TAMDA Zohir
BTS SIO 2 option SLAM
Bloc 3 - Cybersécurité

le 15 octobre 2024

Les failles XSS

Sommaire

Définition..... 3
Faille XSS réfléchi via un contexte HTML..... 4
    Sans le mode sécurisé.....4
    Avec le mode sécurisé.....7
XSS permanent via une page affichant des logs.....12
    Sans le mode sécurisé..... 12

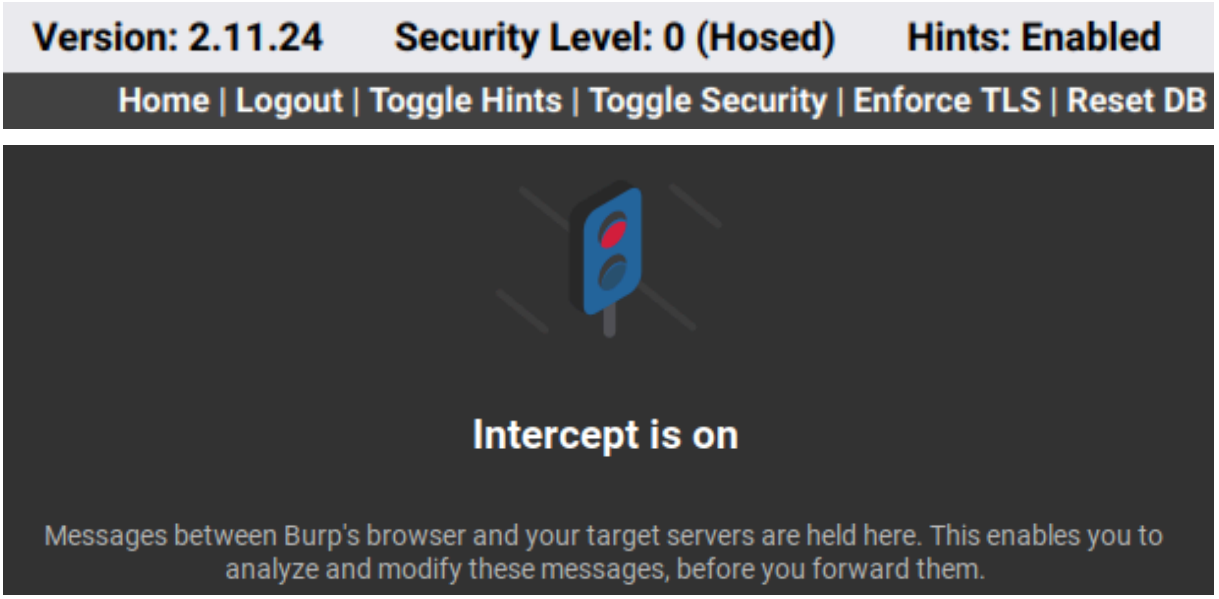
Définition

Une faille XSS (Cross Site Scripting) consiste à injecter du code malveillant et le renvoyer au serveur, que ce soit du code JavaScript ou du code qui est exécuté côté serveur (par exemple: PHP, Node.js, Ruby etc...).

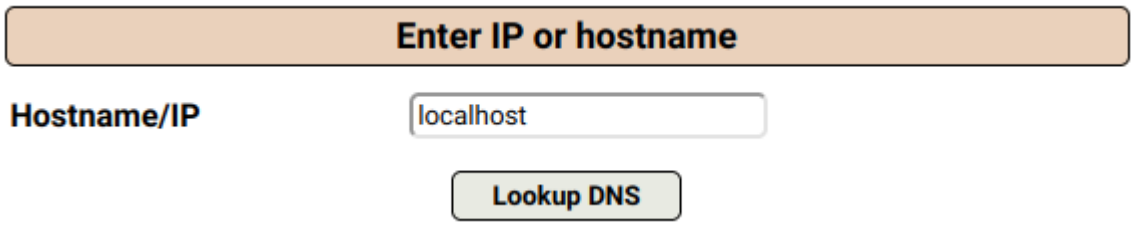
Faille XSS réfléchi via un contexte HTML

Sans le mode sécurisé

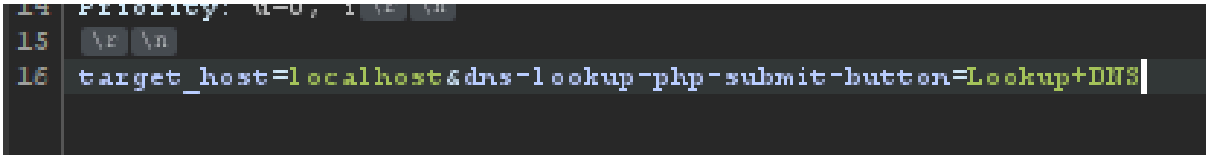
Le niveau de sécurité est défini à 0 sur Mutillidae. Côté proxy, le mode d'interception est activé. Ce qui permet de réaliser l'attaque XSS.



Dans la page "DNS Lookup" dans A7 : Cross Site Scripting (XSS) ⇒ Reflected (First Order), on injecte du code JavaScript qui consiste à afficher le cookie de session dans une boîte de dialogue.



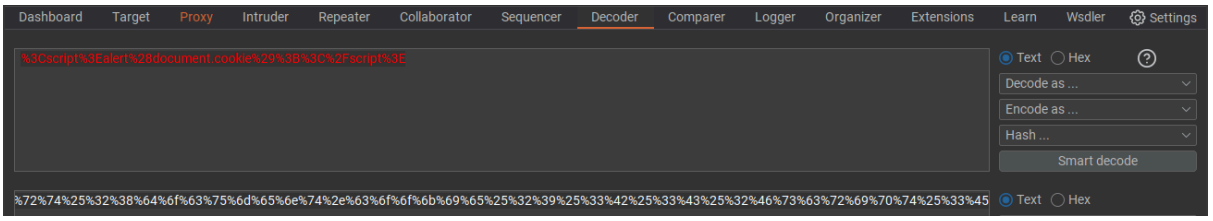
Après la soumission du formulaire, on voit qu'il y a un paramètre assez important : c'est la clé "target_host". Sa valeur est "localhost". Cette requête sera envoyée au répéteur (Repeater).




```
1265
1266                <div class="report-header">
                        Results for localhost
                    </div>
                    <pre class="output">
                        Server:         127.0.0.11
1267                    Address:  127.0.0.11#53
1268
1269                    Name:     localhost
1270                    Address:  127.0.0.1
1271                    Name:     localhost
1272                    Address:  ::1
1273
```

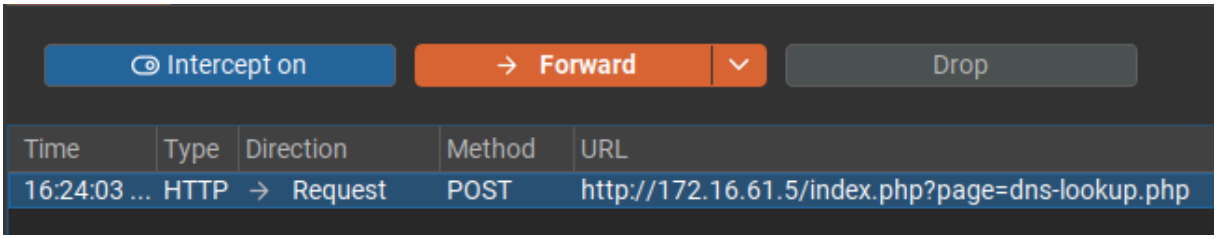
Après avoir cliqué sur “Send”, on obtient la réponse sous format HTML. Maintenant, on souhaite injecter du script malveillant qui a pour but d’afficher une cookie de session dans une alerte : <script>alert(document.cookie);</script>.

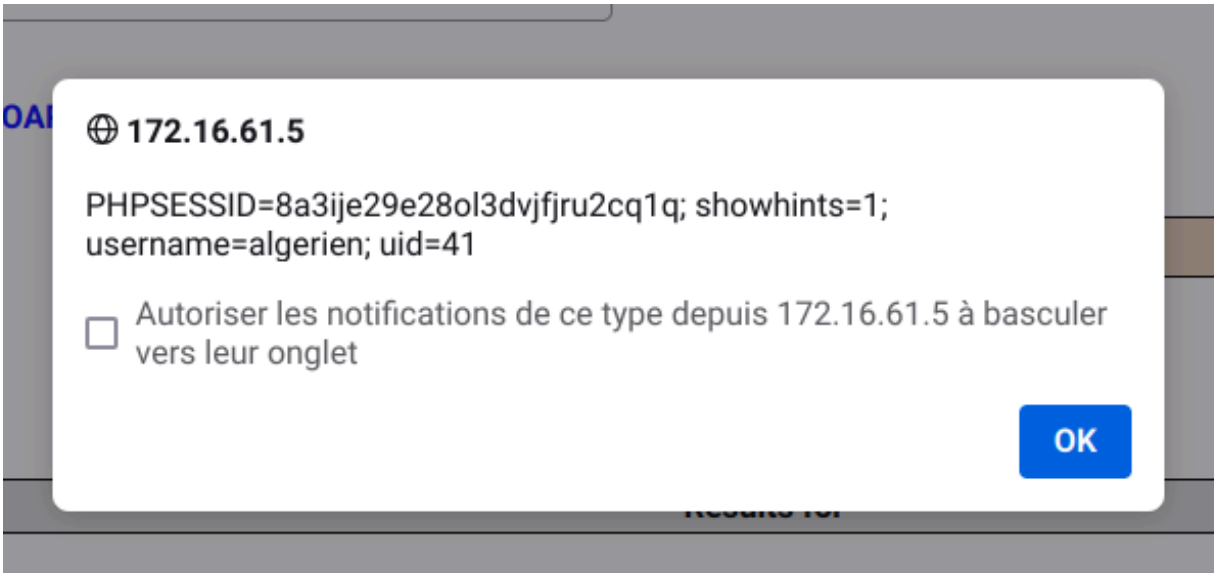
D’abord, il faut faire en sorte que ce script malveillant soit ajouté dans l’URL, c’est le rôle du décodeur (Decoder).



Avec cet outil, on peut encoder le code JavaScript afin de l’ajouter dans l’URL, plus précisément, l’ajouter pour le paramètre “target_host”.

Après cela, on peut laisser passer la requête en cliquant sur “Forward”.

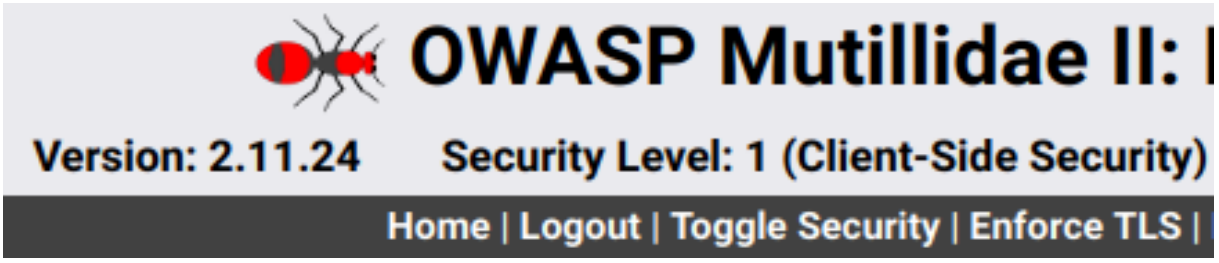




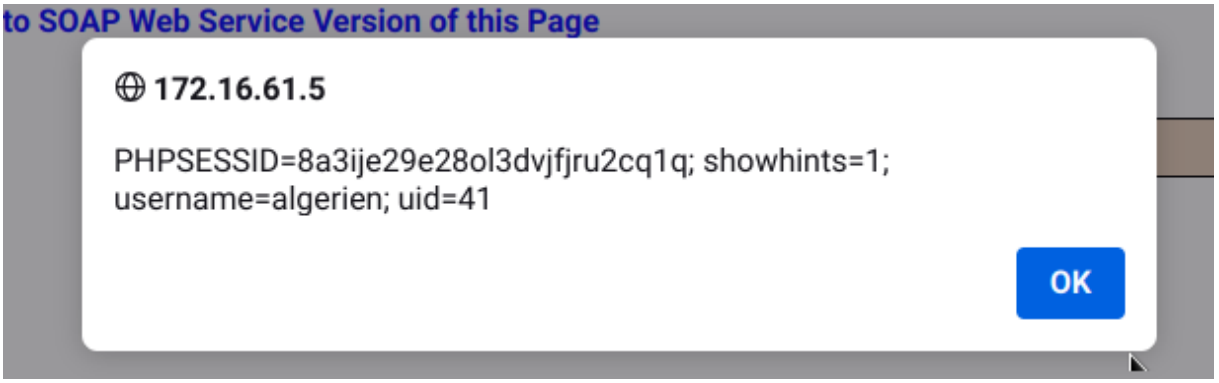
Résultat : la cookie de session s'affiche avec succès. L'attaque XSS a été réalisée.

Avec le mode sécurisé

Le niveau de sécurité est défini à 1.



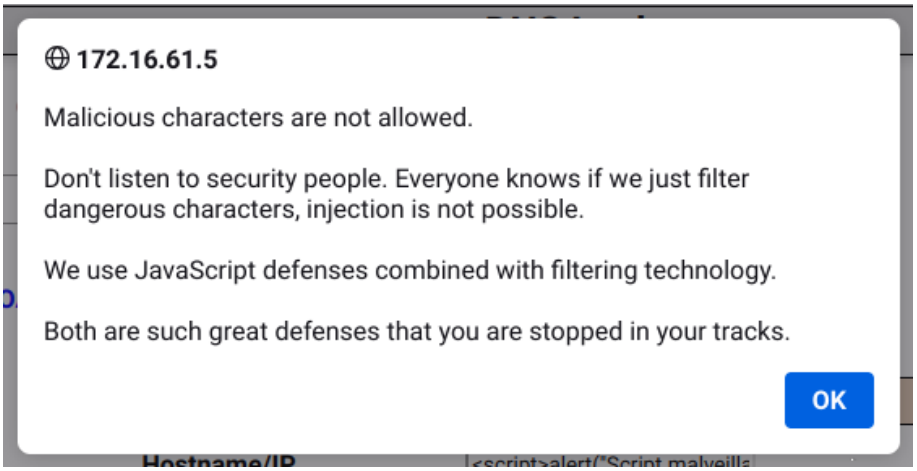
Les procédures sont identiques pour injecter du code malveillant dans BurpSuite.



Avec le niveau de sécurité défini à 1, l'attaque XSS est toujours présente si on tente d'injecter du script malveillant sur le logiciel BurpSuite.
Par contre, côté navigateur, il n'est pas possible d'écrire du code malveillant dans le formulaire car il y a des validations qui sont faites côté client, grâce aux attributs "minlength" et "maxlength".

```
<input id="idTargetHostInput" type="text" name="target_host"
size="20" autofocus="autofocus" minlength="1" maxlength="20"
required="required">
```

Même si on tente de supprimer ces attributs, cela affichera cette alerte après la soumission du formulaire. Ce message explique qu'il n'est pas autorisé d'utiliser les caractères spéciaux.



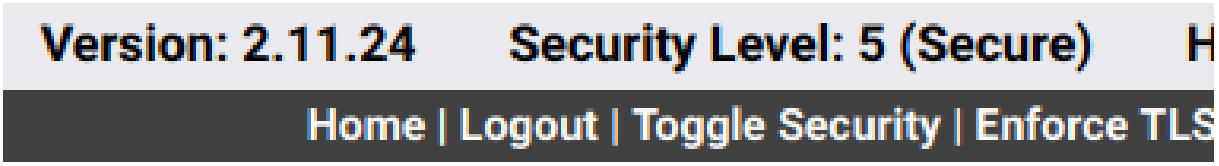

```
  case "1": // This code is insecure. No input validation is performed.
                  $lEnableJavaScriptValidation = true;
                  $lEnableHTMLControls = true;
                  $lProtectAgainstMethodTampering = false;
                  $lProtectAgainstCommandInjection=false;
                  $lProtectAgainstXSS = false;
  break;
```

```
<?php
if($lEnableJavaScriptValidation){
    echo "var lOSCommandInjectionPattern = /[;&|<>]/;";
    echo "var lCrossSiteScriptingPattern = /[<>=()]/;";
}else{
    echo "var lOSCommandInjectionPattern = /[]/;";
    echo "var lCrossSiteScriptingPattern = /[]/;";
}// end if
?>
```

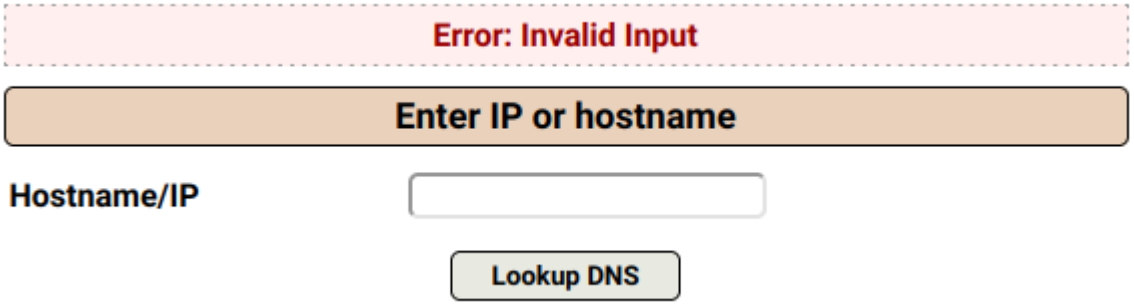
Voici les caractères spéciaux qui ne sont pas autorisés pour effectuer une attaque XSS.

```
<input  type="text" id="idTargetHostInput" name="target_host" size="20"
                autofocus="autofocus"
                <?php
                        if ($lEnableHTMLControls) {
                                echo 'minlength="1" maxlength="20" required="required"';
                        }// end if
                ?>
/>
```

Le niveau de sécurité est défini à 5.

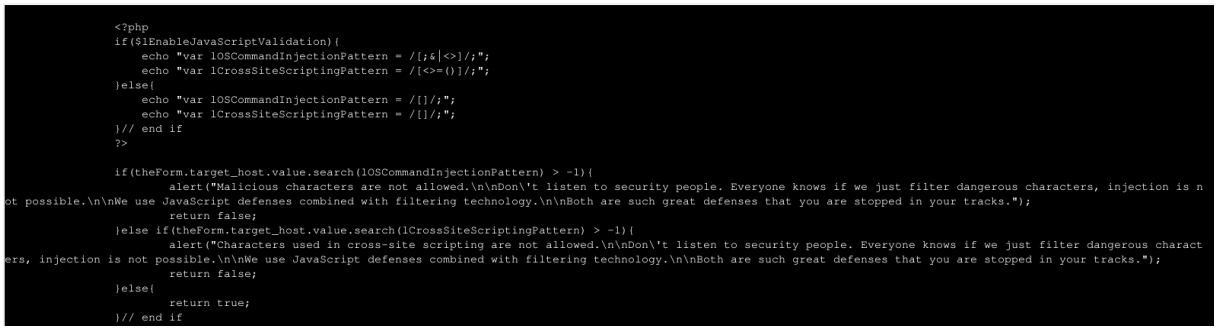


Même si on injecte du code JavaScript malveillant dans la requête, il n'est désormais plus possible de l'exécuter. Donc, le formulaire a bien été sécurisé.

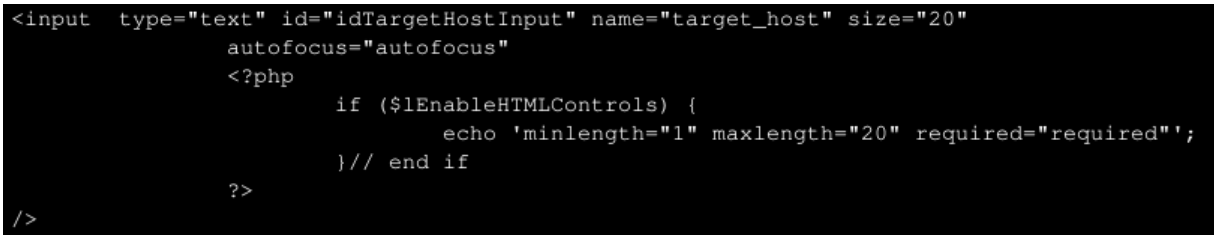



```
case "5": // This code is fairly secure
        $lProtectAgainstCommandInjection=true;
                $lEnableHTMLControls = true;
        $lEnableJavaScriptValidation = true;
                $lProtectAgainstMethodTampering = true;
                $lProtectAgainstXSS = true;
```

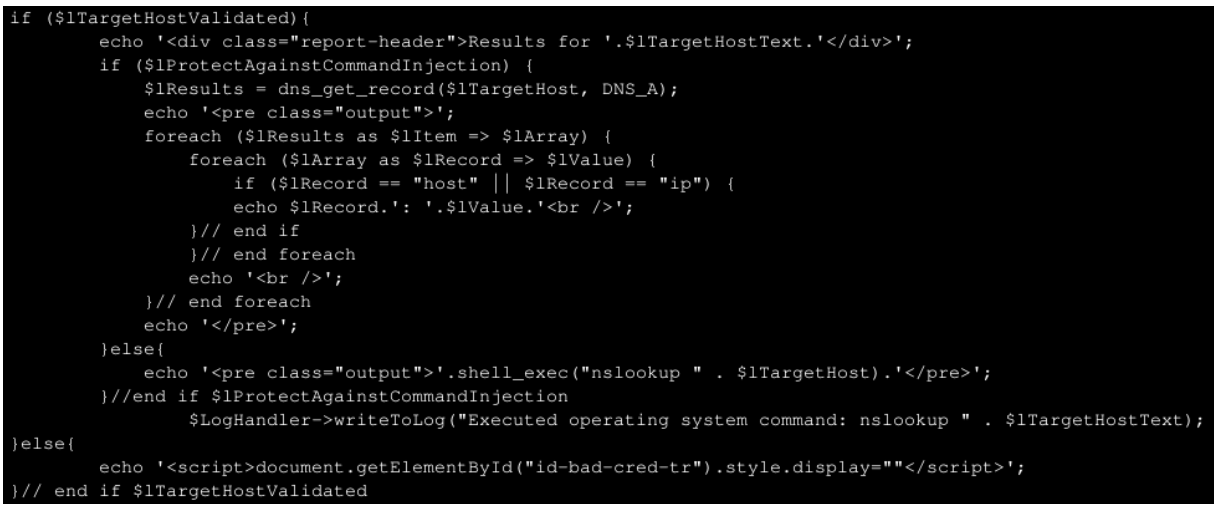
```
                    $lProtectAgainstMethodTampering?$lTargetHost = $_POST["target_host"]:$lTargetHost = $_REQUEST["target_host"];

            if ($lProtectAgainstCommandInjection) {
                        /* Protect against command injection.
                         * We validate that an IP is 4 octets, IPV6 fits the pattern, and that domain name is IANA format */
                    $lTargetHostValidated = preg_match(IPV4_REGEX_PATTERN, $lTargetHost) || preg_match(DOMAIN_NAME_REGEX_PATTERN, $lTargetHost) || preg_match(IPV6_REGEX_PA
TTERN, $lTargetHost);
            }else{
                    $lTargetHostValidated = true;                   // do not perform validation
            }// end if

            if ($lProtectAgainstXSS) {
                    /* Protect against XSS by output encoding */
                    $lTargetHostText = $Encoder->encodeForHTML($lTargetHost);
            }else{
                        $lTargetHostText = $lTargetHost;                //allow XSS by not encoding output
            }//end if
```



```
            <?php
            if($lEnableJavaScriptValidation){
                echo "var lOSCommandInjectionPattern = /[;&|<>]/;";
                echo "var lCrossSiteScriptingPattern = /[<>=()]/;";
            }else{
                echo "var lOSCommandInjectionPattern = /[]/;";
                echo "var lCrossSiteScriptingPattern = /[]/;";
            }// end if
            ?>

            if(theForm.target_host.value.search(lOSCommandInjectionPattern) > -1){
                    alert("Malicious characters are not allowed.\n\nDon\'t listen to security people. Everyone knows if we just filter dangerous characters, injection is n
ot possible.\n\nWe use JavaScript defenses combined with filtering technology.\n\nBoth are such great defenses that you are stopped in your tracks.");
                    return false;
            }else if(theForm.target_host.value.search(lCrossSiteScriptingPattern) > -1){
                    alert("Characters used in cross-site scripting are not allowed.\n\nDon\'t listen to security people. Everyone knows if we just filter dangerous charact
ers, injection is not possible.\n\nWe use JavaScript defenses combined with filtering technology.\n\nBoth are such great defenses that you are stopped in your tracks.");
                    return false;
            }else{
                    return true;
            }// end if
```



```
<input  type="text" id="idTargetHostInput" name="target_host" size="20"
                autofocus="autofocus"
                <?php
                        if ($lEnableHTMLControls) {
                                echo 'minlength="1" maxlength="20" required="required"';
                        }// end if
                ?>
/>
```



```
if ($lTargetHostValidated){
        echo '<div class="report-header">Results for '.$lTargetHostText.'</div>';
        if ($lProtectAgainstCommandInjection) {
            $lResults = dns_get_record($lTargetHost, DNS_A);
            echo '<pre class="output">';
            foreach ($lResults as $lItem => $lArray) {
                foreach ($lArray as $lRecord => $lValue) {
                    if ($lRecord == "host" || $lRecord == "ip") {
                    echo $lRecord.': '.$lValue.'<br />';
                }// end if
                }// end foreach
                echo '<br />';
            }// end foreach
            echo '</pre>';
        }else{
            echo '<pre class="output">'.shell_exec("nslookup " . $lTargetHost).'</pre>';
        }//end if $lProtectAgainstCommandInjection
                $LogHandler->writeToLog("Executed operating system command: nslookup " . $lTargetHostText);
}else{
        echo '<script>document.getElementById("id-bad-cred-tr").style.display=""</script>';
}// end if $lTargetHostValidated
```

`$lProtectAgainstMethodTampering?$lTargetHost = $_POST["target_host"]:$lTargetHost = $_REQUEST["target_host"];`

Le rôle de cette instruction est de récupérer la valeur de l’hôte cible que si le verbe HTTP n’est rien d’autre que POST.

```
if ($lProtectAgainstCommandInjection) {
	$lTargetHostValidated = preg_match(IPV4_REGEX_PATTERN, $lTargetHost) ||
			preg_match(DOMAIN_NAME_REGEX_PATTERN, $lTargetHost) ||
			preg_match(IPV6_REGEX_PATTERN, $lTargetHost);
}else{
	$lTargetHostValidated = true; // do not perform validation
} // end if
```

Pour éviter l'injection de commandes, il faut vérifier la valeur de la cible hôte grâce aux expressions régulières. Ce code vérifie si cette valeur est une adresse IPv4 ou IPv6 ou un nom de domaine; si une de ces conditions ne sont pas réunies, il ne sera pas valide et la commande qui permet de récupérer les informations de l'hôte ne sera pas exécutée.

Pour lutter contre les attaques XSS, la fonction encodeForHTML() est utilisée pour encoder les caractères spéciaux.

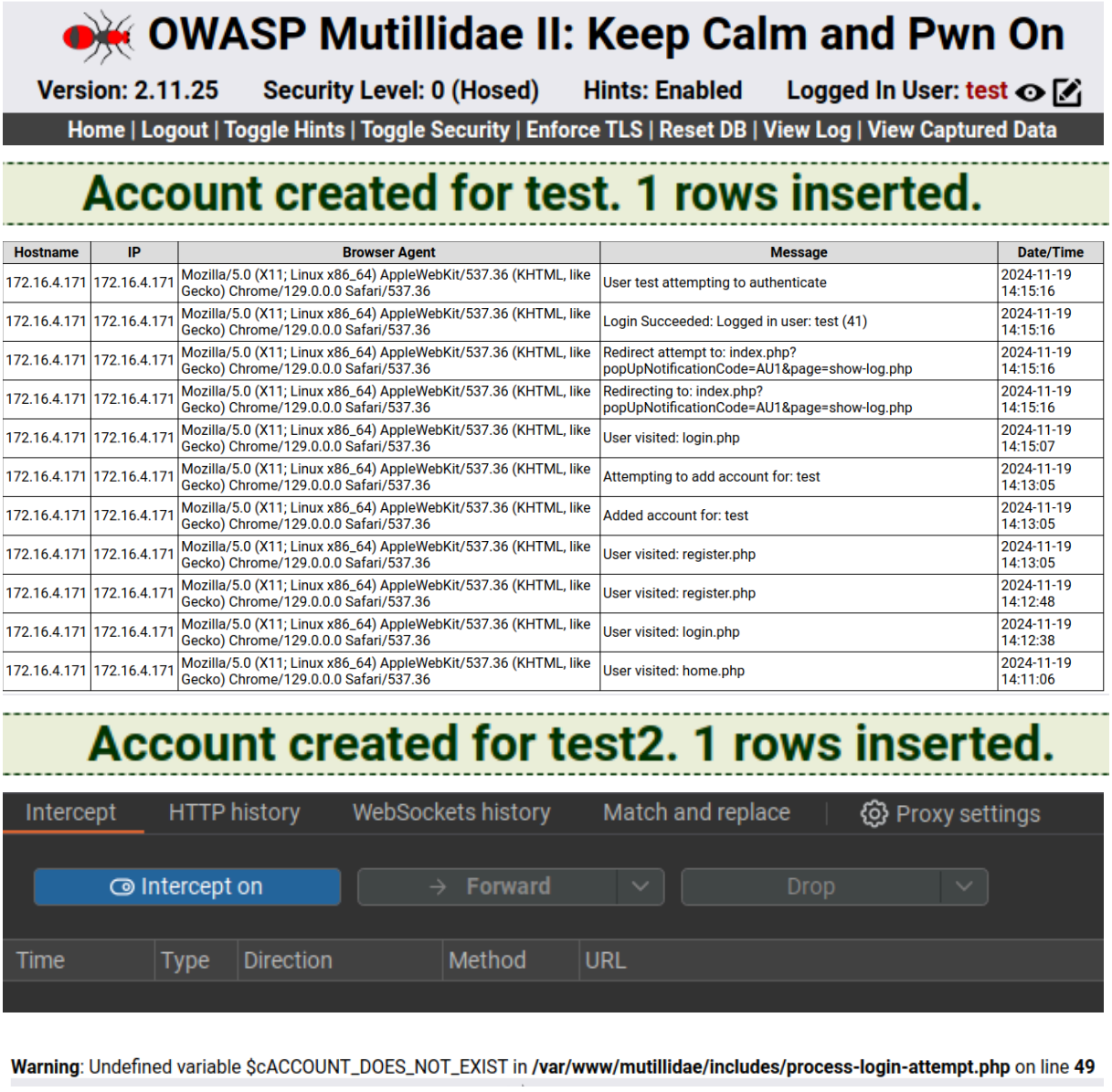
```
if ($lProtectAgainstXSS) {
	/* Protect against XSS by output encoding */
	$lTargetHostText = $Encoder->encodeForHTML($lTargetHost);
} else {
	$lTargetHostText = $lTargetHost; //allow XSS by not encoding output
} //end if
```

Avec le niveau de protection n°5, on a activé :

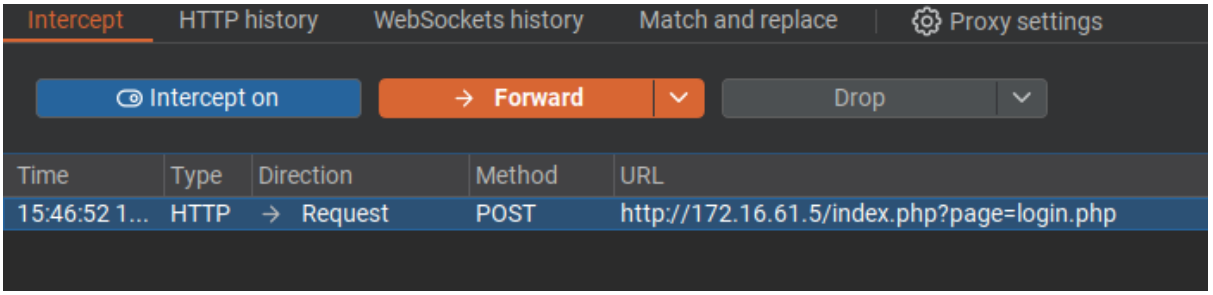
- la protection contre les injections de commandes.
- les validations HTML pour les balise <input>.
- les validations JavaScript.
- la protection "method tampering".
- la protection contre les attaques XSS.

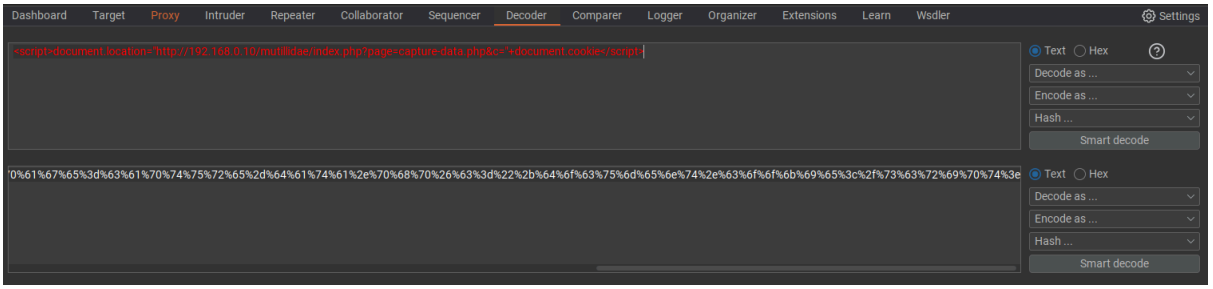
XSS permanent via une page affichant des logs

Sans le mode sécurisé



Hostname	IP	Browser Agent	Message	Date/Time
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	User visited: /var/www/mutillidae/home.php	2024-11-19 14:43:08
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	User test attempting to authenticate	2024-11-19 14:43:01
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	Login Succeeded: Logged in user: test (41)	2024-11-19 14:43:01
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	Redirect attempt to: index.php? popUpNotificationCode=AU1	2024-11-19 14:43:01
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	Redirecting to: index.php? popUpNotificationCode=AU1	2024-11-19 14:43:01
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	User visited: /var/www/mutillidae/home.php	2024-11-19 14:43:01
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	User test attempting to authenticate	2024-11-19 14:42:52
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	Login Failed: Password for test incorrect	2024-11-19 14:42:52
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	User visited: login.php	2024-11-19 14:42:52
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	User visited: login.php	2024-11-19 14:42:48
172.16.4.171	172.16.4.171	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101	Attempting to add account for: test	2024-11-19





View Captured Data

Data Capture Page

This page is designed to capture any parameters sent and store them in a file and a database table. It loops through the POST and GET parameters and records them to a file named **captured-data.txt**. On this system, the file should be found at **/tmp/captured-data.txt**. The page also tries to store the captured data in a database table named captured_data and logs the captured data. There is another page named captured-data.php that attempts to list the contents of this table.

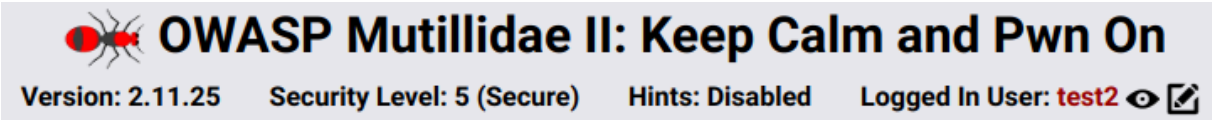
The data captured on this request is: page = capture-data.php PHPSESSID = o0l5vcpoqb74un2i8rkbrdo9ri showhints = 1 username = user2 uid = 41

Would it be possible to hack the hacker? Assume the hacker will view the captured requests with a web browser.

			Firefox/131.0		username = test2 uid = 41	
172.16.4.171	172.16.4.171	38334	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0	http://172.16.61.5/mutillidae/index.php?page=show-log.php	page = capture-data.php c = PHPSESSID=o0l5vcpoqb74un2i8rkbrdo9ri; showhints=1; username=test2; uid=41 PHPSESSID = o0l5vcpoqb74un2i8rkbrdo9ri showhints = 1 username = test2 uid = 41	2024-11-19 15:30:50
			Mozilla/5.0 (X11;		page = capture-data.php c =	

Avec le mode sécurisé

Le niveau de sécurité est défini à 5, ce qui permet de tester l'attaque sur un site sécurisé.



Les procédures sont identiques pour la réalisation de l'attaque XSS.


```php
switch ($_SESSION["security-level"]){
        default: // Default case: This code is insecure
        case "0": // This code is insecure
        case "1": // This code is insecure
                // DO NOTHING: This is insecure
                $lEncodeOutput = false;
                $lLimitOutput = false;
        break;

        case "2":
        case "3":
        case "4":
        case "5": // This code is fairly secure
                /*
                 * NOTE: Input validation is excellent but not enough. The output must be
                 * encoded per context. For example, if output is placed in HTML,
                 * then HTML encode it. Blacklisting is a losing proposition. You
                 * cannot blacklist everything. The business requirements will usually
                 * require allowing dangerous charaters. In the example here, we can
                 * validate username but we have to allow special characters in passwords
                 * least we force weak passwords. We cannot validate the signature hardly
                 * at all. The business requirements for text fields will demand most
                 * characters. Output encoding is the answer. Validate what you can, encode it
                 * all.
                 */
                // encode the output following OWASP standards
                // this will be HTML encoding because we are outputting data into HTML
                $lEncodeOutput = true;
                $lLimitOutput = true;
        break;
}// end switch
```