ARIA Week 11 Tutorial 1:

Week 11 - April 13th, 2012

PART 1: Security Plans for Web Applications

SUMMARY:
We're going to build a security plan for your final project, covering all the surface area

REQUIREMENTS:
You will be working with your own web application for the final project. Work through the following with your particular project in mind.

OUTLINING THE SURFACE AREA

Figuring out what your potential invasion points are is key to keeping your application and data protected.

Forms
Hidden Fields
POST/GET parameters
Cookies
Sessions
Cross-site scripting
SQL injections
DOS (Denial of Service)
Authentication
Cryptography
Buffer Overflows

If you're not sure what these are or how to prevent or test for them, look back at notes from class or follow along below:

Some key terms used in security testing

What is "Vulnerability"?
This is a weakness in the web application. The cause of such a "weakness" can be bugs in the application, an injection (SQL/ script code) or the presence of viruses.

What is "URL manipulation"?

Some web applications communicate additional information between the client (browser) and the server in the URL. Changing some information in the URL may sometimes lead to unintended behavior by the server.

What is "SQL injection"?

This is the process of inserting SQL statements through the web application user interface into some query that is then executed by the server.

What is "XSS (Cross Site Scripting)"?

When a user inserts HTML/ client-side script in the user interface of a web application and this insertion is visible to other users, it is called XSS.

What is "Spoofing"?

The creation of hoax look-alike websites or emails is called spoofing.

Access/Authentication

Access security is implemented by **'Roles and Rights Management'.** It is often done implicitly while covering functionality, e.g.in a Hospital Management System a receptionist is least concerned about the laboratory tests as his job is to just register the patients and schedule their appointments with doctors. So, all the menus, forms and screen related to lab tests will not be available to the Role of 'Receptionist'. Hence, the proper implementation of roles and rights will guarantee the security of access.

How to Test: In order to test this, thorough testing of all roles and rights should be performed. Tester should create several user accounts with different as well multiple roles. Then he should use the application with the help of these accounts and should verify that every role has access to its own modules, screens, forms and menus only. If tester finds any conflict, he should log a security issue with complete confidence.

Data Protection:

Sensitive data is very important to protect. Ensure that **a user can view or utilize only the data which he is supposed to use**. This is also ensured by roles and rights e.g. a TSR (telesales representative) of a company can view the data of available stock, but cannot see how much raw material was purchased for production.

Testing of this aspect is already explained above. Another aspect of data protection is related to **how that data is stored in the database**. All the sensitive data must be encrypted to make it secure. Encryption should be strong especially for sensitive data like passwords of user accounts, credit card numbers or other business critical information.

Third and last aspect is extension of this second aspect. Proper security measures must be adopted when flow of sensitive or business critical data occurs. Whether this data floats between different modules of same application, or is transmitted to different applications it must be encrypted to make it safe.

How to Test Data Protection: The tester should query the database for 'passwords' of user account, billing information of clients, other business critical and sensitive data and should verify that all such data is saved in encrypted form in the DB. Similarly (s) he must verify that between different forms or screens, data is transmitted after proper encryption. Moreover, tester should ensure that the encrypted data is properly decrypted at the destination. Special attention should be paid on different 'submit' actions. The tester must verify that when the information is being transmitted between client and server, it is not displayed in the address bar of web browser in understandable format. If any of these verifications fail, the application definitely has security flaw.

Brute-Force Attacks:

Brute Force Attack is mostly done by software tools. The concept is that using a valid user ID, s**oftware attempts to guess the associated password by trying to login again and again.** A simple example of security against such attack is account suspension for a short period of time as all the mailing applications like 'Yahoo' and 'Hotmail' do. If, a specific number of consecutive attempts (mostly 3) fail to login successfully, then that account is blocked for some time (30 minutes to 24 hrs).

How to test Brute-Force Attack: The tester must verify that some mechanism of account suspension is available and is working accurately. (S)He must attempt to login with invalid user IDs and Passwords alternatively to make sure that software application blocks the accounts that continuously attempt login with invalid information. If the application is doing so, it is secure against brute-force attack. Otherwise, this security vulnerability must be reported by the tester.

SQL Injection and XSS (cross site scripting):

Conceptually speaking, the theme of both these hacking attempts is similar, so these are discussed together. In this approach, **malicious script is used by the hackers in order to manipulate a website**. There are several ways to immune against such attempts. For all input fields of the website, field lengths should be defined small enough to restrict input of any script e.g. Last Name should have field length 30 instead of 255. There may be some input fields where large data input is necessary, for such fields proper validation of input should be performed prior to saving that data in the application. Moreover, in such fields any html tags or script tag input must be prohibited. In order to provoke XSS attacks, the application should discard script redirects from unknown or untrusted applications.

How to test SQL Injection and XSS: Tester must ensure that maximum lengths of all input fields are defined and implemented. (S)He should also ensure that defined length of input fields does not accommodate any script input as well as tag input. Both these can be easily tested e.g. if 20 is the maximum length specified for 'Name' field; and input string "<p>thequickbrownfoxjumpsoverthelazydog" can verify both these constraints. It should also be verified by the tester that application does not support anonymous access methods. In case any of these vulnerabilities exists, the application is in danger.

Service Access Points (Sealed and Secure Open)

Today, businesses depend and collaborate with each other, same holds good for applications especially websites. In such case, both the collaborators should define and publish some access points for each other. So far the scenario seems quite simple and straightforward but, for some web based product like stock trading, things are not so simple and easy. When there is large number of target audience, the access points should be open enough to facilitate all users, accommodating enough to fulfill all users' requests and secure enough to cope with any security-trial.

How to Test Service Access Points: Let me explain it with the example of stock trading web application; an investor (who wants to purchase the shares) should have access to current and historical data of stock prices. User should be given the facility to download this historical data. This demands that application should be open enough. By accommodating and secure, I mean that application should facilitate investors to trade freely (under the legislative regulations). They may purchase or sale 24/7 and the data of transactions must be immune to any hacking attack. Moreover, a large number of users will be interacting

with application simultaneously, so the application should provide enough number access point to entertain all the users.
In some cases these **access points can be sealed for unwanted applications or people**. This depends upon the business domain of application and its users, e.g. a custom web based Office Management System may recognize its users on the basis of IP Addresses and denies to establish a connection with all other systems (applications) that do not lie in the range of valid IPs for that application.

Password cracking:

The security testing on a web application can be kicked off by "password cracking". In order to log in to the private areas of the application, one can either guess a username/ password or use some password cracker tool for the same. Lists of common usernames and passwords are available along with open source password crackers. If the web application does not enforce a complex password (e.g. with alphabets, number and special characters, with at least a required number of characters), it may not take very long to crack the username and password.
If username or password is stored in cookies without encrypting, attacker can use different methods to steal the cookies and then information stored in the cookies like username and password.

URL manipulation through HTTP GET methods:

The tester should check if the application passes important information in the querystring. This happens when the application uses the HTTP GET method to pass information between the client and the server. The information is passed in parameters in the querystring. The tester can modify a parameter value in the querystring to check if the server accepts it.
Via HTTP GET request user information is passed to server for authentication or fetching data. Attacker can manipulate every input variable passed from this GET request to server in order to get the required information or to corrupt the data. In such conditions any unusual behavior by application or web server is the doorway for the attacker to get into the application.

SQL Injection:

The next thing that should be checked is SQL injection. Entering a single quote (') in any textbox should be rejected by the application. Instead, if the tester encounters a database error, it means that the user input is inserted in some query which is then executed by the application. In such a case, the application is vulnerable to SQL injection.
SQL injection attacks are very critical as attacker can get vital information from server database. To check SQL injection entry points into your web application, find out code from your code base where direct MySQL queries are executed on database by accepting some user inputs.
If user input data is crafted in SQL queries to query the database, attacker can inject SQL statements or part of SQL statements as user inputs to extract vital information from database. Even if attacker is successful to crash the application, from the SQL query error shown on browser, attacker can get the information they are looking for. Special characters from user inputs should be handled/escaped properly in such cases.

Cross Site Scripting (XSS):

The tester should additionally check the web application for XSS (Cross site scripting). Any HTML e.g. <HTML> or any script e.g. <SCRIPT> should not be accepted by the application. If it is, the application can be prone to an attack by Cross Site Scripting.
Attacker can use this method to execute malicious script or URL on victim's browser.
Using cross-site scripting, attacker can use scripts like JavaScript to steal user cookies and information stored in the cookies.
Many web applications get some user information and pass this information in some variables from different pages.
E.g.: http://www.examplesite.com/index.php?userid=123&query=xyz
Attacker can easily pass some malicious input or <script> as a '&query' parameter which can explore important user/server data on browser.

Web Form Validation Rules of Thumb:

- Never omit server-side validation.
- Don't provide confusing validation feedback. It should clearly communicate the errors and ways to fix them.
- Don't let users think about what information is required, always clearly mark required fields.
- Never provide validation feedback on a single page or in a popup alert.
- Don't use dynamic effects as compensation for a badly designed form. Fancy effects won't hide a poorly designed web form.
- If you use Captcha, don't forget to provide audio support and enable users to "reload" the Captcha.
- Don't forget to inform users when the form was completed successfully. It is as important as a good validation feedback.

PRIORITISING THE SECURITY ISSUES

Next you're going to have to prioritize your application's security issues. Given the amount of time you have and the danger of each one, give two ratings:

Severity -- This is how much damage could this do to the application if not addressed. A 1 means it would barely affect the application. A 10 means that if ignored, large chunks of data could be compromised or accessed.

Complexity -- This is how much work it is to address the bug or security hole. A simple issue with a rating of 1 might take minutes to fix. A higher complexity issue with a rating of 10 might mean rebuilding a database or more, and might take weeks to fix.

Once you have the severity and complexity of the issues, you can figure out which ones are going to be your low-hanging fruit: the more severe and less complex issues which you can easily and quickly address.

SCHEDULING

The last part of your security plan is the implementation plan and schedule. We've discussed in lecture that security is "a horizontal" work item - it is in progress throughout the application life cycle, and not something that gets tacked on at the end.

Now you need to look at your schedule for your app & figure out when it makes sense to do your security work.

You can use a Gantt chart to line up dependencies. For example, you can't test SQL code injection until you have your database set up. Etc.

[BONUS]

If you've gotten this far, go ahead and start implementing the "low-hanging fruit" aspects of your application if you can. If you don't have enough content in your app, work on that first then.