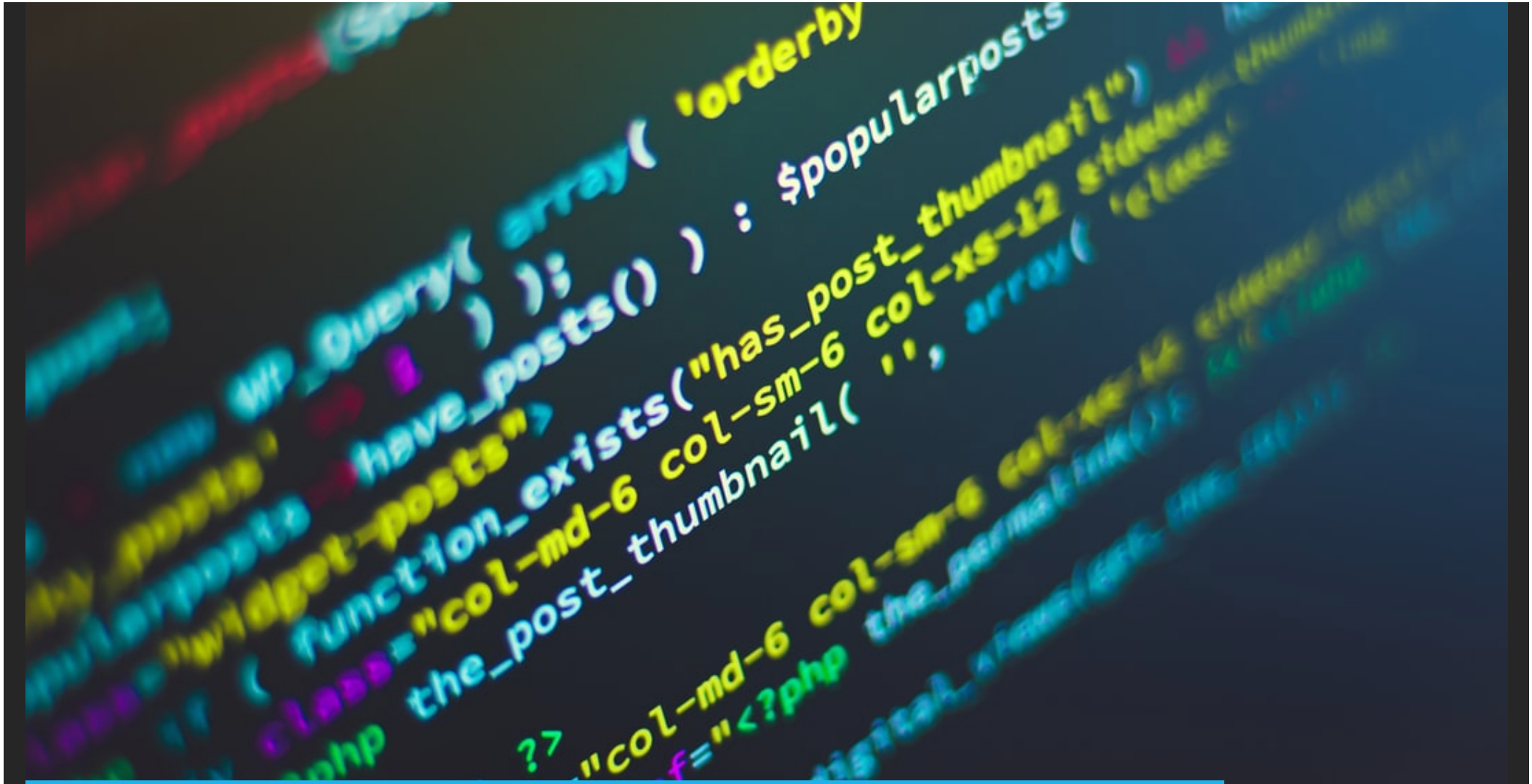
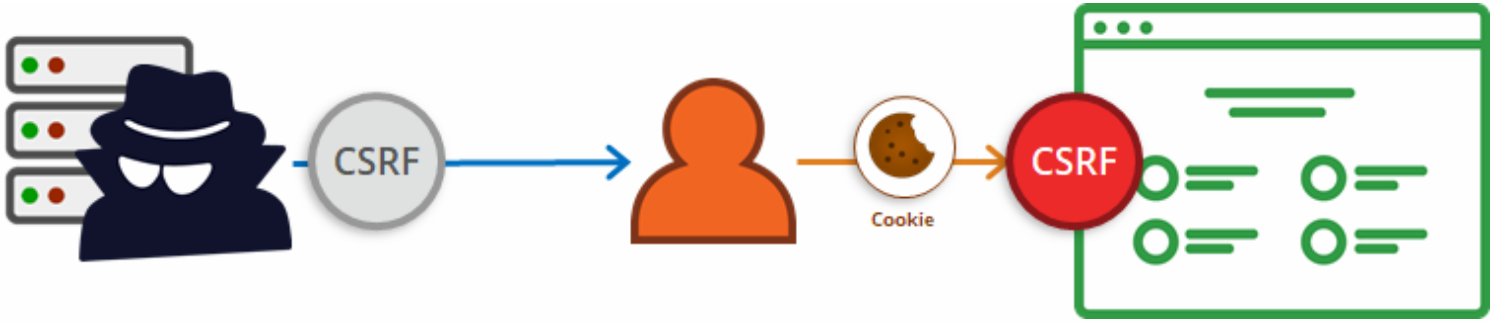


Bypassing Anti-CSRF with Burp Suite Session Handling

I have decided to re-write this intro, because I was recently informed that this attack narrative was confusing at best. This blog post will be talking heavily about the Anti-CSRF protections that web applications use, but the steps included are not meant to be used for a CSRF attack. This is meant to be used when testing input points for possible injection vulnerabilities. For example, a tester might need to test the login page for potential SQL injection, but automated tools might have trouble due to the Anti-CSRF protections.

So, what is Cross-Site Request Forgery (CSRF)? CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.



There are several methods being used to protect web application attacks, the most common method is using Anti-CSRF Tokens. During web application penetration tests, CSRF can often be used correctly and might deter security testers from properly testing injection points. Popular tools such as SQLmap have built in CSRF bypass methods, but recently I found an issue where it wasn't reading the CSRF token from within the body of the response. Therefore, with the help of others I was able to learn how to use Burp Suite to properly bypass these Anti-CSRF protections.

To begin I am using the [Vulnerable Web Application Test Environment]() that was blogged about in 2018. Specifically I will be using the Damn Vulnerable Web App, or more commonly referred to as DVWA , to demonstrate this anti-CSRF bypass. As you can see from the screenshot below, this is our baseline request when submitting a password reset when authenticated to the DVWA.

Request to http://dvwa.local:80 [127.0.0.1]

Forward | Drop | Intercept is on | Action | Comment this item

Raw | Params | Headers | Hex

```
GET /vulnerabilities/csrf/?password_current=test&password_new=test&password_conf=test&Change=Change&user_token=641e45f717a24e1a9b26d8a05a6fbb40 HTTP/1.1
Host: dvwa.local
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://dvwa.local/vulnerabilities/csrf/
Cookie: _ga=GA1.2.86656614.1558286345; _gid=GA1.2.279877446.1558286345; PHPSESSID=q7td8o0ur3o26r4v55j66odei3; security=impossible
Connection: close
Upgrade-Insecure-Requests: 1
```

In the request above you can note from one of the cookies that the security is currently set to Impossible. Inside the DVWA you can change the difficult rating in order to attempt different exploits against web applications. For this post I will be using either the Impossible or High. Which both have Anti-CSRF tokens generated. The screenshot below is the response from the web server when submitting the password reset. Unfortunately, at the impossible security level you are required to submit the current password as well. In order to see if we can get a successful response, we will lower the security rating to High so we will not need the previous password.

```
        <input type= hidden  name= user_token  value= a61003021800 2a3cd308ca3787ade
    </form>
    <pre>Passwords did not match or current password incorrect.</pre>
>
```

In the screenshot below you will see that SQLmap is being used to test a parameter. The `-r` option allows for the user to pass a Burp Request that has been saved to the file system without having to specify multiple flags. Also, it appears that SQLmap has recognized the `user_token` parameter and asks if we want to use CSRF bypass methods. Even though this

method could be used in the instance of DVWA, for this blog post I will not be using the SQLmap CSRF bypass method.

```
root@kali:~# sqlmap -r testRequest
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.3.4#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior
 liability and are not responsible for any misuse or damage caused by this

[*] starting @ 13:28:39 /2019-05-19/

[13:28:39] [INFO] parsing HTTP request from 'testRequest'
GET parameter 'user_token' appears to hold anti-CSRF token. Do you want sq
[13:28:50] [INFO] testing connection to the target URL
[13:28:50] [INFO] checking if the target is protected by some kind of WAF/
[13:28:50] [INFO] testing if the target URL content is stable
[13:28:51] [WARNING] target URL content is not stable (i.e. content differ
results, refer to user's manual paragraph 'Page comparison'
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit]
```

Not every web application penetration test will allow the engineer to have full insight into what his happening, but luckily for us the DVWA gives the option at looking at the source code. Which we can see from the screenshot below at the bottom the final line will `generateSessionToken()`. This will generate an Anti-CSRF token which will then be

passed in the response. The web server will then expect that exact session token upon the next request.

CSRF Source

```php
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $pass_curr = $_GET[ 'password_current' ];
    $pass_new  = $_GET[ 'password_new' ];
    $pass_conf = $_GET[ 'password_conf' ];

    // Sanitise current password input
    $pass_curr = stripslashes( $pass_curr );
    $pass_curr = mysql_real_escape_string( $pass_curr );
    $pass_curr = md5( $pass_curr );

    // Check that the current password is correct
    $data = $db->prepare( 'SELECT password FROM users WHERE user = (:user) AND password = (:password) LIMIT 1;' );
    $data->bindParam( ':user', dvwaCurrentUser(), PDO::PARAM_STR );
    $data->bindParam( ':password', $pass_curr, PDO::PARAM_STR );
    $data->execute();

    // Do both new passwords match and does the current password match the user?
    if( ( $pass_new == $pass_conf ) && ( $data->rowCount() == 1 ) ) {
        // It does!
        $pass_new = stripslashes( $pass_new );
        $pass_new = mysql_real_escape_string( $pass_new );
        $pass_new = md5( $pass_new );

        // Update database with new password
        $data = $db->prepare( 'UPDATE users SET password = (:password) WHERE user = (:user);' );
        $data->bindParam( ':password', $pass_new, PDO::PARAM_STR );
        $data->bindParam( ':user', dvwaCurrentUser(), PDO::PARAM_STR );
        $data->execute();

        // Feedback for the user
        echo "<pre>Password Changed.</pre>";
    }
    else {
        // Issue with passwords matching
```

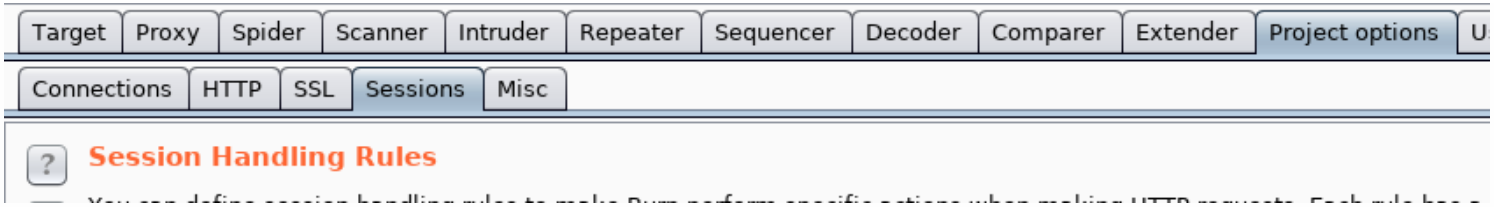
```
        echo "<pre>Passwords did not match or current password incorrect.</pre>";
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

Even though we do not need to know how the CSRF token is being generated it is a good idea to review the function that is being called. Or we might be able to just create our own unique CSRF token that would be valid.

```
function generateSessionToken() {  # Generate a brand new (CSRF) token
        if( isset( $_SESSION[ 'session_token' ] ) ) {
                destroySessionToken();
        }
        $_SESSION[ 'session_token' ] = md5( uniqid() );
}
```

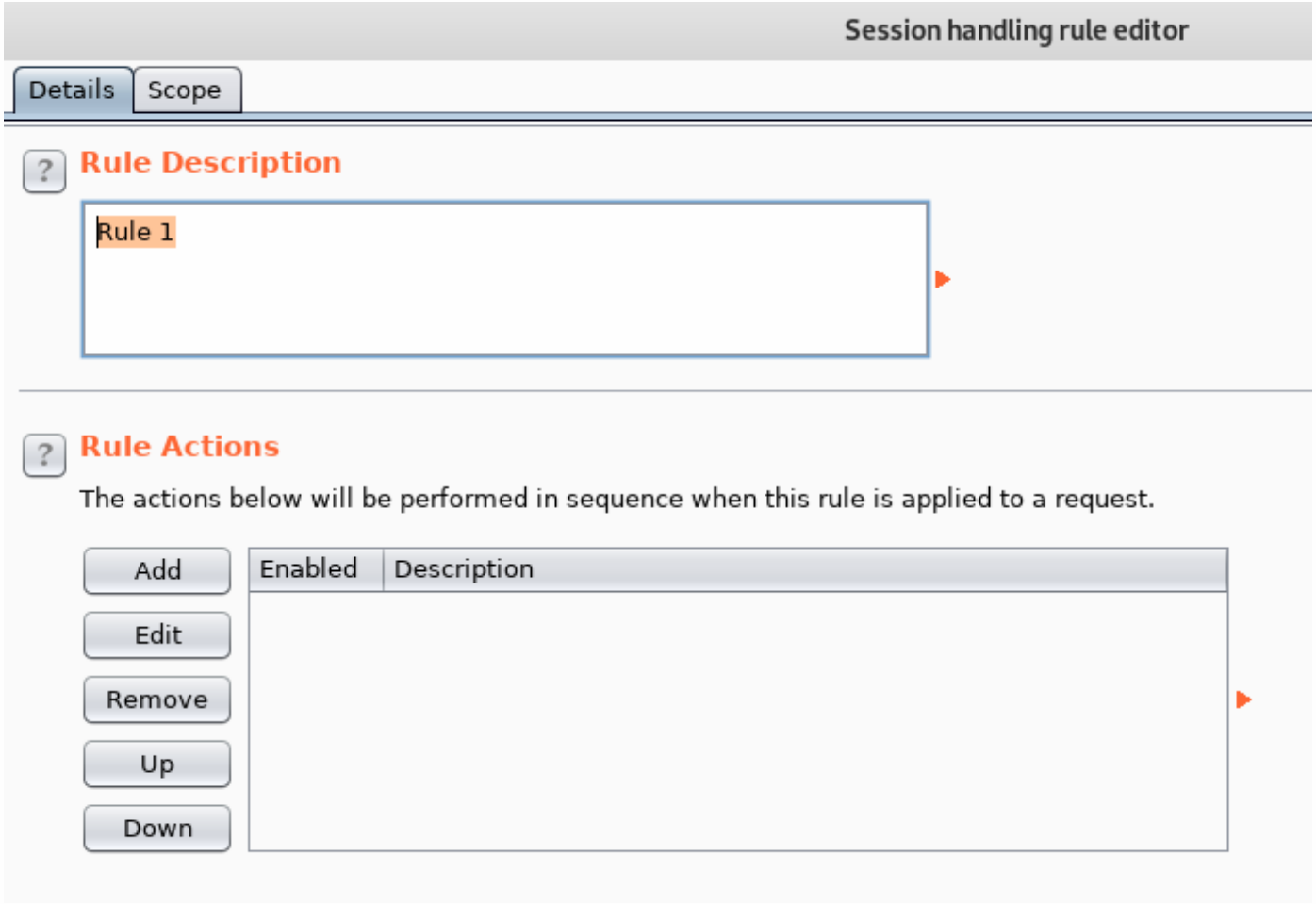
One of the more advanced features of Burp Suite is the ability to create macros that will trigger based on specific requests and responses. For the purpose of this blog I will be walking through the steps I use to bypass anti-CSRFtokens located within the body of a web application.

Additionally, it should be noted that these screenshots were taken while using the Burp Suite community edition which is free to use. Looking at the screenshot below we will begin by navigating to the `Project Options` tab and then into `Sessions`.



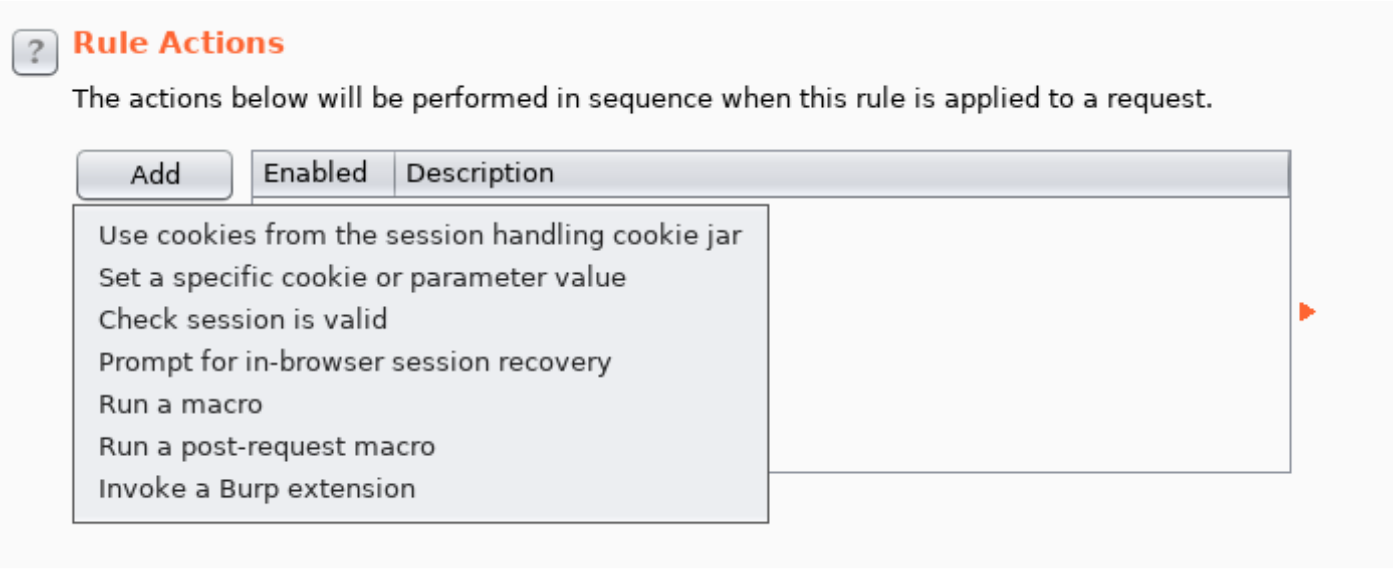
From here Burp Suite defines what Session Handling rules are, and gives some options for how the rules operate. Such as, adding, editing, or if you need a particular to execute before another. For this blog post we will be clicking Add to create a new session handling rule.

Burp Suite will open a new menu called the `Session handling rule editor`. This allows you to define the description and different rule actions that need to be taken. Once we have added a description, we can click the Add button to specify a new rule.

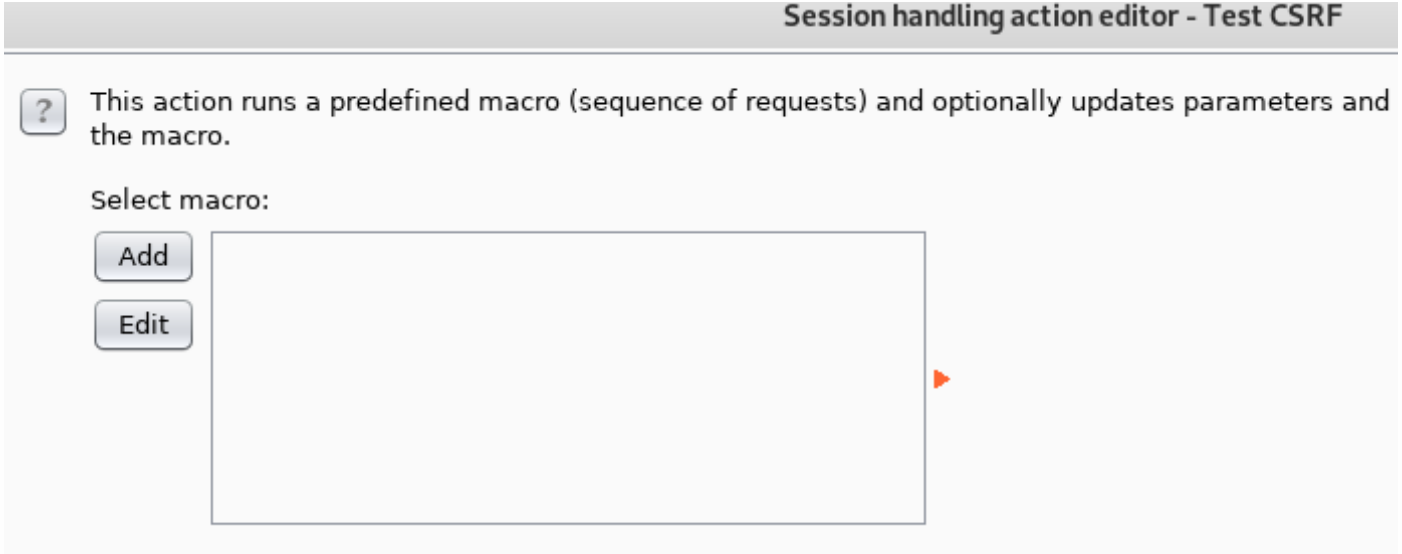


Burp Suite will drop down a menu with different rule actions that can be taken. For example, if you need to use a cookie that is already located in Burp's cookie jar you could utilize that here.

Or you could perform a check on every request to check and make sure the session is valid. Since these rule actions could be an entire blog post in itself, we will focus solely on `Run a macro` for this use case.



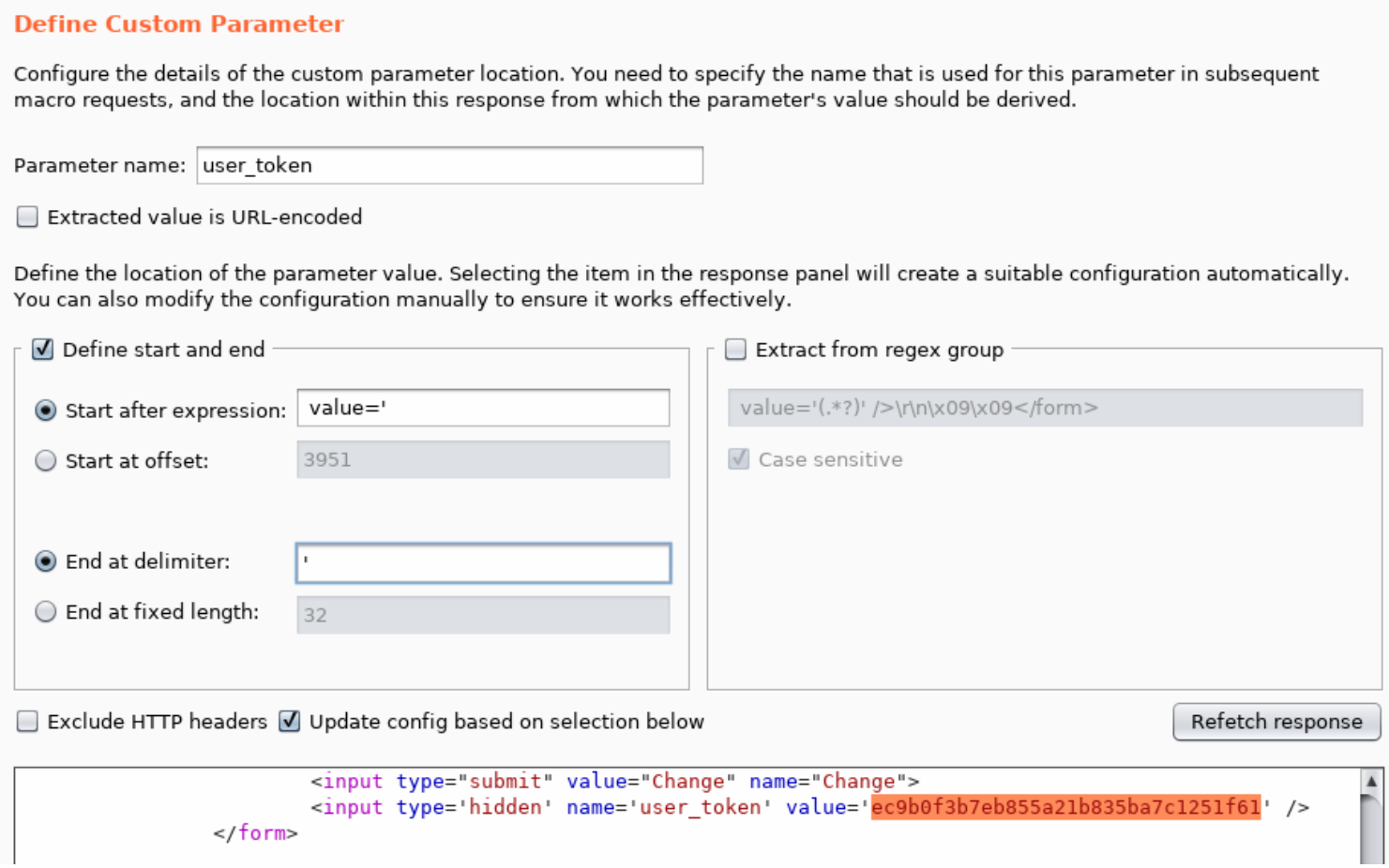
Again, Burp Suite opens another menu which is the `Session handling action editor` followed by the name we provided in the menu above. Here we can Add or Edit existing macros. If you wanted to define a macro that could be used across all web applications being tested you could simply choose that one here. However, since we are using a fresh copy of Burp Suite's community edition, we will select the `Add` button.



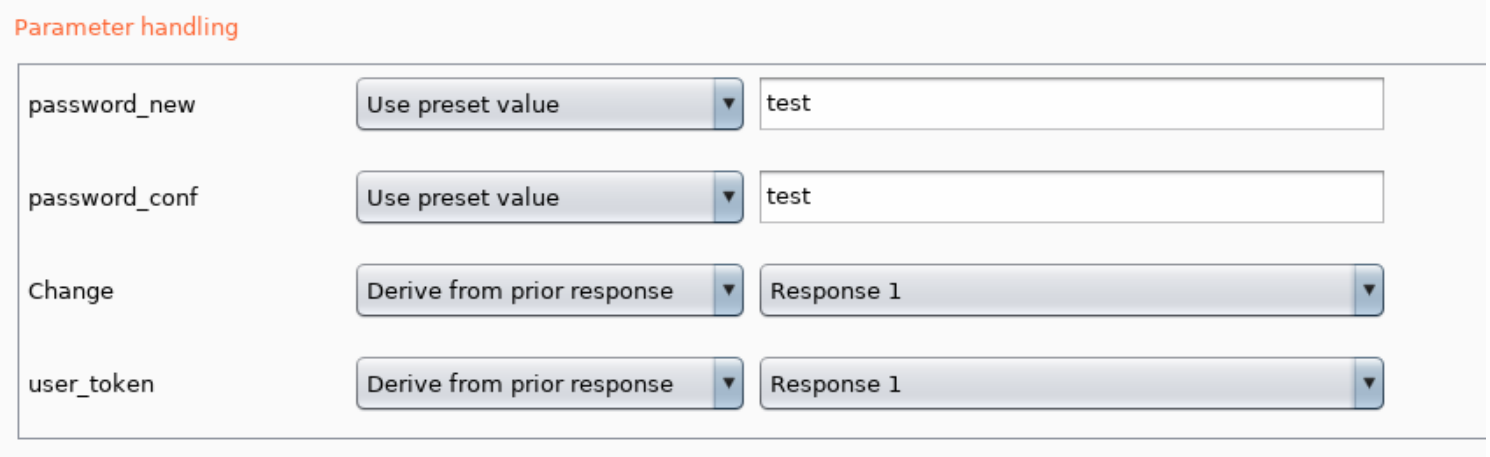
Burp Suite will open yet another two windows (For those keeping track, that is four windows not including the main application). The window that should be at the top layer at this point is the `Macro Recorder`. This menu gives you the option to pick out previous requests from the Proxy history tab and specify which requests will be triggering the macro. For the instance of this blog we will be sending a POST request to the DVWA application, and specifically targeting the `user_token`. We will select two subsequent requests for our Macro Recorder. The first is the initial request to the page which will request a `user_token` and then the second will be to send the password change request using the previous response. The second request will be removed, but for testing we will need to make sure we get a 200 response.

```
<br />
<input type="submit" value="Change" name="Change">
<input type='hidden' name='user_token' value='ec9b0f3b7eb855a21b835ba7c1251f61' />
```

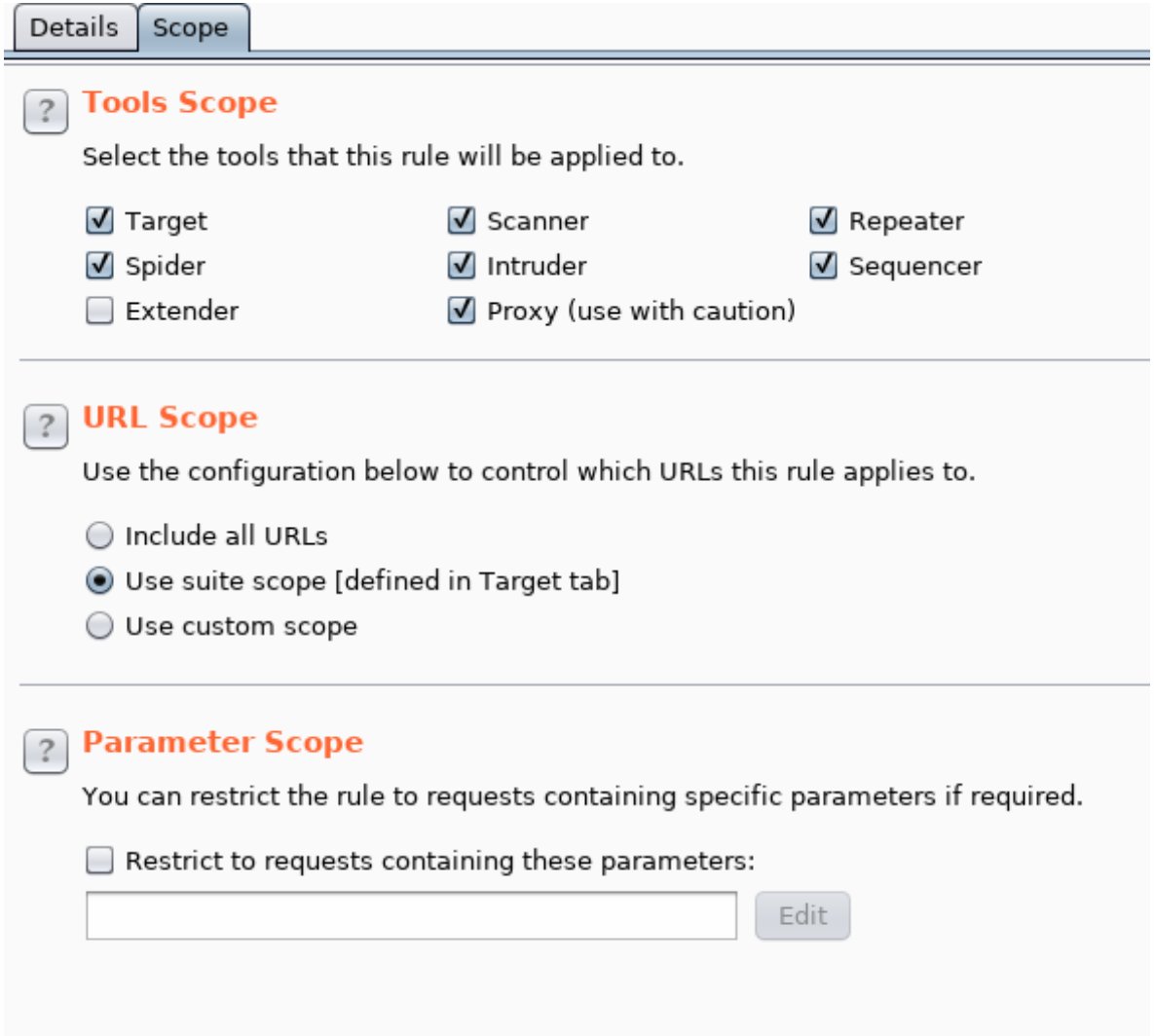
As you can see in the screenshot above the `user_token` is located within the body of the HTML in a hidden input type. Once we select the two requests from Burp Suite's Macro Recorder we will get another window that will allow us to fine tune the `Custom Parameters`. By highlighting the exact token value we need, Burp Suite will automatically generate a regex end delimiter. You can leave the pre-generated regex that Burp Suite defines. Or you could just enter a single `'` in the End at delimiter field. This will tell Burp where our token starts and stops for every response.



In the screenshot below you will see that the `Parameter handling` section will ask us what we want to do with the data that we have captured in our macro. For example, we will want to leave the preset values for the `password_new` and `password_conf` that way any data that is passed in will not be changed. The section that needs to be changed is the `user_token` and we want to use the data from `Response 1`. Ignore the section about `Change` because that value should never change. It is in fact a variable telling the backend what action to take.



In the screenshot below we are setting the `Scope` tab's parameters. This can be important by defining exactly which tools Burp Suite will trigger these macros for. For example, the `Proxy` option is not checked by default, but if we are planning to use Burp Suite in tandem with SQLMap, we will need to make sure that box is checked.



Burp Suite also provides a very helpful tool known as the `Sessions Tracer`. This will help track if the macros that were created above are being processed correctly and what the final request will look like. This is very helpful as it keeps you from having to spend a significant amount of time debugging your macros with the repeater or intercept tool.

To monitor or troubleshoot the behavior of your session handling rules,

Open sessions tracer

Once you open the sessions tracer you will can try changing another password in the DVWA web UI. Once you submit the requests should show in the session tracer. The request actively being view will be highlighted in orange shown in the top half of the screenshot below. The bottom half of the screenshot is the `Events` section. Which is where the actual debug events are. As you can see working form the top to the bottom Burp is triggering the `Test CSRF` session handling rule we have created. Once that rule triggers, Burp knows to run Macro #5 (It is #5 because I messed up 4 Macros before that). Highlighted in yellow is the Macro request which shows that on the next line will process the item with the full URL including parameters. Then in green is the new values that will be used for the parameter `user_token`. Finally, it will issue the macro request. The final line which is also highlighted yellow shows that the password change request we have left in for testing was issued. We can inspect the response from the POST request to see if the password change was submitted correctly using the updated `user_token`.

Requests handled

Time	Tool	URL
14:05:21 19 May 2019	Proxy	http://dvwa.local/vulnerabilities/csrf/index.php?password_new=test&password_conf=t
14:05:21 19 May 2019	Proxy	http://dvwa.local/vulnerabilities/csrf/index.php

Events

```
Applying rule: Test CSRF
  Running macro: Macro 5
    Processing macro item: http://dvwa.local/vulnerabilities/csrf/
      Issuing macro request
    Processing macro item: http://dvwa.local/vulnerabilities/csrf/?password_new=test&password_conf=test&Change=Change&user_token=ec9b0f3b7eb855a21b835ba7c1251f61
      Derived new value for parameter: user_token=ba3bcb13df046dd36cc24fd0b562ef2f
      Issuing macro request
Issued request
```

Macro description: Macro 5

Macro items:

#	Host	Method	URL	Status
1	http://dvwa.local	GET	/vulnerabilities/csrf/	200

Once our debugging is finished, we will remove the second request or Burp Suite will attempt to send two password resets every time we try to test the input field. Now we can move back to SQLmap and use the same request as before, and send it through the proxy for further testing. As you can see in the screenshot below SQLmap again recognizes the `user_token` parameter and asks if we want to use anti-CSRF bypass methods. However, since we are using Burp Suite to do the macro processing, we will simply say no.

```
root@kali:~# sqlmap -r testRequest --proxy http://127.0.0.1:8080
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.3.4#stable}
|_ -| . [(]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility
 liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 14:07:53 /2019-05-19/

[14:07:53] [INFO] parsing HTTP request from 'testRequest'
GET parameter 'user_token' appears to hold anti-CSRF token. Do you want sqlmap to automatically update it in further requests? [y/N] n
[14:07:55] [INFO] testing connection to the target URL
```

Finally, we can inspect our Burp Suite HTTP history to see that the requests are successfully being exchanged based on the prior request and response.

```
  <br />
  <input type="submit" value="Change" name="Change">
  <input type='hidden' name='user_token' value='3750e89f60e002bd83af3124d0b7afc6' />
>
```

Now we can further test our input fields knowing that Burp Suite will handle any anti-CSRF protections the web application will throw at us. There are further fine tuning that can be done with Burp Suite to make sure the macros only execute on the exact requests, but that will be for another time. Until next time keep on side-skirting common security protections!

Credits and Inspiration:

- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
- https://www.acunetix.com/wp-content/uploads/2013/04/csrf.png

- https://www.owasp.org/index.php/Anti_CSRF_Tokens_ASP.NET
- https://jsblog.insiderattack.net/anti-csrf-tokens-to-prevent-cross-site-request-forgery-csrf-79b9d7a5c079
- http://www.dvwa.co.uk/

SHARE

TAGS: BURPSUITE WEB APP WEB TOKEN BYPASS CYBERSECURITY DVWA KALI TOOLS OWASP PENTEST CSRF ANTI-CSRF

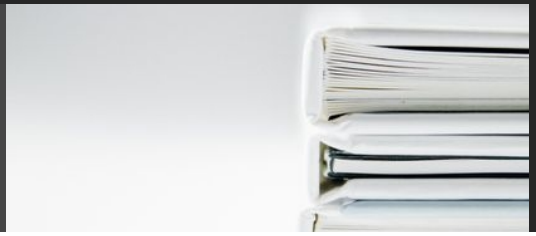
— ABOUT RYAN VILLARREAL

DENVER, COLORADO TWITTER

NEXT

Linux File Hierarchy Standard

JUNE 11, 2019



PREVIOUS

Adding Burp Suite CA Certificate to Kali Linux Certificate Store

MAY 25, 2019

— ABOUT —

Two cybersecurity professionals trying to get better at all things security.

— LATEST POSTS —

Information Gathering With Cobalt Strike
AUGUST 16, 2019

Navigating To A Web Site Step By Step
AUGUST 01, 2019

Atomic Red Team
JULY 30, 2019

— AUTHORS —

- Ryan Smith
- Bestest RedTeam
- Ryan Villarreal

— TAGS —

802.11 | 802.1X | ACTIVE DIRECTORY | ANTI-CSRF | AUTOMATE | AUTOMATION | AWS | BETA | BETTERCAP | BGP | BITCOIN | BLOODHOUND | BLUE TEAM | BURPSUITE | BYPASS | BYT3BL33D3R | C2 | CA | CAPTURE THE FLAG | CERTIFICATES | CLOUD | CLUSTER | CME | COBALT STRIKE | COMMAND AND CONTROL

COMMAND LINE CONTAINER CORS CRACKMAPEXEC CSRF CTF CYBERSECURITY DETECTION DOCKER DOMAIN ADMIN DOMAIN CONTROLLER DVWA ELEARNSECURITY ELK ELKSTACK ENUMERATION EWPT EXECUTIONPOLICY FREERADIUS GHOST GNU RADIO GOOGLE CLOUD GOPHISH GRAPH THEORY HACKING HACKRF HASHCAT HIJACKING HTTP HTTP/2 IMPACKET INFORMATION GATHERING INTERNAL NETWORK INTERNET OF THINGS JAVASCRIPT JUICESHOP JWT KALI LINUX KALI TOOLS KERBEROS LATERAL MOVEMENT LINUX MERLIN MICROSOFT MICROSOFT OFFICE MINING NE0ND0G NEO4J NETWORKING NULL SESSION OFFENSIVE SECURITY OFFSEC OPEN REDIRECT OSWP OWASP PASSWORD CRACKING PENETRATION TEST PENTEST PHISHING PHP PINEAPPLE PIXEL TRACKING PORTAINER POST EXPLOITATION POWERSHELL PROTOCOLS PYTHON RADIO FREQUENCY RECON RED TEAMING RED-BARON REDTEAMING REPORTING REVIEW RF RFC RTL-SDR S3 SAMBA SCANS SCAPY SCRIPTING SERVICE PRINCIPAL NAME SERVICES SHODAN SMB SMBCLIENT SOCIAL ENGINEERING SOFTWARE DEFINED RADIO SPN SWARM SYSADMIN TERRAFORM TERRAFORMFUN TRAINING UUID VULNERABILITY SCANNING WARDRIVING WEB APP WEB APPLICATION WEB TOKEN WEBAPP WIFI WIFU WIGLE WINDOWS WIRELESS WPA XSS