class housing private, confidential, or mission-critical data must authenticate the control class with which it is communicating.
- An entity class has the right to refuse to divulge information.
- An entity class is not allowed to have public data members.

Communication, Activity, and State Diagrams

- A **communication diagram** is a collaboration diagram, and is an alternate view of the sequence diagram in which all of the interactions between the classes are mapped as function calls for the class.
- An **activity diagram** is a type of support diagram in which the workflow of a use case is mapped out in flowchart format with a well-defined starting point and clear end points.
- A **state diagram** is a model of the lifespan of an object within the system.

Vulnerability Mapping

- The overall goal of performing vulnerability mapping is to determine the most likely locations within the system in development where an attacker will strike.
- To start vulnerability mapping, identify the input locations of the system, the internal communications, and the interprocess communications.

Vulnerability Mapping

The following basic classification system will work:

- **V3:** This is the highest level of vulnerability.
- **V2:** This is the moderate level of vulnerability.
- **V1:** This is the lowest priority level of vulnerability.

Development and implementation Goals

- Identify security issues in a given architecture.
- Identify vulnerabilities inherent to common programming languages.
- Implement secure programming practices.
- Perform variable and data tracking throughout a software system.
- Determine necessary documentation as it relates to security.

Architecture Decision

- The architecture decision is a critical moment for a system, much like the decision for the project scope.
- The architecture of a system involves both the hardware that will support the system and the software that will perform the system processes that have been decided in the planning phase of the system.
- The most common forms of software architecture are monolithic, 2-Tier, 3-Tier, NTier, distributed, and peer-to-peer.

Monolithic

- A **monolithic architecture** is a system that is contained on a single client machine.
- Encoded secrets should never be allowed in this type of application.
- A monolithic system could be composed of separate components that all deliver the same overall application as long as they all reside on the same machine.

2-Tier

- In the 2-Tier model of architecture, the user interface and business logic is performed on the client machine, but the data storage is handled remotely by a separate system.
- Eavesdropping on the transmission information can potentially compromise privacy or sensitive information quickly if it is not protected.

3-Tier

- The 3-Tier architecture model removes the business logic from the client end of the system.
- This alleviates some of the vulnerability to the database because the only communication to and from the database server is from within the network environment.
- Eavesdropping within the network is the biggest vulnerability here.

N-Tier

- An N-Tier architecture is a further distribution of modules at different levels of processing and storage.
- The data storage is typically still centralized, though this might not be the case if a separation of authentication levels is required to access the data.
- Eavesdropping and password cracking are high risks for this type of system. Locking down the front end from forged or malformed traffic needs to be the first line of defense.

Distributed Computing

- The distributed computing model of software architecture introduces a new paradigm of client equality.
- There may or may not be a central authority that distributes the work, but the processing is mainly completed on the client end.
- As long as the architecture accounts for managing the increasing number of clients, it is virtually infinitely scalable.
- Authentication should be a concern with this type of system in general and with a centralized manager specifically.

Software Sources

- **Commercial Off the Shelf (COTS)** system.
  - **Pros** – No development cost, functionality included, and fixed price
  - **Cons** – inherit vulnerabilities, limited ability to modify functionality, maintenance of most current version needed
- **Outsourcing** has become a popular method for external development, and the price of it has dropped considerably.
  - **Pros –** generally low cost, no in-house development needed
  - **Cons** – No input on source code other than system specs, security left to the outsourced developers not asserted from within organization, thorough review needed for liability

Software Sources

- **Open Source Software (also Free open source software aka FOSS)**
  - **Pros** – free code, functionality generally well implemented and documented on most common large projects
  - **Cons** – difficult to modify, difficult to secure any inherited vulnerabilities, maintenance of most current version needed
- **In-house development**
  - **Pros –** system will match security and functionality specified in design
  - **Cons** – high cost of development; resources needed for development and maintenance; depends on skill, expertise, and availability of development team

Watch Your Language

- There is no language in existence that has perfect security.
- The best language or languages to choose for the system are those tailored to the functionality of the proposed system.
- Mitigation techniques are also essential when a known compromise of the resource exists.

Class Security Analysis

There are several best practices for securing a class:

- **Never allow data changes by reference in external interfaces.**
- **Utilize the context of the request to determine data access.**
- **Support completion verification in data updates.**
- **Authenticate whenever prudent and possible.**

Procedural Security

Both the input and the output need the following information to be included in the documentation:

- The variables names that are required
- The data types for each variable name
- The variable names that are possible, but optional
- The data type for each optional variable
- The access type (reference or copy) for all variables, required or optional
- The specific allowed or possible combinations of variables produced

Documenting Security Procedures

The following additional information is required for documenting security in procedures and functions:

- The external resources accessed by the procedure
- The error information that is reported upon failure for the procedure
- The state of the system when the fail case has occurred

Security Procedures Best Practice

The following guidelines are best practices when it comes to procedural security:

- Be cautious of the variables that are called by reference.
- Assert a good running condition before accessing external resources.
- Do not leave expected output empty.
- Have a clear error state as part of the procedure output.
- Have a plan in place for how the system will respond to an error in the procedure.
- Monitor where and how error information is propagated.

Modular Mayhem

- The best means of code reuse are developing internal libraries for software within your organization and modularizing your code.
- When you are reusing code that comes from either a previous version of the system or an internal source, it should be subjected to code review.
- Code review is a part of penetration testing, but it should also be adopted for development.

The Life of Data

- Variable consistency and usage is a critical area of security in code.
- Performing a **first-order scope map** is simply a matter of diagramming the connections that can occur from the variable’s inception to its retirement.
- This becomes a linear process because anywhere the function calls feedback into a variable for which the map was completed, the process is finished for that branch.

First-order scope map

- Generally overkill to do for every process, but good idea for at least one example for each environment path

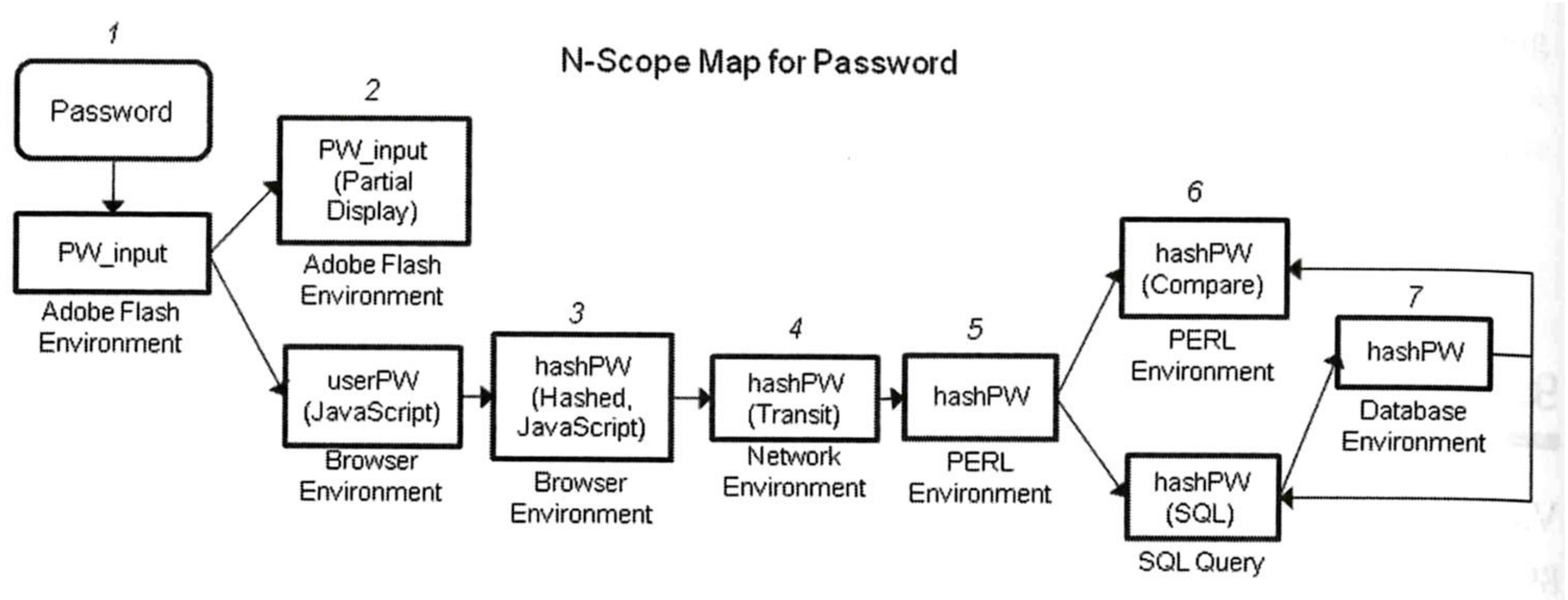


Figure 9.1

Example of an *n*-order scope map for user password entry.

Attack Surface Reduction

- For high-priority vulnerabilities (typically V3 vulnerabilities), a **threat model** is a useful construct.
- A threat model is a diagram and description that tells a story of how an attacker could exploit the vulnerability.
- Examining the overall mapping of vulnerabilities and threat models, you can generally deduce likely attack vectors.

Threat model

- Examine high priority vulnerabilities and diagram parts of potential attack to identify mitigation points

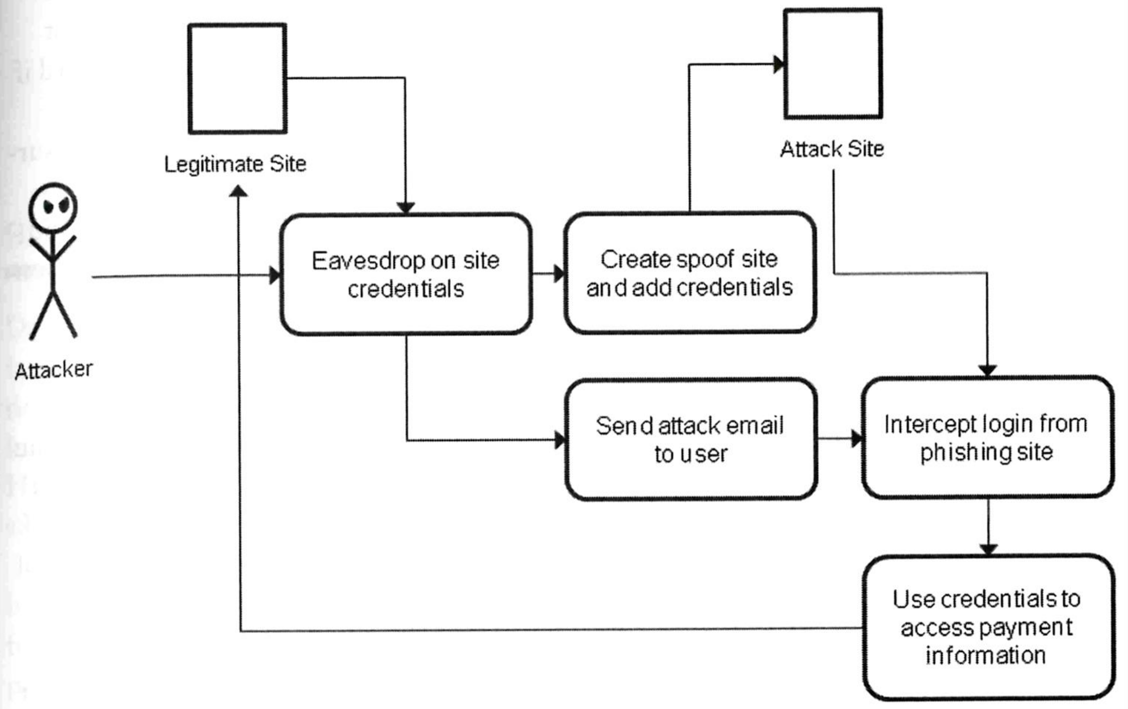

Legitimate Site
Attack Site
Attacker
Eavesdrop on site credentials
Create spoof site and add credentials
Send attack email to user
Intercept login from phishing site
Use credentials to access payment information

Figure 9.2

Example of a threat model for the sample system.

Attack Surface Reduction

When determining the likelihood of an attack vector, consider the potential usefulness of the exploited vulnerability:

- Would this compromise have monetary value?
- Does this allow additional privileges in the system?
- Does this reveal user information or privacy data?
- Does this circumvent authentication or authorization?
- Does this avoid a security mechanism?

Attack Surface Reduction

The **attack surface** is the profile of the system that is accessible to potential attackers. Here are several strategies to reduce the overall attack surface of the system:

1. **Run with the principle of least privilege**
2. **Shut down access when possible**
3. **Restrict entry into the system**
4. **Deny by default**
5. **Never directly code secrets**
6. **Quiet your error messages**

Document, Document, Document

Effective documentation will have the following qualities:

- **Sufficiency**
- **Efficiency**
- **Clarity**
- **Organization**
- **Purpose**