CIS 418/518 – Secure Software Engineering
Web Application Security

Jagadeesh Nandigam

School of Computing
Grand Valley State University
nandigaj@gvsu.edu

Outline

1. Web Applications and Security
2. Strategies for Securing Web Applications
3. Input and Output Validation for the Web
4. HTTP Considerations
5. Maintaining Session State

Web Applications and Security

- Securing Web applications is tricky for a number of reasons.
- Due to the ubiquity of networking, users (both normal and malicious) have easy access to Web applications. No physical access to the application is needed.
- We cannot assume that a request sent to a Web application will be benign.
- HTTP is a request-response protocol originally designed to exchange or transfer hypertext resources over the Internet. It was not designed with applications in mind and certainly not for building secure applications.

Web Applications and Security

- HTTP is a stateless protocol. A Web application has to maintain session state using hidden fields within web forms, cookies, or session identifiers. An attacker can access or modify these to exploit security vulnerabilities in a Web applicaiton.
- A Web application not only has to defend itself against malicious users, but it has to defend honest users against malicious users too.
- A Web application can't afford to trust its clients (honest or malicious)!!

Strategies for Securing Web Applications

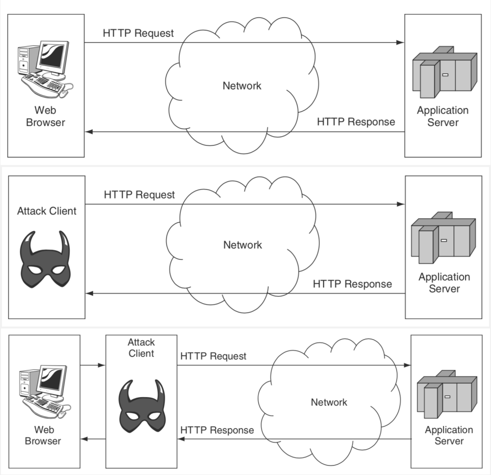
- Input and Output Validation for the Web
- HTTP Considerations
- Maintaining Session State

Input and Output Validation for the Web

- Attackers are not limited to the values they can enter into the Web page or even the values a standard Web browser is capable of generating.
- Attackers can do a lot to learn and exploit vulnerabilities in a Web application
  - Examine the Web page (via Show Source option in the browser) to find information useful for an attack.
  - Enter malicious data into form fields
  - Change request/response headers
  - Change cookies, hidden fields, post parameters
  - Post URLs in the wrong order, at the wrong time, and for the wrong reasons
  - Change any other aspect of an HTTP request

Input and Output Validation for the Web

Attackers can bypass standard browser behavior by communicating with the Web application using an attack client:



Input and Output Validation for the Web

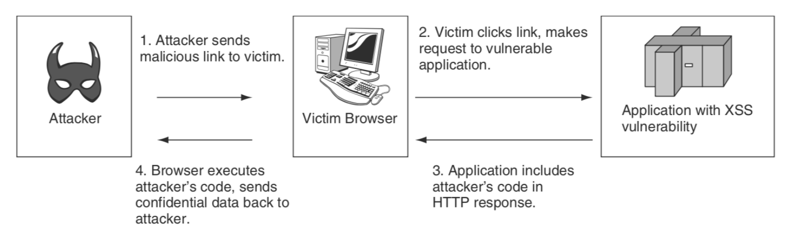
- Perform *input validation and output validation/encoding* on the server regardless of the checks that are performed on the client to prevent the following vulnerabilities:
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - HTTP Response Splitting
  - Open Redirects
  - ...

Input and Output Validation for the Web

- Cross-Site Scripting (XSS) occurs when the following steps take place:
  - Data enters a Web application through an untrusted source, most frequently an HTTP request or a data store such as a database.
  - The application includes the data in dynamic content that is sent to a Web user without properly validating it for malicious content.
- XSS vulnerability involves malicious content from one source being injected into legitimate content from another.
- A Web application has the responsibility to protect its clients from XSS attacks by taking preventive measures using output validation/encoding as well as input validation.

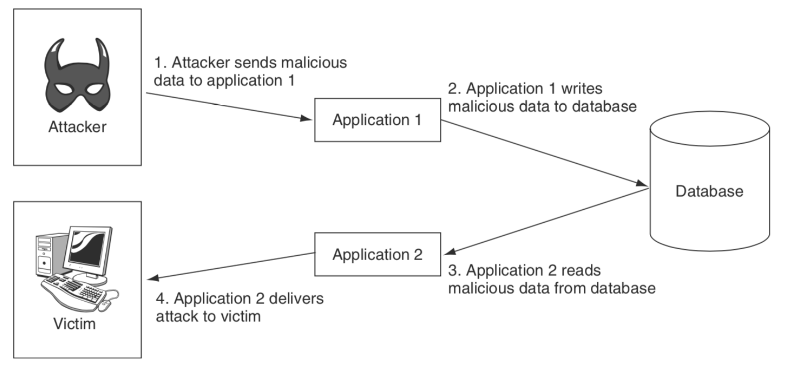
Input and Output Validation for the Web

- Reflected Cross-Site Scripting (XSS) occurs when the Web application reflects the attack back to the client.


Attacker
1. Attacker sends malicious link to victim.
Victim Browser
2. Victim clicks link, makes request to vulnerable application.
Application with XSS vulnerability
3. Application includes attacker's code in HTTP response.
4. Browser executes attacker's code, sends confidential data back to attacker.

Input and Output Validation for the Web

- A data store such as a database can be a conduit for XSS attacks. This form of vulnerability is called *stored cross-site scripting*.
- A stored cross-site scripting can involve/affect multiple applications.


Attacker
1. Attacker sends malicious data to application 1
Application 1
2. Application 1 writes malicious data to database
Database
3. Application 2 reads malicious data from database
Application 2
4. Application 2 delivers attack to victim
Victim

Input and Output Validation for the Web

- HTTP Response Splitting
  - HTTP response splitting vulnerability allows an attacker to write data into an HTTP header.
  - The attacker will have the ability to control the remaining headers and the body of the current response or even craft an entirely new HTTP response.
  - Output validation and encoding as well as input validation are effective ways to mitigate this vulnerability.

Input and Output Validation for the Web

- Open Redirects
  - Attackers lure potential victims (in a phishing attack) to click on a link that actually takes victims to a legitimate Web site but then immediately forwards them on to another site controlled by the attacker that harvests the sensitive information.
  - Phishing attacks are the result of a vulnerability in the application where pages used in a redirect are not validated for legitimacy.
  - Using whitelisting and a level of indirection allows the programmer to control where the redirect goes, as shown in the code below:

```
String nextPage = pageLookup.get(request.getParameter("next"));
response.sendRedirect(nextPage);
```

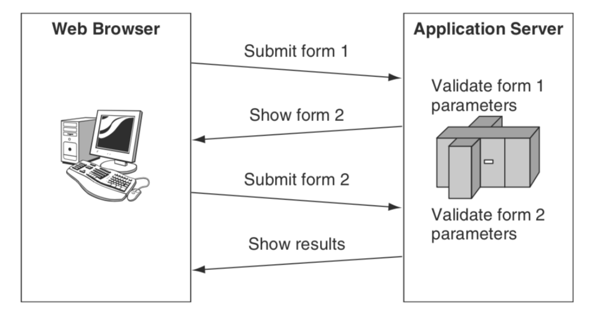
HTTP Considerations

- HTTP protocol contains a number of traps that can cause security problems.
- Use POST, not GET request method
  - GET requests expose (sensitive) information in the query string portion of the URL.
  - POST is preferable to GET for form submission because request parameters appear in the body of the request instead of being appended to the query string.
  - Good practice – Send GET requests to an error page as shown below.
  - In addition to protecting sensitive data, disallowing the use of the GET method can also make it harder to exploit reflected cross-site scripting vulnerabilities.

```
public void doGet( HttpServletRequest request,
                   HttpServletResponse response )
                   throws ServletException, IOException
{
   response.sendRedirect(ERROR_PAGE);
}
```

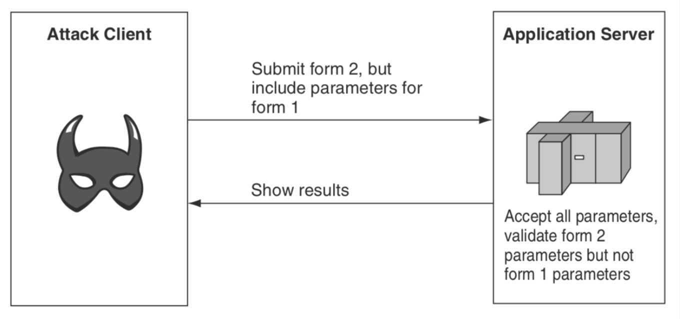
HTTP Considerations

- Request Ordering – In poorly designed systems, attackers can use out-of-order requests to bypass application logic.
- Do not depend on requests arriving in the order you expect.
- An expected request sequence for a multi-form submission request goes like below:



HTTP Considerations

- The attacker submits the second form but includes the parameters for the first form in order to bypass the validation for the first form.


Attack Client
Application Server
Submit form 2, but include parameters for form 1
Show results
Accept all parameters, validate form 2 parameters but not form 1 parameters

HTTP Considerations

- Error Handling in Web Applications
  - Do not use HTTP error codes to communicate information about errors. Requests that result in an error should still return `200 OK`.
  - Use a single generic error page for all unexpected events, including HTTP errors and unhandled exceptions. Attackers can use the information on HTTP error codes and exceptions to probe your application.
  - *Fail Securely* by not leaking information such as stack traces or other system data to a Web user.
  - Use a very broad catch-all exception handler at the top-level.
  - Do not generate error messages that give an attacker clues about how the system works or where it might be vulnerable.
  - Use carefully crafted error messages to avoid leaking important information such as identity of users, network details, or specifics about the application or server environment.

HTTP Considerations

- Request Provenance Validation
  - A Web application that uses session cookies must take special precautions to ensure that an attacker can't trick users into submitting bogus requests.
  - If not careful, this could lead to a cross-site request forgery (CSRF) attack.
  - CSRF is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.
  - 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.

Maintaining Session State

- HTTP is a stateless protocol where each request and response pair is independent of other web interactions.
- A *web session* is a sequence of network HTTP request and response transactions associated to the same user.
- Because HTTP is stateless, managing a web session in an application requires passing a *session identifier or ID or token* back and forth to associate a user's previous requests with his/her next.
- Today's most applications handle a web session using session identifiers stored in HTTP cookies.
- A session id is a "name=value" pair stored in HTTP cookies.
- The disclosure, capture, prediction, brute force, or fixation of the session ID will lead to session hijacking (or sidejacking) attacks, where an attacker is able to fully impersonate a victim user in the web application.

Maintaining Session State

- Use of good session management facilities in an application is critical to the security of a Web application.
- A good session identifier must meet the following properties:
  - The name used by the session ID should not be extremely descriptive nor offer unnecessary details about the purpose and meaning of the ID.
  - The session ID must be long enough to prevent brute force attacks. The session ID length must be at least 128 bits.
  - Generate session ID using a cryptographically secure random number generator with at least 64 bits of entropy.

Maintaining Session State

- A good session identifier must meet the following properties:
  - The session ID content (value) must be meaningless to prevent information disclosure attacks.
  - Make use of security features offered in the form of attributes that can be associated with a cookie used for session ID: *Secure*, *HttpOnly*, *SameSite*, *Domain*, *Path*, *Expire*, and *Max-Age*.
  - Invalidate/terminate a session when “session idle timeout” or “max session lifetime” limits exceed for a session.
  - Always generate a new session ID when a user authenticates to prevents session fixation attacks.

References

- Brian Chess and Jacob West, “Secure Programming with Static Analysis”, Chapter on “Web Applications”, Addison Wesley, 2007.