Secure Web Development Teaching Modules[1]

Threat Assessment

Contents

1 Concepts

1.1 Software Assurance Maturity Model

Security maturity models provide a template for integrating security practices into the business functions and goals of software systems. Although some of these practices have been widely adopted, there is no industry standard in place for new comers to follow. Although these models are reference models rather than technical standards, they offer practitioners' perspective on how to incorporate security practices in software development process.

OWASP's Software Assurance Maturity Model (OpenSAMM, as in Figure 1) and Build Security In Maturity Model (BSIMM2, as in Figure 2) are two of these models. Both models map security practices

into four stages of software development. The goal is to incorporate security practices in software during its developmental stages instead of just testing for security vulnerabilities after the software being completed.

Microsoft's Security Development Lifecycle[2] (as in Figure 3) provides a similar reference model but it tights security practices with system development life cycle. In addition, Microsoft's approach provides proprietary solutions to conduct each security practice included in the model.

While considering web application security, software developers could utilize the security maturity models to determine what security practices they should consider and when the security practices can be adopted. This document will explain security practices based on OpenSAMM since it does not tight directly to specific vendors and it describes each security practice in specific activities and expected results. In addition, we will illustrate some of the practices using open sources tools for web security in the laboratory exercises.

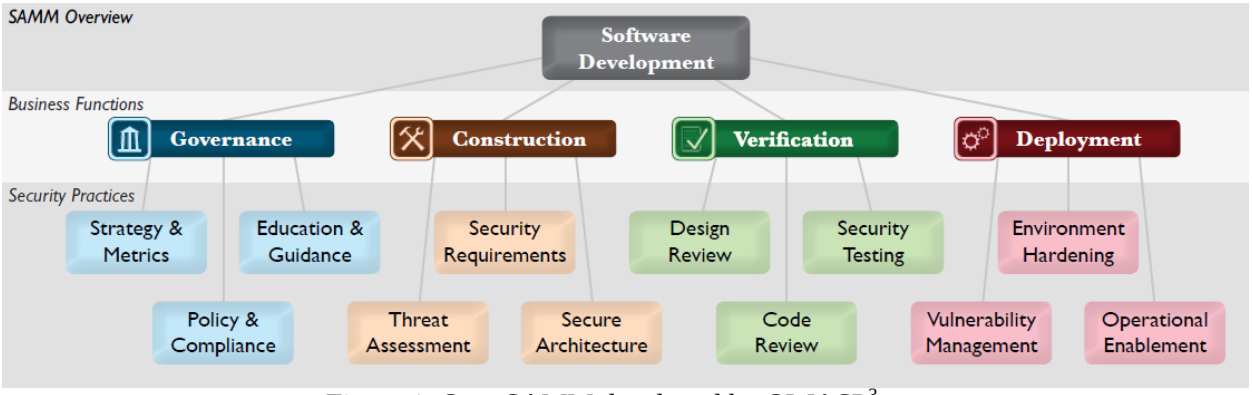

Figure 1: OpenSAMM developed by OWASP[3]

domain	practice	business goals
Governance	Strategy and Metrics	Transparency of expectations, Accountability for results
	Compliance and Policy	Prescriptive guidance for all stakeholders, Auditability
	Training	Knowledgeable workforce, Error correction
Intelligence	Attack Models	Customized knowledge
	Security Features and Designs	Reusable designs, Prescriptive guidance for all stakeholders
	Standards and Requirements	Prescriptive guidance for all stakeholders
SSDL Touchpoints	Architecture Analysis	Quality control
	Code Review	Quality control
	Security Testing	Quality control
Deployment	Penetration Testing	Quality control
	Software Environment	Change management
	Vulnerability Mgmt and Change Management	Change management

Figure 2: BSIMM2 developed by Gary McGraw[4] and Brian Chess

[2] Michael Howard and Steve Lipner, "The Security Development Lifecycle," Microsoft Press, May 2006.

[3] The Open Web Application Project (OWASP), Software Assurance Maturity Model, Version 1.0, released March 25, 2009, , available at http://www.opensamm.org/.

[4] Gary McGraw and Brian Chess, "Building Security In Maturity Model version 2 (BSIMM2)," May 2010, available at http://bsimm2.com/.

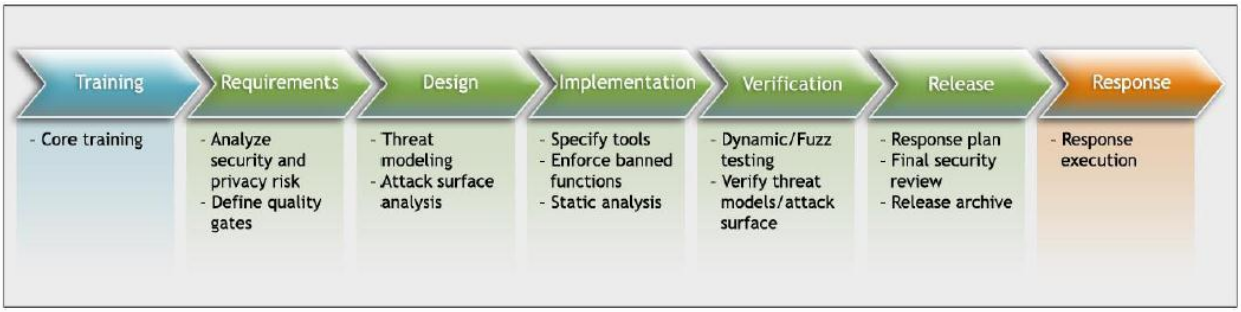

Figure 3: Microsoft's Security Development Lifecycle

1.2 Security practices for construction

Based on OpenSAMM, the software development includes four stages: governance, construction, verification and deployment. We will discuss the construction stage in this document. OpenSAMM describes it as *" the processes and activities related to how an organization defines goals and creates software within development projects. In general, this will include product management, requirements gathering, high-level architecture specification, detailed design, and implementation."* From this description, the "construction" stage is equivalent to the "analysis" and "design" stages of a system development life cycle.

The security practices involved in this stage include security requirement, security architecture and threat assessment. Security requirements specify expected software behavior related to security. For example, a web site should have user authentication methods in place is one of the requirement for web sites involving sensitive transactions. The security requirements are derived during the governance stage when the software project is planned and should be included in software development to address security in terms of the functionalities of the software. Security architecture refers to promote security designs and control over technologies and frameworks upon which software is built. For example, identifying security services and infrastructure or maintaining a list of recommended software frameworks. Threat assessment refers to identify potential attacks against software being developed, to understand the risks and to manage the risks.

1.3 Web application security risks

This section will introduce several common threats against web applications. Poor authentication, injection flaws and cross-site scripting top the list of the top ten web application security risks[5].

1.3.1 Poor Authentication and Session Management

Flawed authentication can lead to significant risks for web applications. Authentication of web servers are done through TLS/SSL but authentication of web users are done by a variety of methods, such as password login, cookies, challenge question, etc. The evidence that users need to provide for authentication can be something that they know, such as passwords or answers to challenge questions,

[5] The OWASP top 10 web application security risks, available at http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

something that they have, such as cookies or security tokens, and something whom you are, such as biometrics.

Poor implementations of user authentication methods often create vulnerabilities in web applications. For example, a security challenge question that asks users about their favorite color could be a weak authentication method by itself since attackers can easily break in the authentication by guessing a list of common colors.

Authentication using cookies without TLS/SSL is subject to session manipulation. When client site cookies are used to maintain web sessions without any encryption, attackers might be able to eavesdrop cookies and substitute their contents with unexpected contents. By doing this, attackers will be able to manipulate session information and therefore again unauthorized access to a web site.

1.3.2 Injection Flaws

Injection flaws occur when an attacker sends specific crafted data, usually malicious codes, to a trusted web application and results in unintended actions returned by the trusted web application. The impact of injection flaws could vary depending on the malicious codes. If the malicious codes were past to an operating system as system calls or shell scripts, they can result in executing unauthorized commands. If the malicious codes were past to a database as SQL commands, they can result in the access of information that is not intended to be shown. For example, on a customer directory search page, an attacker can input specific data in the html form that asks for customer name and trick a web server to reveal the entire content of the customer database.

SQL injection is an attack that exploits the vulnerability of the invalidated user input and reveals the content of the data that are not supposed to be seen by an attacker. For example, in order to search for the information of a customer “Smith” , the SQL command past from the web application to the database is below:

SELECT * FROM *user_data* WHERE *name*=’Smith’

In this example, the table name in the database is *user_data* and the attribute name for searching is *name*. A user **Smith** can type in his name and search for his account information. However, an attacker may type in **Smith’ OR ‘1’=’1**, which will result in the following SQL past to the database:

SELECT * FROM user_data WHERE name = ‘**Smith’ OR ‘1’=’1**’

Since the logical statement, 1=1, is always true, the database will return will all user data in the table *user_data*. The ramification for this attack is to examine user inputs for invalid inputs, such as the one in the example. User input validation can be done at either the web application layer or the database layer.

At the web application layer, input validation is done by comparing user inputs with either an acceptable list of inputs (whitelisting) or a list of potential malicious values (blacklisting). Whitelisting is more restricted on user inputs but blacklisting can create loopholes.

At the database layer, stored procedures can be used to validate inputs. Stored procedures are subroutines that are called upon by the database once associated data attributes are accessed. To validate user inputs, application developers have to write stored procedures for the data attributes that are related to the inputs and set up constraints for accessing these data attributes.

1.3.3 Cross Site Scripting

Cross Site Scripting (XSS) is a type of injection flaw that an attacker injects malicious codes through either the URL or the user inputs of a web page, resulting in the redirection of other naive users' web pages to a malicious web site, usually without the awareness of the naive users.

There are many forms of XSS. Figure 4 illustrates an example. An attacker injects malicious contents into a trusted web site as user inputs and the malicious contents are stored in the trusted web site's database. Once a naive user accessed the malicious contents posted on the trusted web site, such as by clicking a link posted on the trusted web site or by reading the user messages posted on a guest book, the trusted web site would either redirect the user to the attacker's web site or to run a malicious script program on the naive user's computer. In additional to injecting malicious contents into a trusted web site, an attacker can also send a naive user mails with an URL pointing to a trusted web site but embedded with malicious scripts.

The goal of XSS attacks is usually compromising users' privacy or information confidentiality. XSS is often used as a technique for phishing or cookie stealing. Once naive users tricked to access the attacker's site or to run a malicious script, they are usually asked to reveal their confidential information, such as user name, password, or credit card information. Since the naive users may think that they are communicating with the trusted web site, they would inevitably give their information to the malicious site.

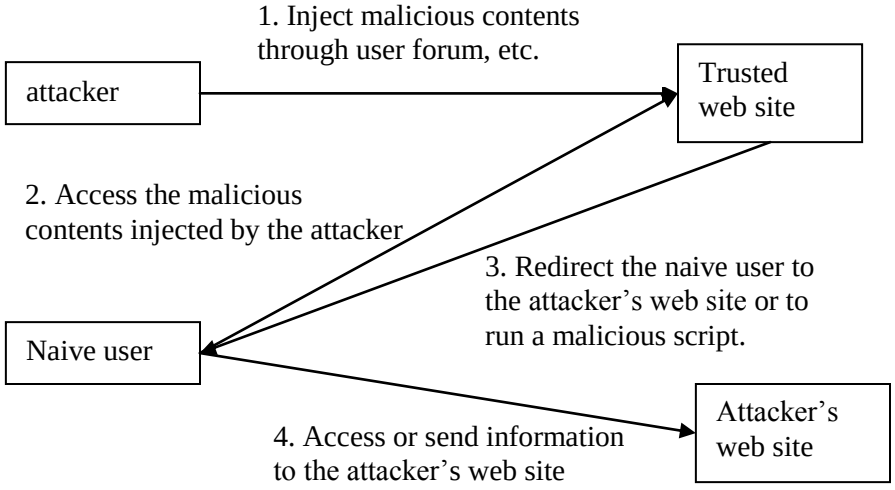

Figure4: A XSS example

Countermeasures to XSS can be categorized as server-side solutions and client-side solutions. Server-side solutions are implemented on the web applications to detect malicious codes injected by attackers and client-side solutions are used to prevent naive users from running malicious URL or scripts on the user site.

2 Lab Objectives

In this lab, you will learn to examine the following web application vulnerabilities:

- Poor authentication,

- SQL Injections, and
- Cross Site Scripting.

3 Lab Setup

1. You will use the *ubuntu 10* virtual machine for SWEET teaching modules.
2. Extract the virtual machine from *ubuntu10tm.exe*.
3. Under the folder *ubuntu10tm*, double click on *ubuntu10tm.vmx* to start the virtual machine.
4. The username is "user" and the password is "123456".

4 Lab Guide

4.1 Virtual Machine Startup

Step1: After logging in, you will see the Ubuntu desktop interface. The virtual machine runs Linux as an independent virtual computer separate from the host operating system (i.g. Windows).

Step2: Please spend a little time to familiarize yourself with the Linux interface and try the following.

1) Swap back and forth between your Linux virtual machine and the host machine (i.g. Windows). The Linux is run within its own VMware window.

2) Explore the menu bar of the Linux GUI on top of the VM window. The menu bar includes Applications (similar to Windows Start Panel), Places (all devices and storage), and System (Linux system functions).

3) To copy a file from your host machine to the VM, you can drag and drop the file between the two platforms or vice versa.

4.2 Starting WebGoat

Step1: The web server has already been pre-installed and all you need to do is run the program.

Step2: From the menu bar on the top of the VM, select Applications > Accessories > Terminal. This will open up a Linux shell command terminal.

Step3: Run the following commands to start the web server.

cd ~/tools/tomcat/bin

sh startup.sh

Step4: Use the Firefox browser to connect to the Apache Tomcat server which hosts WebGoat. Before connecting, make sure the browser is using the system proxy settings.

a) On the menu bar of the browser. Select Edit > Preferences > Advanced > Network Tab > Settings.

b) Make sure the Use system proxy settings radio button is selected. Click OK and Close.

Step 6: Go to the Apache server homepage on your VM by entering http://localhost:8080/ into the Firefox address bar.

Tip: When running a program on port 80, you do not have to specify the port number as it is the default HTTP port used by most web applications and servers.

7. Paste a screenshot of the Apache Tomcat welcome page from the browser.

_____

4.3 Setup the Proxy Server

We will investigate the web traffic between your browser and the Web server using a web proxy called Paros. All web communication between the browser and the Web server will be sent to Paros (the proxy server) first before it reaches the appropriate destination.

Step1: To start Paros, you need another Linux terminal window. Select Applications > Accessories > Terminal. Run these commands in the terminal window:

```
cd ~/tools/paros
sh startserver.sh
```

The Java-based Paros will execute and you will be greeted with its interface.

Step2: Now, you will need to change the proxy server settings in Firefox to redirect the web traffic to the proxy server.

1) The proxy server is run under 127.0.0.1 and port 8080. On Paros menu, go to Tools > Options > Local Proxy, do the following settings

- Address 127.0.0.1
- Port: 8008 (you may need to change it from 8080 to 8008)

2) Go back to your browser. Select Edit > Preferences > Advanced > Network Tab > Settings.

3) Select the Manual Proxy Configuration radio button.

4) Enter these values into the fields: HTTP: 127.0.0.1 Port: 8008 (the proxy settings need to match the setting on Paros)

5) If there are any values in the No Proxy For: text field, delete them. This is important to make the proxy work successfully.

Step 3: You have just enabled all HTTP traffic generated by Firefox to be sent to the running Paros proxy server which can analyze HTTP traffic before it is sent off to its final destination. If you reload http://localhost:8080/ on the browser, you will see Paros capture the web content sent between the server and the browser.

4.4 Web Goat Login

Step1: Go back to the browser in Ubuntu. Browse to http://localhost:8080/WebGoat/attack.

Step2: When prompted for login information, the user name is guest and the password is guest. On the WebGoat home page, click Start WebGoat.

Step3: Go back to Paros. You should see Paros logs of the connection between the browser and WebGoat. What is the HTTP command used in the first transaction between the browser and WebGoat? _____ What is the HTTP version? _____

Step4: On the left side of the screen is a list of WebGoat exercises you can try. We will try the first lesson, click on "General" and underneath it choose "HTTP Basics".

Step5: This WebGoat exercise will accept a value in the text box that you enter and reverse it. In the "Enter your name" textbox, type your name and click on Go.

Step6: You can see the parameters sent between your browser and WebGoat with Paros.

a) In Paros, look under Sites > http://localhost:8080 > WebGoat > POST:Attack(Screen,menu).

b) Change the view from Raw View to Tabular View.

*You can see the session ID by looking under the Request tab in Paros next to the Cookie: reference.

Step7: There are two parameters in the transactions using POST: person and SUBMIT.

The value for person is _____ and the value for submit is _____.

Explain the functions of the POST command with the two parameters. _____

Step8: You can also see the solution for the exercise by clicking on Show Solution in WebGoat. For all the exercises below, you can either read instructions from this manual or read the instructions from Show Solution (We use Paros instead of WebScarab in our lab).

4.5 Authentication Flaws in Login

With this exercise, you are going to exploit a common weakness in password retrieval systems. Web applications such as Web e-mail and Web forums provide their users the ability to retrieve a forgotten password with security questions. The problem is that there is no criterion to the strength of the security answers and some questions are overly easy to guess.

Step1: Click on Authentication Flaws and underneath select Forgot Password in the left pane of WebGoat. This exercise will show how vulnerable a bad authentication scheme is.

Step2: The goal here is to retrieve the password of another user. You are going to retrieve a password from a regular user with the username 'webgoat'. Type in the user name into the *User Name textbox and hit Submit. You are presented with an over simplistic security question of "What is your favorite color?" Try different colors and see what happens when you enter a wrong color as the security answer. Enter "red" and you should be able to retrieve the password for the 'webgoat' user.

In the next step you will try to retrieve the password for the WebGoat administrator account. Try to complete it without any hints or help.

Step2: The goal here is to retrieve the password of another user. You are going to retrieve a password from a regular user with the username 'webgoat'.

1) Type in the user name into the User Name textbox and hit Submit.

2) You are presented with an over simplistic security question of "What is your favorite color?"

3) Try different colors and see what happens when you enter a wrong color as the security answer.

4) Enter "red" and you should be able to retrieve the password for the 'webgoat' user.

* If you have trouble breaking into the system, see Show Solution in the WebGoat menu.

What is the administrator's user name and what is the password? _____

Describe one method to improve the login security here. _____

Explain what you have learned from this exercise. _____

4.6 Injection Flaws – String SQL Injection

SQL injection is used to gain access, extract data or compromise secured databases. The methods behind an attack are simple and the damage dealt can range from inconvenience to system compromise.

Step1. Click on Injection Flaws and underneath select String SQL Injection in the left pane in WebGoat.

Step2: The exercise first prompts you to enter the last name Smith (SQL is case sensitive). A simple SQL query to a database would looks like this

SELECT * FROM user_data WHERE last_name = 'Smith'

In this case, Smith is the value you enter in the "Enter your name" textbox. The SQL will select all information about the user 'Smith' from the database.

How many entries have you obtained? _____

Are they all information regarding the user "Smith"? _____

Step3: Let us look at the following SQL command

SELECT * FROM user_data WHERE name = '**Smith' OR '1'='1**'

This SQL command will ask the database to show all user information since '1'='1' is always true. String SQL injection exploit the vulnerability that a database that does not conduct checks on constraints so that attackers can inject SQL commands through the regular user interface.

Step4: Now, enter **Smith' OR '1'='1** in the "Enter your last name" textbox.

Step5: Paste a screen shot of your results. _____

Step6: What is the security implication of your results? _____

Step 7: Describe a method to fix the vulnerability in this exercise. _____

4.7 Cross Site Scripting (XSS) – Stored XSS attack

Stored XSS attacks allow users to create message content that could cause another user to load an undesirable page or undesirable content when the message is viewed or accessed. In this exercise you will create such a message.

Step1: Click on Cross-Site Scripting (XSS) and underneath select Stored XSS Attacks in the left pane in WebGoat.

Step2: In the title text box, type "XSS example"

Step3. In the message text box, copy and paste the following HTML content.

<script language="javascript" type="text/javascript">alert("Ha! Ha! You are hacked! ");</script>

Step4. Click on Submit. You will click on the message you have just posted under message list.

Step5. What will happen if someone clicks on the message you have just posted? _____

Step6. Paste a screenshot of the results below. _____

Step 7: What is the security implication of your results? _____

4.8 Authentication Flaws in Session Management

This exercise illustrates the vulnerability of session management using Cookies. Using a proxy server, the attacker will be able to trap important information regarding transaction authorization information, such as authorization header and cookies. By manipulating these login information, the attacker can re-login as a different user.

Step1: To complete this exercise, you need to observe what HTTP requests are sent between the browser and the server. You can trap the web requests at the proxy server. To do this, on Paros, click on the **Trap tab**, check **Trap request**. This will trap the HTTP request at the proxy server, while the server waits for the HTTP request to arrive, letting the proxy server be the "man-in-the-middle" and giving the proxy server the ability to manipulate the HTTP request before letting the request continue to its destination.

Step2: Click on Authentication Flaws and underneath select Basic Authentication on the left pane in WebGoat.

Step3: Read the WebGoat description of the exercise and click on Submit. You will need to setup Paros before clicking Submit to trap the request as mentioned previously in order to continue.

Step4: Go to Paros and look under Sites > http://localhost:8080 > WebGoat > POST:Attack(Screen,menu)(SUBMIT,person). You will see your HTTP request intercepted at the proxy server and you can see the HTTP command you have just submitted. The HTTP header contains a basic authentication header called "Authorization" within the HTTP request. You should be able to see its value on the HTTP header window in Paros. The value should be some code following "**Authorization: Basic"** in the HTTP header request.

Step5: Write down the code here: _____

Step6: The code is encoded in BASE64 and we need to decode it to see the ASCII value. Use the decoder under Tools, Encoder/Hash. Decode it using BASE64 decode.

Step7: What is the plain text value of the authentication header that you have just decoded?
_____

Step8: Now, uncheck "**Trap request**" and click **Continue** to let the HTTP request go through.

Step9: Go back to WebGoat. Enter the name of the authentication header (**Authorization**) and the value of it (**the code you have just decoded above**) into the appropriate fields. Click on Submit.

Step10: Once you have successfully finished this exercise, paste a screen shot below.

_____

Step11: WebGoat will ask you to further investigate authentication by logging in again using the username basic with password basic. Authentication header and Cookie (JSESSIONID) are the information that HTTP uses to manage a session. You are currently logging in as guest. The server can recognize you since the browser sends the server the same authentication header and JSESSIONID every time. If you want the server to ask you for a re-login, you need to corrupt your authentication header and JSESSIONID.

Step12: We can use Paros to alter the HTTP request and corrupt the authentication header and JSESSIONID. Go to Paros, check "Trap request" to intercept the HTTP command.

Step13: Go back to WebGoat, click on Basic authentication. Go back to Paros. You can see the intercepted HTTP command under the Trap tab. **You can then delete the value after Authorization and Cookie**. Click **Continue** to send the HTTP command to the server.

Step14: You will be prompted for re-login. You can now re-login as user **basic** using password **basic.**

Step15: Go to Paros. You will see the new authorization code for the user but the cookie (JSESSIONID) is the same because the browser sent the value that was cached. So, corrupt the Cookie by changing the JSESSIONID value to =novalidsession, **uncheck Trap request and click Continue**.

Step16: WebGoat losses all the information about the user "guest" and regards you as a new user "basic". Now login as the user "basic" using the password "basic" and click on Start WebGoat to restart the exercise. Click on Basic Authentication to complete this exercise.

Step17: Paste a screen shot below that shows you have completed the Basic Authentication exercise. _____

Step 18: Describe what you have learned from this Exercise. _____

Change setting in Paros back to port 8080 such as tools->options->Local Proxy->Address 127.0.0.1 port: 8008

4.9 Turn off virtual machine

1. Change the settings in Paros back to port 8080 such as tools->options->Local proxy->Address 127.0.0.1 port: 8080
2. Click on File| Exit to close Paros.
3. **Important Note**: After finishing this exercise, you should reset *Firefox* proxy setting so it stops using the proxy server. Otherwise you would not be able to visit web sites without running the proxy server. To do so, Launch your Firefox web browser, and follow its menu item path "Edit|Preferences|Advanced|network Tab|Settings button" to reach the "Connection Settings" window. Check the "Use System Proxy Settings" checkbox.
4. Run the following commands to stop the web server.

   cd ~/tools/tomcat/bin

   sh shutdown.sh

Close all the windows. Turn off the virtual machine by clicking the power button on the upper-right corner.