# CS 493
# Secure Software Systems
## Ch 9 Development and implementation


Why great care and consideration should be taken when selecting the proper password

Dr. Williams Central Connecticut State University

# 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

- When dealing with classes within a language, it is imperative to protect the necessary data that is encapsulated into them.
- When you have an array or linked list of information, care should be taken to bound the size of the variable and check any length of input that comes into an object of the class during execution.
- Consistency of inheritance is another crucial element in creating secure classes.

# Class Security Analysis

The following information needs to be documented for a class in terms of security:

- Protected data that is owned by the class
- Authentication procedures used by the class before the data is revealed
- Functions that change the value of data owned by the class
- Functions that reveal the data that is owned by the class
- The minimum guarantee of security for the data owned by the class

# 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

- Procedural programming is the use of a single imperative set of instructions from start to finish that allows the calling of sub procedures, but does not facilitate nonlinear branching through the system.
- Any procedure that takes place in your system, either as strict procedural code or as a function inside of an object, needs to have a listing of the minimum expectation of input with optional input allowed.

# 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

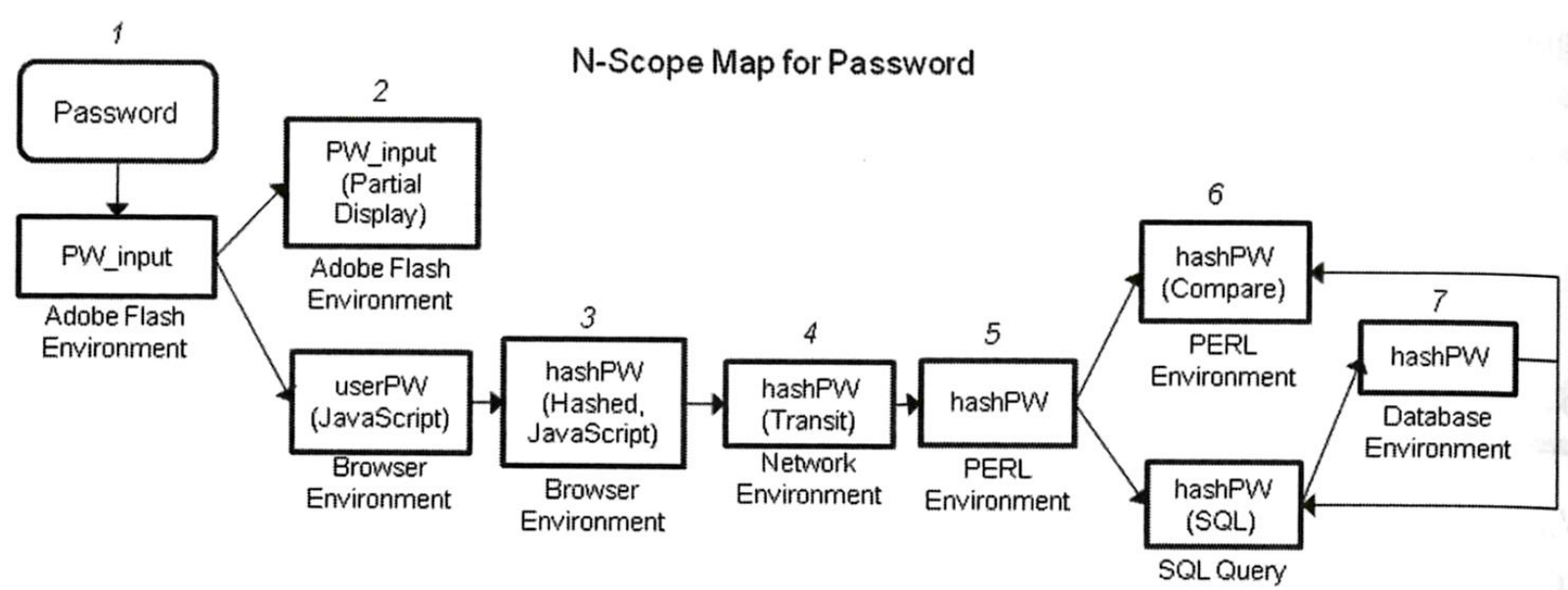


Figure 9.1

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

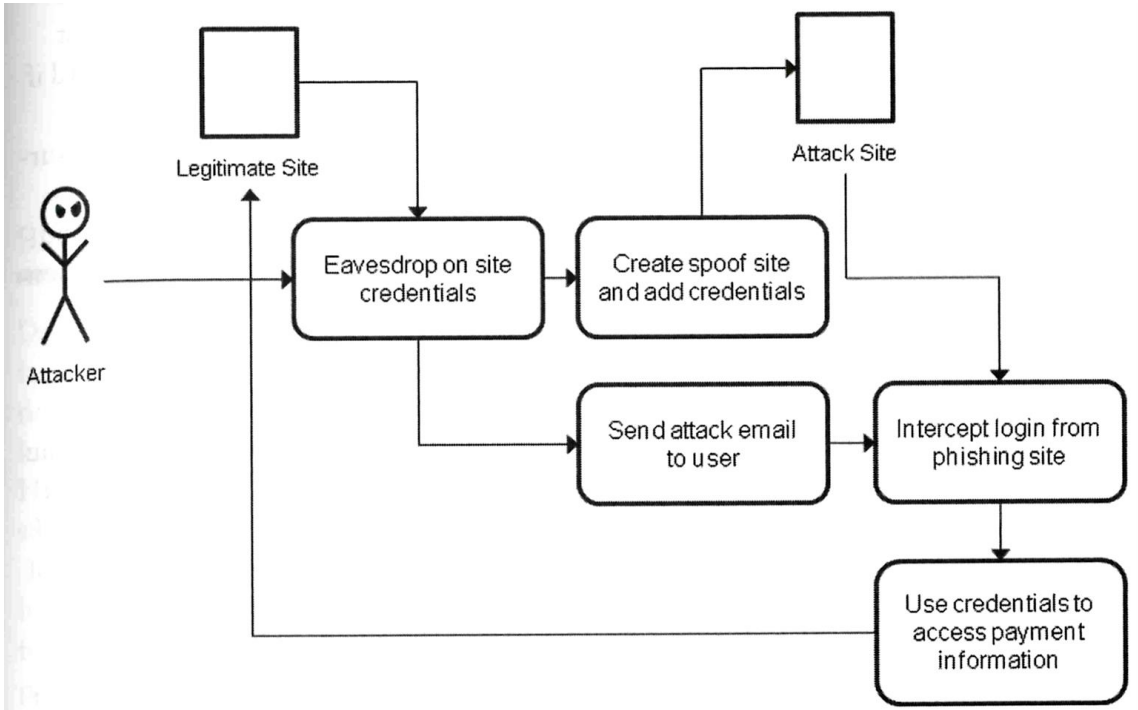


Figure 9.2

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**

# Summary

- The implementation phase is probably the most critical element of establishing security in the system.
- The attack surface of the system will grow during this phase.
- Minimizing the overall attack surface of the software and maintaining the level of security promised by the requirements are the two most critical activities for this phase.

# Group work

- Identify the languages needed for implementing the case project.
  - What are the known vulnerabilities of these languages and where does the system provide opportunity for misuse?
- Create a general strategy for secure coding practices that should be implemented regardless of language or system specifics

# In class exercises revisited

# Bob's Pizza Shack

In groups consider this scenario

Bob is a small business owner of Bob's Pizza Shack and wants to create a website to allow online credit card delivery orders

- Identify the languages needed for implementing the case project.
  - What are the known vulnerabilities of these languages and where does the system provide opportunity for misuse?

# Alice's Online Bank

In groups consider this scenario

- Alice opens Alice's Online Bank (AOB)
  - Internal use bank employees
  - Interoperates with another bank
  - Interacts with customers
    - Web
    - Mobile app
- Identify the languages needed for implementing the case project.
  - What are the known vulnerabilities of these languages and where does the system provide opportunity for misuse?

# Location based social media app

In groups consider this scenario Open source group wants to create a mobile app to allow groups to communicate/find each other in public demonstrations/protests

- Communication internet, as well as, P2P (WiFi/Bluetooth) in case internet cut off – so if person you want to contact is on other side of crowd and no internet as long as P2P network can be established with app can reach somebody outside your immediate vicinity via the P2P network
- Should be able to communicate securely messages and images to people you identify within your group (group as whole or direct)
- Should be able to share GPS location with people in group (group as whole or direct)