Unit 5 Module 3: Secure Development in an Insecure World

Dylan Lane McDonald

CNM STEMulus Center
Web Development with PHP

September 25, 2014

Outline

What is Security?

What is security?

What is Security?

What is security?

1. **Confidentiality**: the prevention of disclosure of information to unauthorized parties (known to laymen as "privacy")
2. **Integrity**: the guarantee that the information has not be altered, delayed, or otherwise had its consistency compromised
3. **Availability**: the state of having the information accessible once applicable security protocols and credentials have been provided

A secure system is one that consists of all three of these properties. These seemingly conflicting objectives are depicted in Figure 1.

Many lay people confuse "security" for solely the "confidentiality" aspect of it. As a result, many people overfocus on confidentiality and neglect integrity and availability.

Security Triad

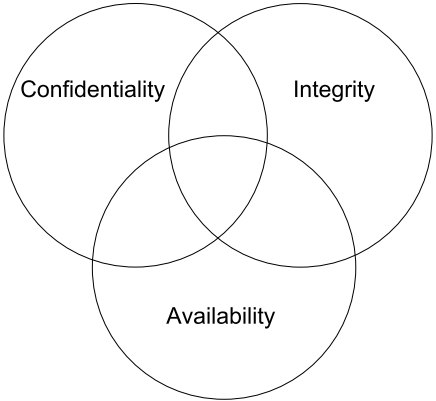

Figure 1: Security Triad

Onion Model

In order to accomplish all three objectives in Figure 1, a multi-faceted solution is necessary.

Onion Model

In order to accomplish all three objectives in Figure 1, a multi-faceted solution is necessary.

Definition

The **onion model** *is a common model illustrating the many "layers" of information security. The model, depicted in Figure 2, represents how the data can be protected by multiple strategies as it exists and interacts with a system.*

Onion Model

In order to accomplish all three objectives in Figure 1, a multi-faceted solution is necessary.

Definition

The **onion model** *is a common model illustrating the many "layers" of information security. The model, depicted in Figure 2, represents how the data can be protected by multiple strategies as it exists and interacts with a system.*

The onion model has many roles played by many players, both in terms of system roles and physical (human) roles.

Onion Model

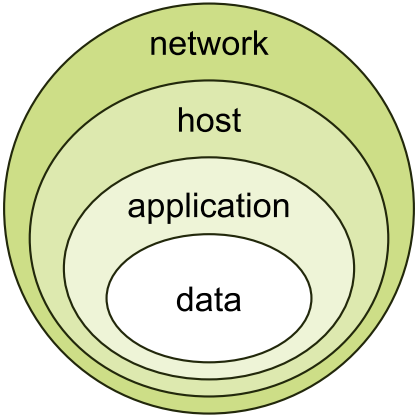

Figure 2: Onion Model

System Roles

Some of the system roles in the onion model are:

- **Firewall/Router**: prevents malicious traffic from ever contacting the application
- **Limitation of Access**: the less door one opens to the data, the less the chance the data will be compromised
- **Encryption**: scrambling data while transmitted or in storage to prevent attackers from reading data
- **Input Sanitization**: filtering inputs before they hit the application to minimize the chance of compromising the data's integrity

As a web developer, your most powerful and most used role in the onion model is...

System Roles

Some of the system roles in the onion model are:

- **Firewall/Router**: prevents malicious traffic from ever contacting the application
- **Limitation of Access**: the less door one opens to the data, the less the chance the data will be compromised
- **Encryption**: scrambling data while transmitted or in storage to prevent attackers from reading data
- **Input Sanitization**: filtering inputs before they hit the application to minimize the chance of compromising the data's integrity

As a web developer, your most powerful and most used role in the onion model is... **input sanitization**!

Human Roles

Humans are inevitably involved in the operation and development of any system. The major roles are:

- **Systems Administrator**: responsible for setting up servers and routers and keeping internal data accessible only via secure methods
- **Software Architect**: responsible for deciding on how different methods will be implemented; often, these will have large security implications
- **Software Developer**: responsible for using good coding practices and programmatically preventing malicious users from succeeding

Working as a cohesive team, these three roles can create a united front against attacks and create secure software.

Meet the Players

Security scenarios are almost always illustrated with the following players:

- **Alice**, **Bob**, & **Charlie**: ordinary end users who use the system
- **Eve**: an eavesdropper listening to Alice & Bob
- **Mallory**: a malicious user who compromises Alice or Bob

There are more players on stage, but these are by far the most common.

Meet the Players

Security scenarios are almost always illustrated with the following players:

- **Alice**, **Bob**, & **Charlie**: ordinary end users who use the system
- **Eve**: an eavesdropper listening to Alice & Bob
- **Mallory**: a malicious user who compromises Alice or Bob

There are more players on stage, but these are by far the most common.

Vitæ

Ron Rivest, **Adi Shamir** *and* **Leonard Adleman** *developed the RSA encryption algorithm, which is the basis of encryption algorithms such as SSL, PGP, and SSH.*

Hashes

A hash is the industry standard way for storing passwords.

Definition

A **hash** *is a mathematical function* $f : \mathcal{S} \to \mathbb{N}$ *such that* $H(x_1) \neq H(x_2)$ *for all* $x_1 \in \mathcal{S}$ *and* $x_2 \in \mathcal{S}$ *and that* $H^{-1}(x_1) = \emptyset$.

Notice the definition of a hash guarantees two important properties:

1. **Unique**: No two inputs will produce the same output
2. **Singular**: Given an output h, there is no way to determine which input produced h

These two reasons make hashes the ideal choice for storing passwords. Passwords are never stored in a database. Instead, given a password p, $H(p)$ is stored.

Hash Algorithms

There are two hash algorithms in use today: MD5 and SHA-2:

1. **MD5**: A fast, efficient, insecure hash algorithm. A collision (where $MD5(x_1) = MD5(x_2)$) has been found. MD5 is still present in legacy code and used as a signature where the security of the data is unimportant.
2. **SHA-2**: A slower, fully secure hash algorithm. SHA-512[1] is considered secure for storing and protecting sensitive information such as passwords and private keys.

Avoid using MD5 where possible. The US government now requires all software discontinue using MD5 and use SHA-2 instead.

[1] SHA-2 with a 512 bit output

Birthday Attack: Setup

Take a room with n people in it. What is the probability that two people will be born on the same day?

$$P_d(n) = 1 - \left(\frac{364}{365}\right)^n$$

And what is the probability two people have the same birthday?

$$P_a(n) = 1 - \frac{365!}{(365-n)! \cdot 365^n}$$

Intuitively, both $P_a(n)$ and $P_d(n)$ are quite small. However, if $n = 30$, $P_a(n) \approx 70.63\%$ and $P_d(n) \approx 7.901\%$.

Birthday Attack: Impact

Given a function $H(x)$, find two inputs x_1 and x_2 such that $H(x_1) = H(x_2)$ (i.e., a collision). How much work needs to be done to find a collision? Matching the terms to the previous slide:

1. **Birthday Problem**: hash function
2. **People**: inputs to the hash function
3. **People's Birthdays**: outputs to the hash function

The impact of the birthday attack is an exact measure of how difficult it is to find a random collision for a hash function. This is why hashes are **birthday bound**. For instance, a 128 bit hash is birthday bound at 2.2×10^{19} since, on average, it will take 2.2×10^{19} attempts to find a collision.

Symmetric Encryption

Problem: Alice wants to send a message to Bob securely.

Solution: Alice encrypts her message with a **private key**, k. She then uses k to transform the message to something Eve cannot read without previous knowledge of k. Bob receives the message and uses k to transform the message back to its original state.

Definition

The scenario just outlined is known as **symmetric encryption**. *Symmetric encryption requires that Alice and Bob have the same secret information in common, the private key k.*

Any symmetric encryption scenario is just as secure as the storage & exchange of k

Asymmetric Encryption

Problem: Alice wants to send a message to Bob securely.

Solution: Alice sends her public key P_a to Bob. Bob replies with his public key P_b. Alice then encrypts the message using her private key, K_a and P_b. Bob receives the message and decrypts the message using his private key, K_b and P_a.

Definition

The scenario just outlined is known as **asymmetric encryption**. *Asymmetric encryption is considered more secure because the public keys are by definition public and don't need to be protected. The private keys are assigned per user and one user doesn't want/need to know the other user's private key.*

Common Encryption Algorithms

Algorithm	**Type**	**Comment**
3DES	Symmetrical	Used internally in credit card systems
AES	Symmetrical	Current standard for symmetrical algorithms
Diffie-Hellman	Asymmetrical	Used in secure key exchange schemes
RSA	Asymmetrical	Underlying algorithm for PGP, SSH, & SSL

Table 1: Common Encryption Algorithms

SQL Injection

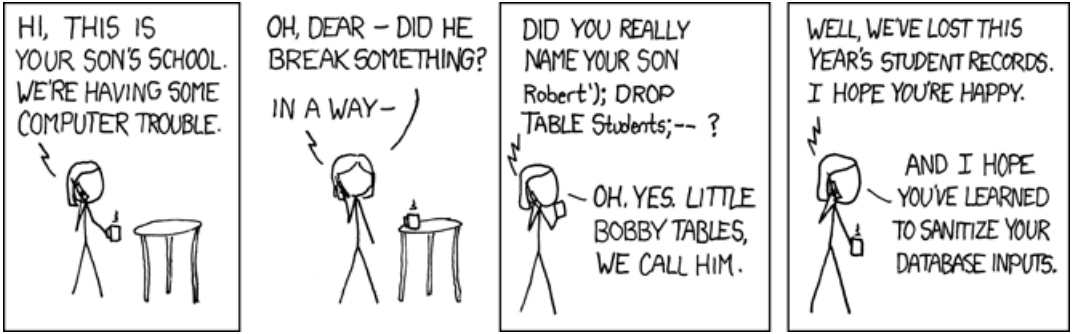

Figure 3: SQL Injection Attack [1]

SQL Injection

What the comic in Figure 3 is referencing is when HTML form fields directly reference/affect a database table. When these inputs are not sanitized, an attacker can use carefully crafted inputs to make the system perform actions that are not expected, usually by elevating an unauthorized user to a higher status.

The key solution to this problem is **input sanitization** and using prepared statements. This preprocessing step reduces the vulnerability of sites and prevents the malicious inputs from executing on the SQL server.

Cross Site Scripting (XSS)

Cross site scripting (XSS) allows an attacker to inject arbitrary code into a web site. For instance, say Alice visits Bob's site, which has a login system and a search engine. Alice searches for *foo* while logged into Bob's site. Bob's site typically displays something about the query for *foo*. Mallory knows this and sends Alice an Email enticing her to search for *foo* again. Mallory's Email exploits a weakness in Bob's input sanitizations and hijacks Alice's session and allows Mallory's full control over Alice's account.

This is known as a **non-persistent** attack and is by far the most common cross site scripting attack.

Cross Site Scripting

Extending the example from the previous slide, suppose Mallory posts a link to Bob's site on a social media site or forum. The link contains hidden code that allows Mallory to take control of any unsuspecting visitors who click the link. This is a **persistent** attack.

Cross Site Scripting

Extending the example from the previous slide, suppose Mallory posts a link to Bob's site on a social media site or forum. The link contains hidden code that allows Mallory to take control of any unsuspecting visitors who click the link. This is a **persistent** attack.

The solution to both these attacks is simple: **input sanitization**. By not trusting user inputs and stripping potential attack vectors (e.g., HTML tags), Bob's site is no longer vulnerable to cross site scripting attacks.

Cross Site Request Forgery (CSRF)

Cross Site Request Forgery (CSRF), like XSS, allows an attacker to exploit Alice's open session. In CSRF, the attacker uses a link to one site to exploit Alice's session on another site. For instance, suppose Alice has an active session at Bob Bank. Mallory sends an Email (or posts a message on social media) that contains `POST` variables that will manipulate Alice's session at Bob Bank, such as changing her password or emptying her balance.

Cross Site Request Forgery (CSRF)

Cross Site Request Forgery (CSRF), like XSS, allows an attacker to exploit Alice's open session. In CSRF, the attacker uses a link to one site to exploit Alice's session on another site. For instance, suppose Alice has an active session at Bob Bank. Mallory sends an Email (or posts a message on social media) that contains `POST` variables that will manipulate Alice's session at Bob Bank, such as changing her password or emptying her balance.

The solution to CSRF is to randomly generate a challenge/response with every `POST` operation. Mallory, at best, can see the contents of cookies, but not the contents of the session. By verifying a challenge and response every time, Bob Bank ensures that the request came from Alice.

Safe Coding Principles

What can a developer do to prevent attacks in the first place?

[2] Often referred to as "root"

Safe Coding Principles

What can a developer do to prevent attacks in the first place? **INPUT SANITIZATION**

[2] Often referred to as "root"

Safe Coding Principles

What can a developer do to prevent attacks in the first place? **INPUT SANITIZATION**

In addition, whenever a user account is created (an SQL user, an account an end-user signs up for, etc.), give that user minimal control over the system. This is known as the **Principle of Least Privilege**. [2] For instance, it is common practice to have a "super user"[2] that has full control of a database and an "application user" that is limited to `DELETE`, `INSERT`, `SELECT`, and `UPDATE`. Any schema changes, imports, or other "powerful" changes are done by the root user. The web application is only granted access to the more limited application user.

[2] Often referred to as "root"

Coding with a Safe Attitude

Following a few simple rules will facilitate a secure mindset:

- **Trust Nothing**: Always suppose the server, mySQL instance, etc. are compromised. Write your code as if nothing but malicious users use your program.
- **Keep Code Simple**: Simpler code has less of an opportunity for bugs that will potentially open security vulnerabilities.
- **Principle of Least Privilege**: Carefully consider to what extent any user accounts have in your system and limit their access to the absolute minimum.
- **Input Sanitization**: All inputs sent to a system are unsafe. This is the most effective technique in secure coding.

More detailed advice can be found from security expert Gary McGraw. [3]

Works Cited


Randall Munroe.
Exploits of a mom.
http://xkcd.com/327/.


Open Web Application Security Project.
Principle of least privilege.
https://www.owasp.org/index.php/Least_privilege.


Gary McGraw.
Gary mcgraw's 10 steps to secure software.
http://news.cnet.com/2008-1082-276319.html.