Web Security

1 HTTP, HTML, and JavaScript

- HTTP Request

  - Request line, such as GET /images/logo.gif HTTP/1.1, which requests a resource called /images/logo.gif from server
  - Headers, such as Accept-Language: en
  - An empty line
  - An optional message body

- Request methods

  - GET Request: attach the data in the URL
  - POST Request: Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request.

- HTML: An Example

```
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

- JavaScript in HTML: A "Hello World" Example

```
<script type="text/javascript">
   document.write('Hello World!');
</script>
```

- What can JavaScript do?

  - JavaScript gives HTML designers a programming tool.
  - JavaScript can put dynamic text into an HTML page: A JavaScript statement like the following, which write a variable text into an HTML page:

```
document.write("<h1>" + name + "</h1>")
```

  - JavaScript can react to events: A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.

```
<a href="xyz.html" onclick="alert('I have been clicked!')">
```

- JavaScript can read and write HTML elements: A JavaScript can read and change the content of an HTML element.

  ```
  var doc = document.childNodes[0];
  ```

- JavaScript can be used to validate data: A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing.
- JavaScript can be used to detect the visitor's browser: A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- JavaScript can access cookies: A JavaScript can be used to store and retrieve information on the visitor's computer

  ```
  var cookie = document.cookie;
  ```

- JavaScript can interact with the server (e.g. using Ajax).

- Ajax: Ajax (shorthand for asynchronous JavaScript + XML) is a group of interrelated web development techniques used on the client-side to create interactive web applications. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. The use of Ajax techniques has led to an increase in interactive or dynamic interfaces on web pages and better quality of Web services due to the asynchronous mode. Data is usually retrieved using the XMLHttpRequest object. Despite the name, the use of JavaScript and XML is not actually required, nor do the requests need to be asynchronous.

```
<html>
<body>

<script type="text/javascript">
var xmlhttp;
function ajaxFunction()
{
  xmlhttp=new XMLHttpRequest();
  xmlhttp.onreadystatechange=readyFunction()
  xmlhttp.open("GET","time.asp",true);
  xmlhttp.send(null);
}
function readyFunction()
{
    if(xmlhttp.readyState==4)
    {
      document.myForm.time.value=xmlhttp.responseText;
    }
}
</script>

<form name="myForm">
Name: <input type="text" name="username" onkeyup="ajaxFunction();" />
Time: <input type="text" name="time" />
</form>

</body>
</html>
```

- Secure Web Access

  - Authentication

  - Access Control: what is the basis of access control?

  - Discussion

- Difference between Web Access Control and OS Access Control

  - OS is stateful. After an user is authenticated, it is remembered until the user logs out. The OS keeps the state: the autenticated user gets a process with his/her privileges; this process keeps the fact that the user is authenticated. Other users cannot hijack this process.

  - Web server is stateless. When a user is authenitcated, he/she may send several other requests. The entire duration is called a session. Since web server is stateless, it does not remember anything about this session. Namely, when the user sends a request, the server does not know whether they are from the same session (hence, from the same user). To put in another perspective, because of the lack of session concept at web server, each web request has to be authenticated; otherwise, attackers can hijack a session.

- Session ID

  - Web applications have to remember sessions. For example, when a host needs to customize the content of a website for a user, the web application must be written to track the user's progress from page to page.

  - How to know two requests are from the same sessions, hence do not need seperate authenication?

  - One common solution is to use session ID.

  - Where is session ID stored:
    * Using cookies to record states: will be included automatically in the HTTP request.
    * Using hidden variables in forms: will be sent automatically when the form is submitted.
    * Using URL encoded parameters: has to attach the session ID in the HTTP request. Here is an example:
      ```
      /index.php?session_id=some_unique_session_code.
      ```

- Cookies and Session ID

  - A cookie (also tracking cookie, browser cookie, and HTTP cookie) is a small piece of text stored on a user's computer by a web browser. A cookie consists of one or more name-value pairs containing bits of information such as user preferences, shopping cart contents, the identifier for a server-based session, or other data used by websites.

  - It is sent as an HTTP header by a web server to a web browser and then sent back unchanged by the browser each time it accesses that server. A cookie can be used for authenticating, session tracking (state maintenance), and remembering specific information about users, such as site preferences or the contents of their electronic shopping carts. The term "cookie" is derived from "magic cookie", a well-known concept in UNIX computing which inspired both the idea and the name of browser cookies. Some alternatives to cookies exist; each has its own uses, advantages, and drawbacks.

– Being simple pieces of text, cookies are not executable. They are neither spyware or viruses, although cookies from certain sites are detected by many anti-spyware products because they can allow users to be tracked when they visit various sites.

- Server-Side Access Control

  – Subject: authentication with session id.

  – Objects: files, data, etc.

  – Policy: can be DAC (e.g. Facebook), MAC or others.

- Browser-Side Access Control

  – The server-side access control relies on the integrity of the browser-side access control.

  – The integrity of user's behavior should also be preserved: i.e., malicious users cannot affect/change a legitimate users's behavior.

  – What are the essential requirements?

    * Session id: cannot be stolen.

    * Program/Data: cannot be modified by unauthorized users.

  – Policy: Same Origin Policy (SOP)

2 Access Control on JavaScript

In class, we will give students 15 minutes to discuss how they want to restrict JavaScript's access. Basically, students are asked to implement an access control system for web browser to protect users against malicious JavaScript code. Decision has to be justified, and balance between usability and security needs to maintained.

- Where do we start? We need to understand what are *subject* and *object* first, then we can talk about the access control.

- Subject (fine granularity): The origin of JavaScript code.

- Objects (fine graularity): DOM, Cookies, Operating System Resources (files, processes, devices, networks, keyboard, mouse, memory, etc).

- Policies:

  – Same Origin Policy (SOP).

  – Directly accessing of the OS resources are prohibited.

  – Browsers provide APIs to JavaScript, so OS resources can be indirectly accessed. For example, JavaScript can send out messages using `Ajax` APIs, access DOM objects and cookies using DOM APIs, and so on.

3   Cross-Site Scripting (XSS) Attack

- Objective of XSS:

  - Attacker injects malicious JavaScript code to the target web site X.
  - When other users browse the infected pages from X, the browser believes that the JavaScript is from X.
  - The Same Origin Policy allows the malicious JavaScript to access cookies of X, which can send legitimate HTTP requests to X onbehalf of the users, without the users' concent.

- Samy worms (see the narrative from Samy at `http://namb.la/popular/`).

  - Myspace.com: Samy add JavaScript code in his profile; whoever browses the profile will get infected.
  - The worm added Samy to the victim's friend list, and then further propogate the worms to those who view their profiles.
  - Samy become a friend of one million users in less than 20 hours.

- Difficult to filtering out JavaScript code: Myspace did have filters that tried to filter out JavaScript code, but the Samy worms had overcame those obstacles (technical details are described in `http://namb.la/popular/tech.html`):

  - Myspace blocks a lot of tags, including `<script>`, `<body>`, and `onClick`, `onAnything`. Therefore, the Samy worm could not use these tags. However, some browsers (IE, some versions of Safari, others) allow javascript within CSS tags (i.e. without using these tags). For example, the following tag include a JavaScript code without using those forbidden tags:

    ```
    <div style="background:url('javascript:alert(1)')">
    ```

  - Myspace strips out the word `javascript` from anywhere. Fortunately, some browsers will actually interpret `java<NEWLINE>script` as `javascript`:

    ```
    <div style="background:url('java
                script:alert(1)')">
    ```

  - The Samy worm needs to use AJAX in order for the actual client to make HTTP GETs and POSTs to pages. However, myspace strips out the word `"onreadystatechange"` which is necessary for XML-HTTP requests. One can use an `eval` to evade this. Namely, instead of writing the code in Choice 1, we can use Choice 2 (The Samy worm uses several tricks like this):

    ```
    Choice 1: xmlhttp.onreadystatechange = callback;
    Choice 2: eval('xmlhttp.onread' + 'ystatechange = callback');
    ```

  - Myspace also filters out several other things, but they were successfully circumvented by the Samy worm.

- Potential Damage

  - Sending unauthorized requests on behalf of the victims.
  - Web defacing: the malicous JavaScript code can access and modify the DOM objects within the page. For example, it can replace a picture in the web page with a different picture.

- Countermeasures

    - Do a better filtering (proven difficult).
    - *Noscript* region: Do not allow JavaScript to appear in certain region of a web page.

4 Cross-Site Request Forgery (CSRF) Attack

- CSRF Attack

    - Web application tasks are usually linked to specific urls (Example: `http://site/buy_stocks?buy=200&stock=yahoo`) allowing specific actions to be performed when requested.
    - If a user is logged into the site and an attacker tricks their browser into making a request to one of these task urls, then the task is performed and logged as the logged in user. The tricks can be placed on a web page from the attacker; all the attacker needs to do is to trick the user to visit their attacking web page while being logged into the targeted site.
    - When the request is made by the user (whether the user is tricked or not), the cookie will be attached to the request automatically by browsers.
    - For web applications using HTTP GET: attacker can use image tag `<img>` to cause the victim's browser to send out a HTTP GET request (when the victim visits the attacker's web page, the HTTP GET request will be initiated by the image tag. Here is an example:

        ```
        <img src="http://site/buy_stocks?buy=200&stock=yahoo">
        ```

    - For web applications using HTTP POST: sending data to such applications is not as easy as sending data to a GET-based applications, because we cannot append the data to the end of URL for POST-based applications. However, with the help of JavaScript, attackers can send the data. The basic idea is for the attacker to craft a web form on his/her site (using JavaScript), and then use JavaScript to automatically submit the form to the target site.
      We cannot use AJAX here, because AJAX can only talk back to the source of the web page (SOP policy).

- Difference between CSRF and XSS

    - CSRF does not need to run JavaScript code (for GET only); XSS does.
    - Using JavaScript code:
        * CSRF: the code runs directly from the attacker's web page.
        * XSS: the code has to be injected to the target web site's page.
    - Server-side input validation:
        * It does not prevent CSRF, because the attacking contents are not on the target web site.
        * It can prevent XSS to certain degree, if the malicious JavaScript code can be filtered out.

- Countermeasures

    - Because the JavaScript code used (if used) by CSRF does not come from the target web site, the malicious JavaScript cannot see the cookies from the target web site.

– We can require that all the HTTP request (both GET and POST) to also include something from the cookie (such as the session ID) in the attached parameters, in addition to the cookies that are already attached automatically by the browser. JavaScript code from the target web site can get the secret from the cookie, but the JavaScript code from the malicious web site cannot access the cookies.

5   Fundamental Problems of XSS and CSRF

What is the fundamental problem of XSS and CSRF? Let us evaluate these problems from the access control perspective. Is there anything wrong with the access control model currently used by web browser (i.e. the SOP model)? If not, can we pinpoint what has gone wrong from the design perspective?

Let us review the principles of access control formuated by Saltzer and Schroeder in their classical paper titled *The Protection of Information in Computer Systems* [1]. We have covered these principles in our access control lectures. Here we will evaluate an access control design using these principles:

- Economy of mechanism

- Fail-safe defaults

- Complete mediation

- Open design

- Separation of privilege

- Least privilege

- Least common mechanism

- Least common mechanism

- Psychological acceptability

We will then have a 15-minute in-class discussion on the following topic: which of the above principles are violated by the design of SOP? what should we do if we want to follow these principles?

References

[1] J. H. Saltzer and M. D. Schroeder. *The Protection of Information in Computer Systems*. In Proceedings of the IEEE, Vol. 63, No. 9. (1975), pp. 1278-1308.