Advanced Topic: Engineering Secure Software

Author: Daler Asrorov

8 May 2015

Overview

This lecture discusses the security properties, design principles, definitions of common vulnerabilities, definitions of common vulnerabilities, methods for preventing vulnerabilities, and process of penetration testing. The lecture goes outside of agile development, and focuses on issues that can significantly impact the software development process such as losing essential data, configuration, and control.

Important Slides

Slide n°2: This slide contains an example of the costs caused by the broken security that made the BRS WorldPay lost $6.6 million dollars and where the attacker was able to get the info 1.5 million cardholders.

Slide n°3: The slide contains the graph that shows how the frequency of web-based vulnurabiliteis increased and surpasses the traditional vulnerabilities since 2008. It is caused due to increasing number of mobile devices and intensive use of web applications.

Slide n°4-5:

Unfortunately, SQL-based applications are vulnerable to some of these kinds of problems. While generally immune to buffer overflows in user software, PHP is every bit as vulnerable to filesystem and directory traversal exploits as is Perl or Python because these vulnerabilities are a product of the underlying execution environment, not the language being used. Furthermore, these kinds of violations (and other unintended consequences such as crashes) often occur because of a lack of input validation. Rather than attempting to verify its validity, input is trusted to be valid and not malicious. Additionally, sometimes valid input still represents a security violation because it exercises more privilege than the application requires to properly operate.

```
$ssn = $_POST['ssn'];                          // get ssn from web form POST data
$query = "SELECT * FROM personal WHERE ssn LIKE '$ssn'";   // construct query (notice embedded $ssn variable)
$result = mysql_query($query);                 // execute query
echo "$result\n";                              // output result
```

We can find a specific SSN using the following MySQL command:

```
SELECT * FROM personal WHERE ssn LIKE '123006789'
```

Or output all SSN numbers from the database. The application foolishly takes your input as valid and blithely plugs it in to the SQL statement:

```
SELECT * FROM personal WHERE ssn LIKE '%'
```

Solution:

```
$unsafe_variable = $_POST["user-input"];
$safe_variable = mysql_real_escape_string($unsafe_variable);

mysql_query("INSERT INTO table (column) VALUES ('" . $safe_variable . "')");
```

Slide n°7: The slide talks about the differences between prevention and detection. Prevention identifies and fixes vulnerabilities in the code, design, or requirements while detection identifies WHEN an attack is occurring at runtime and either stop its execution or log its occurrence.

Slide n°8: The slide talks about preventative software engineering techniques such as requirements (include security methods), design (make sure design principles are secure), implementation (use best coding practices, e.g. make critical methods private, variables private, etc.), testing (use penetration testing to prevent leakage of data).

Slide n°9: Lists important security properties. The main three are integrity, confidentiality, and availability. Others are privacy, anonymity, non-repudation, safety, liveness.

Slide n°10-15: Slides define and give examples of the most important properties:

- Integrity: no improper modification of data. Crucial property.
- Confidentiality: protect information from improper parties.
- Availability: make sure that system is available and responds to requests.
- Non-repudation: ability to convince a third party that an event occurred
- Safety: make sure that nothing bad ever happens
- Liveness: "something good eventual happens". Make sure that system works. It's violated by DDoS attacks.

Slide n°16: *The Protection of Information in Computer Systems* By JEROME H. SALTZER, SENIOR MEMBER, IEEE, AND MICHAEL D. SCHROEDER, MEMBER, IEEE

Slide n°17: Produce a system at any level of functionality that prevents all such anuthorized acts has proved to be extremely difficult. Some knowledgable users are able to use at least one technique that will crash the system, deny other users authorized access to stored info. But there are experiences that gave us principles that can guide the design.

Slide n°18-30: These slides talk about the security principles that will help prevent us from encountering severe security flaws:

- Economy of Mechanism: keep the design as simple as possible, emphasis on protection mechanism.

- Fail-safe defaults: base access decisions on permission rather than exclusion. Being able to refuse permission to unknown parties will prevent the info from getting stolen.
- Complete meditation: all accesses to every object must by checked by authority.
- Open design: the design should be secret, the mechanisms should not depend on the ignorance of potential attackers, but rather depend on the possession of specific, more easily protected, keys or passwords. Permits to mechanism should be reviewed by many reviewers.
- Speration of Privilege: The more protection techniques included, the better.
- Least Privilege: every program and every user should operate using the least set of priveleges necessary to complete the job, it will limit the damage that can result from an accident error and reduce the number of potential interactions among privileged programs.
- Least Common Mechanism: minimize the amount of mechanism common to more than one user.
- Psychological acceptability: User interface should be user-friendly so that he could automatically apply the protection mechanisms by himself.
- Work Factor: compare the cost of security resources with resources of a potential attacker.
- Compromise Recording: Audit logs. Mechanisms that record the compromise of information occurred and place it in more elaborate mechanisms that completely prevent loss. Difficult to guarantee discovery once security is broken, records of tampering can be undone by clever attacker.
- Implementation: know the common vulnerabilities, many vulnerabilities occur because of sloppy programming practice. Generalize solutions and apply them to prevent new and unknown vulnerabilities.

Slide n°32-34: Talks about SQL injection and what it means. Good examples are given on the 33rd slide of how it's actually done. To fight against injection, avoid the interprepeter entirely or use an interface that supports bind variables. Encode user input before passing it in.

Slide n°35-37: Talks about broken authentication and session management. Since HTTP is a "stateless" protocol that should use SSL for everything requiring authentification, there is Sessio ID that used to track state. It is a good credential to an attacker and is typically exposed on the network, in browser, in logs. Refer to slide 36 for detailed process. To fight against it, one should verify his architecture by simplifying, centralizing, and standardizing, authentication. Check SSL certificate and make sure it's protected.

Slide n°38-41: Talk about Cross-Site Scripting (XSS). Attacker sends raw data to innocent user's browser. The input is stored in database, reflected from web input, sent directly into rich JavaScript client. Attacker can steal user's session, steal sensitive data, rewrite web page, redirect user to phishing or malware site. To solve the issue, don't include user supplied input in the output page.

Slide n°42-44: Insecure Direction Object References occur with sloppy programming techniques and not using authorization methods. A common mistake only listing the "authorized" objects for the current user, or hiding the object references in hidden fields. If not resolved, users will be able to access unauthorized files or data. On slide 43 they show how the attacker can change the acct parameter and access victim's account info. On slide 44 they provide guidelines on how to avoid inscure direct object references.

Slide n°45-47: Web application rely on secure foundation, from the OS up through App server. Security Misconfiguration can cause the access by the attacker and allow him to install backdoor through missing OS or server patches, and give him access to default accounts, application functionality or data, or unused but accessible functionality due to poor server configuration.

Slide n°48-52: Sensitive Data Exposure is storing and transmitting sensitive data insecurely. It caused by failure to identify all sensitive data, all the places that this sensitive data gets stored, etc. The impact is severe: attackers access or modify confidential data or private info (credit cards, health care records, financial data, etc.) Slide 49 shows how it works. To resolve it, developers must verify their architecture (identify sensitive data, etc.), protect with appropriate mechanisms with encryption of various parts of software information, use standard algorithms and provide keys, and verify implementation (whether everything is executed according to the plan).

Slide n°53-55: Missing Function Level Control is responsible for incorrect protection of URLs.This is part of enforcing proper "authorization", along with insecure direct object references. The common mistakes are: displaying only authorized links and menu choices. The attacker can then invoke functions and services they are not authorized for and perform privileged actions. Look at slide 54 to see how it works. The attacker manually changes the directories to go up the hierarchy of access. On slide 55, it says how to solve the problem: 1) restrict access to authenticated users (if not public), enforce any user or role based permissions (if private), and 3) completely disallow requests to unauthorized page types (e.g., config files, log files, source files, etc.), verify implementation and architecture.

Slide n°56-60: Cross Site Request Forgery (CSRF) is an attack where the victim's browser is tricked into issuing a command to a vulnerable web application. The vlunrability is caused by browsers automatically including user authentication data (session ID, IP address, etc…) with each request. The attacker is then able to initiate transactions, access sensitive data, change account details. The problem is that web browsers automatically include most credentials with each request even for requsts caused by a form, script, or image on another site. Credentials provided automatically are session cookie, IP address, basic authentication header, client side SSL certificates, etc. To avoid it, we should add a secret token to all sensitive requests.

Slide n°61-63: Using Known Vulnerable Components are common… some vulnerable components can be identified and exploited with automated tools. All libraries and components should be up-to-date and developers should know what components they are using. If the attack is successful, the attacker can completely take over your data. To avoid this, we should have automation checks periodically to see if the libraries are up-to-date and checks for known vulnerabilities.

Slide n°64-67: Unvalidated Redirects and Forwards frequently include user supplied parameters in the destination URL. If they aren't validated attacker can send victim to a site of their choice and allow unauthorized function or data access. Very rare since users don't usually click on uknown links. To resolve it, we should avoid using redirects and forwards as much as we can, if used though, don't involve user parameters in defining the target URL. Validation of each parameter is required.

Slide n°68: How do we address these problems? We should develop secure code (follow best practices, use standard security components) and review our application (have an expert team review our application, and review it later ourselves using well-known guidelines).

Slide n°69-72: Pnetration testing evaluates the security of a system by simulating attacks by malicious users. It provides developers with a list of possible security issues in the tested application, and used to improve the security of the application. There is no silver bullet for securing software and it remains a best-effort activity.

Possible Questions:

List the design principles? What is integrity responsible for? Confidentiality? Availability? Nonrepudation?

What are some of the design principles?

What does injection mean? How do we prevent injection?

How does penetration testing help developers resolve security flaws?