Web basics: HTTP cookies

Myrto Arapinis
School of Informatics
University of Edinburgh

March 27, 2019

Web access control - 3 key aspects

- Authentication - username and passwords
- Session management - link sequences of requests of authenticated users
- Authorisation - check and enforce permissions of authenticated users

Session management

- Goal - the server should not require a user to re-authenticate at each HTTP(s) request
- Problem - HTTP is stateless
- Solution -
  - User logs in once
  - The server generated session identifier and sends it to the client (browser)
    temporary token that identifying an authenticated user
  - The client returns the session identifier in subsequent requests
  - 2 main approches: hidden fields and cookies

Hidden fields (slide from Web basics lecture)

The principle

Include an HTML form with a hidden field containing a session ID in all the HTML pages sent to the client. This hidden field will be returned back to the server in the request.

Example: the web server can send a hidden HTML form field along with a unique session ID as follows:

```
<input type="hidden" name="sessionid" value="12345">
```

When the form is submitted, the specified name and value are automatically included in the POST data.

Cookies (slide from Web basics lecture)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments

Cookies (slide from Web basics lecture)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie

Cookies (slide from Web basics lecture)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie
- Cookies are only sent back by the browser to their originating host and not any other hosts. Domain and path specify which server (and path) to return the cookie

Cookies (slide from Web basics lecture)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie
- Cookies are only sent back by the browser to their originating host and not any other hosts. Domain and path specify which server (and path) to return the cookie
- A server can set the cookie's value to uniquely identify a client. Hence, cookies are commonly used for session and user management

Cookies (slide from Web basics lecture)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie
- Cookies are only sent back by the browser to their originating host and not any other hosts. Domain and path specify which server (and path) to return the cookie
- A server can set the cookie's value to uniquely identify a client. Hence, cookies are commonly used for session and user management
- Cookies can be used to hold personalized information, or to help in on-line sales/service (e.g. shopping cart), or tracking popular links.

Web security: session hijacking

Session hijacking

Wikipedia

Session hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system

Sessions could be compromised (hijacked) in different ways; the most common are:

Session hijacking

Wikipedia

Session hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system

Sessions could be compromised (hijacked) in different ways; the most common are:

- Session token theft vulnerabilities:

Session hijacking

Wikipedia

Session hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system

Sessions could be compromised (hijacked) in different ways; the most common are:

- Session token theft vulnerabilities:
  - Predictable session tokens:

Session hijacking

Wikipedia

Session hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system

Sessions could be compromised (hijacked) in different ways; the most common are:

- Session token theft vulnerabilities:
  - Predictable session tokens:
    $\Longrightarrow$ cookies should be unpredictable

Session hijacking

Wikipedia

Session hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system

Sessions could be compromised (hijacked) in different ways; the most common are:

- Session token theft vulnerabilities:
  - Predictable session tokens:
    $\Longrightarrow$ cookies should be unpredictable
  - HTTPS/HTTP: site has mixed HTTPS/HTTP pages, and token is sent over HTTP

Session hijacking

Wikipedia

Session hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system

Sessions could be compromised (hijacked) in different ways; the most common are:

- Session token theft vulnerabilities:
  - Predictable session tokens:
    $\Longrightarrow$ cookies should be unpredictable
  - HTTPS/HTTP: site has mixed HTTPS/HTTP pages, and token is sent over HTTP
    $\Longrightarrow$ set the `secure` attribute for session tokens (cookies)
    $\Longrightarrow$ when elevating user from anonymous to logged-in, always issue a new session token

Session hijacking

Wikipedia

Session hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system

Sessions could be compromised (hijacked) in different ways; the most common are:

- Session token theft vulnerabilities:
  - Predictable session tokens:
    $\Longrightarrow$ cookies should be unpredictable
  - HTTPS/HTTP: site has mixed HTTPS/HTTP pages, and token is sent over HTTP
    $\Longrightarrow$ set the `secure` attribute for session tokens (cookies)
    $\Longrightarrow$ when elevating user from anonymous to logged-in, always issue a new session token
  - Cross-site scripting (XSS) vulnerabilities

Session hijacking

Wikipedia

Session hijacking is the exploitation of a valid computer session to gain unauthorized access to information or services in a computer system

Sessions could be compromised (hijacked) in different ways; the most common are:

- Session token theft vulnerabilities:
  - Predictable session tokens:
    $\Longrightarrow$ cookies should be unpredictable
  - HTTPS/HTTP: site has mixed HTTPS/HTTP pages, and token is sent over HTTP
    $\Longrightarrow$ set the `secure` attribute for session tokens (cookies)
    $\Longrightarrow$ when elevating user from anonymous to logged-in, always issue a new session token
  - Cross-site scripting (XSS) vulnerabilities
- Cross-site request forgery (CSRF) vulnerabilities

Cross-site request forgery (CSRF)

CSRF

OWASP

CSRF forces a user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

CSRF

OWASP

CSRF forces a user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

Target: user who has an account on vulnerable server

CSRF

OWASP

CSRF forces a user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

Target: user who has an account on vulnerable server

Main steps of attack:

1. build an exploit URL
2. trick the victim into making a request to the vulnerable server as if intentional

CSRF

OWASP

CSRF forces a user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

Target: user who has an account on vulnerable server

Main steps of attack:

1. build an exploit URL
2. trick the victim into making a request to the vulnerable server as if intentional

Attacker tools:

1. ability to get the user to "click exploit link"
2. ability to have the victim visit attacker's server while logged-in to vulnerable server

CSRF

OWASP

CSRF forces a user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

Target: user who has an account on vulnerable server

Main steps of attack:

1. build an exploit URL
2. trick the victim into making a request to the vulnerable server as if intentional

Attacker tools:

1. ability to get the user to "click exploit link"
2. ability to have the victim visit attacker's server while logged-in to vulnerable server

Keys ingredient: requests to vulnerable server have predictable structure

CSRF: a simple example

Alice wishes to transfer $100 to Bob using the bank.com web application. This money transfer operation reduces to a request like:

```
GET http://bank.com/transfer.do?acct=BOB&amount=100
HTTP/1.1
```

CSRF: a simple example

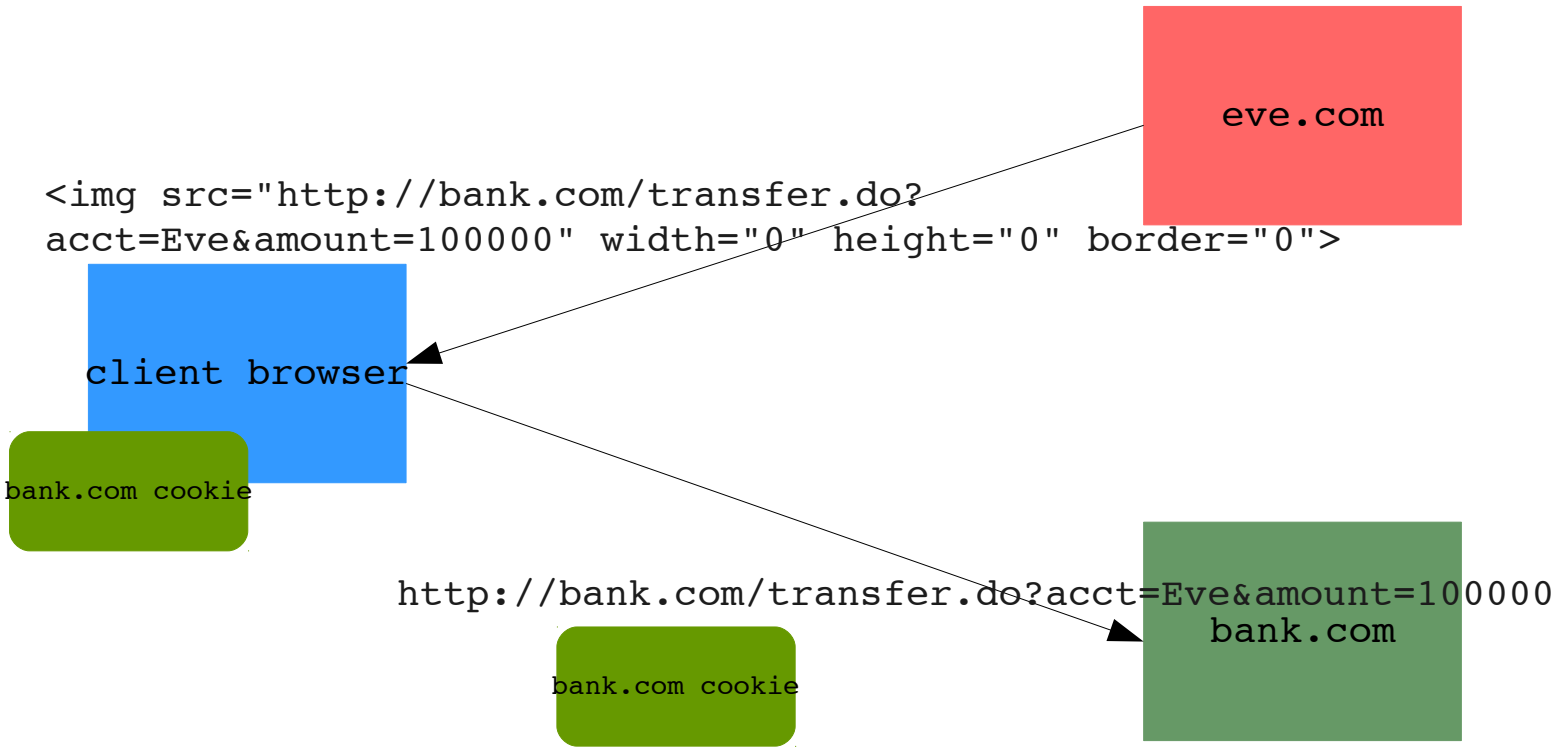
Alice wishes to transfer $100 to Bob using the bank.com web application. This money transfer operation reduces to a request like:

```
GET http://bank.com/transfer.do?acct=BOB&amount=100
HTTP/1.1
```

The `bank.com` server is vulnerable to CSRF: the attacker can generate a valid malicious request for Alice to execute!!
The attack comprises the following steps:

1. Eve crafts the following URL
   `http://bank.com/transfer.do?acct=Eve&amount=100000`
2. When Alice visits Eve's website she tricks Alice's browser into accessing this URL

CSRF: a simple example



CSRF defenses

- **Check the referrer** header in the client's HTTP request can prevent CSRF attacks. Ensuring that the HTTP request has come from the original site means that attacks from other sites will not function

CSRF defenses

- **Check the referrer** header in the client's HTTP request can prevent CSRF attacks. Ensuring that the HTTP request has come from the original site means that attacks from other sites will not function

- **Include a secret in every link/form!**
  - Can use a hidden form field, custom HTTP header, or encode it directly in the URL
  - Must be unpredictable!
  - Can be same value as session token (cookie)
  - Ruby on Rails embeds secrets in every link automatically
  - To avoid any replay attack should be different in each server response

CSRF defenses

- **Check the referrer** header in the client's HTTP request can prevent CSRF attacks. Ensuring that the HTTP request has come from the original site means that attacks from other sites will not function

- **Include a secret in every link/form!**
  - Can use a hidden form field, custom HTTP header, or encode it directly in the URL
  - Must be unpredictable!
  - Can be same value as session token (cookie)
  - Ruby on Rails embeds secrets in every link automatically
  - To avoid any replay attack should be different in each server response

- **Set the** `SameSite` **cookie attribute** - prevents coloes from being sent in cross-site requests. But this is a very recent standard and might not be supported by all browsers.

Twitter SMS account hijacking (Nov. 2013)