Ossama Ashraf : ossama.ashraf@etu.univ-cotedazur.fr
Ralph El Chalfoun : ralph.el-chalfoun@etu.unice.fr
Jérémy Hirth Daumas : jeremy.hirth-daumas@etu.unice.fr

Attack	Link	Vulnerable program	Exploit	Defense	Page
XSS RULE 0	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-0-never-insert-untrusted-data-except-in-allowed-locations	Yes	Yes	Yes	2
XSS RULE 1	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-1-html-encode-before-inserting-untrusted-data-into-html-element-content	Yes	Yes	Yes	3
XSS RULE 2	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-2-attribute-encode-before-inserting-untrusted-data-into-html-common-attributes	Yes	Yes	Yes	4
XSS RULE 3	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-3-javascript-encode-before-inserting-untrusted-data-into-javascript-data-values	Yes	Yes	Yes	5
XSS RULE 3.1	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-31-html-encode-json-values-in-an-html-context-and-read-the-data-with-jsonparse	Yes	Yes	Yes	6-7
XSS RULE 4	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-4-css-encode-and-strictly-validate-before-inserting-untrusted-data-into-html-style-property-values	Yes	Yes	Yes	8
XSS RULE 5	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-5-url-encode-before-inserting-untrusted-data-into-html-url-parameter-values	Yes	Yes	Yes	9
XSS RULE 6	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-6-sanitize-html-markup-with-a-library-designed-for-the-job	Yes	Yes	Yes	10
XSS RULE 7	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-7-avoid-javascript-urls	Yes	Yes	Yes	11
DOM BASED XSS	https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html	Yes	Yes	Yes	12
SESSION MANAGEMENT httponly-attribute	https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#httponly-attribute	Yes	Yes	Yes	13
SESSION MANAGEMENT samesite-attribute	https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#samesite-attribute	Yes (But certificate require to test)	Yes	Yes (But certificate require to test)	14
SESSION MANAGEMENT secure-attribute	https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#secure-attribute	Yes	Yes	Yes	15
XXE	https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html	Yes	Yes	Yes	16
SQL INJECTION	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html	Yes	Yes	Yes	17-18
PASSWORD STORAGE	https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html	Yes	Yes	Yes	19

RULE #0 - HTML Encode Before Inserting Untrusted Data into HTML Element Content

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-0-never-insert-untrusted-data-except-in-allowed-locations

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 0(Vulnérabilité)</title>
</head>  <body>
        <h1>Liste de courses</h1>
        <form type="get" action="">
            <input type="text" name="keyword" />
            <input type="submit" value="Ajouter" />
        </form>
        <br>
        <div id="list">
           <!-- <?php if(!empty($_GET['keyword'])){ echo "Vous avez ajouté : ".$_GET['keyword']; } ?> -->
        </div>
    </body>
</html>
```

Exploit :

--> <script>alert("XSS")</script> <!--

Defense:

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 0(Defense)</title>
</head>  <body>
        <h1>Liste de courses</h1>

        <form type="get" action="">
            <input type="text" name="keyword" />
            <input type="submit" value="Ajouter" />
        </form>
        <br>
        <div id="list">
           <!-- <?php if(!empty($_GET['keyword'])){ echo "Vous avez ajouté :
".htmlspecialchars($_GET['keyword']); } ?> -->
        </div>
    </body>
</html>
```

RULE #1 - HTML Encode Before Inserting Untrusted Data into HTML Element Content
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-1-html-encode-before-inserting-untrusted-data-into-html-element-content

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 1(Vulnérabilité)</title>
</head>  <body>
        <h1>Liste de courses</h1>

        <form type="get" action="">
            <input type="text" name="keyword" />
            <input type="submit" value="Ajouter" />
        </form>
        <br>
        <div id="list">
            <?php
            if(!empty($_GET['keyword'])){
                echo "Vous avez ajouté : ".$_GET['keyword'];
            }
            ?>
        </div>
    </body>
</html>
```

Exploit :

```
<script>
   var parent = document.getElementById('list');
   var a = document.createElement('a');
   var linkText = document.createTextNode('Nutella');
   a.appendChild(linkText);
   a.title = 'Nutella';
   a.href = 'https://hackernicois.herokuapp.com/';
   parent.appendChild(a);
 </script>
```

Defense:

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 1(Defense)</title>
 </head>  <body>
        <h1>Liste de courses</h1>
        <form type="get" action="">
            <input type="text" name="keyword" />
            <input type="submit" value="Ajouter" />
        </form>
        <br>
        <div id="list">
            <?php
            if(!empty($_GET['keyword'])){
                echo "Vous avez ajouté : ".htmlspecialchars($_GET['keyword']);
            }
            ?>
 </div>    </body> </html>
```

RULE #2 - Attribute Encode Before Inserting Untrusted Data into HTML Common Attributes
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-2-attribute-encode-before-inserting-untrusted-data-into-html-common-attributes

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 2(Vulnérabilité)</title>
 </head>  <body>
        <h1>Voici une pomme</h1>
        <p>Mais vous pouvez choisir à quoi cette pomme ressemblera en changeant l'image :) !</p>
        <form type="get" action="">
            <input type="text" name="link" />
            <input type="submit" value="Choisir la photo" />
        </form>
        <?php if(!empty($_GET['link'])){ $link = $_GET['link'];  }else{
$link="https://st2.depositphotos.com/7036298/10694/i/600/depositphotos_106948346-stock-photo-ripe-red-apple-with-green.jpg"; }   ?>

        <img src=<?php echo $link?> id="pomme" width="300px"></img>


    </body>
</html>
```

Exploit :
https://fotomelia.com/wp-content/uploads/edd/2015/09/sites-de-t%C3%A9l%C3%A9chargement-gratuit-d-images-photos-libres-de-droits58-1560x1181.jpg onclick='alert("XSS")'
Il faut cliquer sur la photo pour XSS

Defense:

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 2(Defense)</title>
 </head>  <body>
        <h1>Voici une pomme</h1>
        <p>Mais vous pouvez choisir à quoi cette pomme ressemblera en changeant l'image :) !</p>
        <form type="get" action="">
            <input type="text" name="link" />
            <input type="submit" value="Choisir la photo" />
        </form>

        <?php if(!empty($_GET['link'])){ $link = $_GET['link'];}else{
$link="https://st2.depositphotos.com/7036298/10694/i/600/depositphotos_106948346-stock-photo-ripe-red-apple-with-green.jpg"; }
             $link = str_replace(array("'", "\"", "&quot;"), "", htmlspecialchars($link ) );  ?>

        <img src="<?php echo $link?>" id="pomme" width="300px"></img>
    </body>
</html>
```

RULE #3 - JavaScript Encode Before Inserting Untrusted Data into JavaScript Data Values

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-3-javascript-encode-before-inserting-untrusted-data-into-javascript-data-values

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 3(Vulnérabilité)</title>
 </head>  <body>
        <h1>Voici un carré</h1>
        <p>Choisissez sa taille en pixel</p>

        <form type="get" action="">
            <input type="text" name="pixel" />
            <input type="submit" value="Changer" />
        </form> <br>
        <span id="carre"></span>
        <style>
            #carre { width  : 50px; height : 50px; position: fixed; background : green;  }
        </style>
        <script>
            var taille =  "<?php if(!empty($_GET['pixel'])){ echo $_GET['pixel'];  }   ?>"
            document.querySelector('#carre').style =    'width: ' + taille  +'px;' +
                                                        'height: ' +   taille + 'px;';
</script></body></html>
```

Exploit :

" ;alert('xss');//

Defense:

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 3(Defense)</title>
 </head>  <body>
        <h1>Voici un carré</h1>
        <p>Choisissez sa taille en pixel</p>
        <form type="get" action="">
            <input type="text" name="pixel" />
            <input type="submit" value="Changer" />
        </form>
        <br>
        <span id="carre"></span>
        <style>
            #carre { width  : 50px; height : 50px; position: fixed;  background : green; }
        </style>
            <?php
            $taille= "";
            if(!empty($_GET['pixel'])){
                $taille = htmlspecialchars($_GET['pixel']);
                $taille = preg_replace('#\'#','&apos;',$taille);     // Change [ ' ] To [ &apos; ]
                $taille = preg_replace('#\"#','&quot;',$taille);     // Change [ " ] To [ &quot; ]
                $taille = preg_replace('#\\\\#','',$taille);    // To remove [ \ ]
                $taille = htmlspecialchars($taille);
            }
            ?>
        <script>
            var taille =  "<?php echo $taille ?>"
            document.querySelector('#carre').style =    'width: ' + taille  +'px;' +
                                                        'height: ' +   taille + 'px;';
        </script> </body> </html>
```

RULE #3.1 - HTML Encode JSON values in an HTML context and read the data with JSON.parse

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-31-html-encode-json-values-in-an-html-context-and-read-the-data-with-jsonparse

Vulnerability :

```
<!DOCTYPE html> <html lang="fr"> <head> <title>XSS Rule 3.1(Vulnérabilité)</title> <meta charset="utf-8" />
      <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    </head> <body>
        <h1>Liste de courses</h1>
        <input id="keyword" type="text" name="keyword" placeholder="Article"/>
        <input id="value" type="text" name="value" placeholder="Quantité"/>
        <button type="button"  onclick="tojson();">Ajouter</button>
        <div id="list">
            L'article : <p id="article"></p> a bien été ajouté
            en quantité : <p id="nombre"></p>
        </div>
        <script>
            var list = document.getElementById("list");
            function tojson(){
                var keyword =  document.getElementById("keyword").value;
                var value = document.getElementById("value").value;
                var xhr = new XMLHttpRequest();
                var url ="http://origin2:80/serv.php"
                xhr.open("POST", url, true);
                xhr.setRequestHeader('Access-Control-Allow-Origin','*');
                xhr.setRequestHeader('Access-Control-Allow-Methods','POST, GET');
                xhr.setRequestHeader("Content-Type", "application/json");
                xhr.onreadystatechange = function () {
                    if (xhr.readyState === 4 && xhr.status === 200) {
                        var json = JSON.parse(xhr.responseText);
                        document.getElementById("article").innerHTML = json.keyword;
                        document.getElementById("nombre").innerHTML = json.value;
                    }
                };
                var data = JSON.stringify({"keyword": keyword, "value": value});
                xhr.send(data); }
        </script> </body> </html>
```

(Serveur sur virtual host origin2)

```
<?php
    // Handling data in JSON format on the server-side using PHP
    //
    header("Access-Control-Allow-Origin: *");
    header("Access-Control-Allow-Headers: *");
    header("Access-Control-Allow-Methods: POST, GET");
    header("Content-Type: application/json");
    // build a PHP variable from JSON sent using GET method
    $v = json_decode(stripslashes(file_get_contents("php://input")));
    // encode the PHP variable to JSON and send it back on client-side
    echo json_encode($v);

?>
```

Exploit à mettre dans n'importe quel input puis cliquer sur le bouton qui apparaît :
<button onclick='alert(1);'>CLICK ME</button>

Le premier programme à lancer sur un virtual host "origin1" et le second sur un autre nommé "origin2".

Defense script bundleXSS.js requis au même emplacement que le code suivant:

```
<!DOCTYPE html>
<html lang="fr">
    <head>
      <meta charset="utf-8" />
      <title>XSS Rule 3.1(Defense)</title>
      <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
      <script  src="https://github.com/iBananos/Secu-Web2021/blob/main/src/RULE_3-1_XSS/bundle.js"></script>
    </head>
    <body>

        <h1>Liste de courses</h1>
        <input id="keyword" type="text" name="keyword" placeholder="Article"/>
        <input id="value" type="text" name="value" placeholder="Quantité"/>
        <button type="button"  onclick="tojson();">Ajouter</button>

        <div id="list">
            L'article : <p id="article"></p> a bien été ajouté
            en quantité : <p id="nombre"></p>
        </div>
        <script>
            var list = document.getElementById("list");
            function tojson(){
                var keyword =  document.getElementById("keyword").value;
                var value = document.getElementById("value").value;
                var xhr = new XMLHttpRequest();
                var url ="http://origin2:80/serv.php"
                xhr.open("POST", url, true);
                xhr.setRequestHeader('Access-Control-Allow-Origin','*');
                xhr.setRequestHeader('Access-Control-Allow-Methods','POST, GET');
                xhr.setRequestHeader("Content-Type", "application/json");
                xhr.onreadystatechange = function () {
                    if (xhr.readyState === 4 && xhr.status === 200) {
                        var json = JSON.parse(xhr.responseText);
                        document.getElementById("article").innerHTML = json.keyword;
                        document.getElementById("nombre").innerHTML = json.value;
                    }
                };
                var data = serialize({"keyword": keyword, "value": value});
                //data = JSON.stringify(data);
                xhr.send(data);
            }
        </script>
    </body>
</html>
```

RULE #4 - CSS Encode And Strictly Validate Before Inserting Untrusted Data into HTML Style Property Values

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-4-css-encode-and-strictly-validate-before-inserting-untrusted-data-into-html-style-property-values

Vulnerability :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
      <meta charset="utf-8" /> <title>XSS Rule 4(Vulnérabilité)</title>
    </head>
    <body>
        <h1>Voici un carré</h1>
        <p>Choisissez l'url d'une photo à mettre dans ce jolie carré</p>
        <?php $url ="" ; if(!empty($_GET['url'])){ $url =  $_GET['url'];  }  ?>
        <form type="get" action="">
            <input type="text" name="url" />
            <input type="submit" value="Changer" />
        </form>
        <br>
        <span id="carre"></span>
        <style>
            #carre {  height : 400px; width : 400px; position: fixed;  background-image:url(<?php echo $url
?>);
           background-position:center center; border-width:6px; border-style:dotted; border-color:black;}
       </style>   </body>  </html>
```

Exploit :

https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/gettyimages-1185282377.jpg);</style> <script>alert('XSS');</script><style>

Defense:

```
<!DOCTYPE html>
<html lang="fr">
    <head>
      <meta charset="utf-8" /> <title>XSS Rule 4(Defense)</title>
    </head>
    <body>
        <h1>Voici un carré</h1>
        <p>Choisissez l'url d'une photo à mettre dans ce jolie carré</p>
        <?php  $url ="" ; if(!empty($_GET['url'])){ $url =  htmlspecialchars($_GET['url']);  } ?>

        <form type="get" action="">
            <input type="text" name="url" />
            <input type="submit" value="Changer" />
        </form> <br>
        <span id="carre"></span>
        <style>
            #carre { height : 400px; width : 400px; position: fixed; background-image:url("<?php echo $url
?>");
           background-position:center center; border-width:6px;border-style:dotted;  border-color:black; }
        </style>
    </body>
</html>
```

RULE #5 - URL Encode Before Inserting Untrusted Data into HTML URL Parameter Values

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-5-url-encode-before-inserting-untrusted-data-into-html-url-parameter-values

Vulnerability :

```
<!DOCTYPE html><html lang="fr">    <head>      <meta charset="utf-8" />
<title>XSS Rule 5(Vulnérabilité)</title>   </head>    <body>
        <h1>Vous allez voir un carré</h1>
        <p>Choisissez sa taille en pixel</p>
        <form type="get" action="">
            <input type="text" name="pixel" />
            <?php
                $url = $_SERVER['PHP_SELF'] ;
                if(!empty($_GET['pixel'])){
                    $monUrl = $url."?value=".$_GET['pixel']; echo "<a href=". $monUrl.">Voir le carré</a>";
                }else {
                    $monUrl = $url."?value="; echo "<input type='submit' value='Valider la taille' />";}
                $pixel = 0;
                if(!empty($_GET['value'])){$pixel = htmlspecialchars($_GET['value']);} ?>
        </form> <br>
       <span id="carre"></span>
        <style>
            #carre { width  : <?php echo $pixel; ?>px; height : <?php echo $pixel; ?>px; position: fixed;
            background : green; }
        </style>  </body> </html>
```

Exploit :

"> <script> alert("XSS");</script>

ou

" onclick=alert('XSS');

Defense:

```
<!DOCTYPE html>
<html lang="fr">
    <head>
      <meta charset="utf-8" /> <title>XSS Rule 5(Defense)</title>
    </head>
    <body>
        <h1>Vous allez voir un carré</h1>
        <p>Choisissez sa taille en pixel</p>
        <form type="get" action="">
            <input type="text" name="pixel" />
            <?php
                $url = $_SERVER['PHP_SELF'] ;
                if(!empty($_GET['pixel'])){$monUrl = $url."?value=".htmlspecialchars($_GET['pixel']);
                    echo '<a href="'. $monUrl.'">Voir le carré</a>';
                }else {
                    $monUrl = $url."?value="; echo "<input type='submit' value='Valider la taille' />";  }
                $pixel = 0;
                if(!empty($_GET['value'])){$pixel = htmlspecialchars($_GET['value']);}?>
        </form> <br>
        <span id="carre"></span>
        <style>
            #carre {
                width  : <?php echo $pixel; ?>px; height : <?php echo $pixel; ?>px; position: fixed;
                background : green; }
        </style> </body>  </html>
```

RULE #6 - Sanitize HTML Markup with a Library Designed for the Job

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-6-sanitize-html-markup-with-a-library-designed-for-the-job

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /><title>XSS Rule 6(Vulnérabilité)</title>
</head> <body>
<h1>Welcome to our Guest Book, Leave us a Message PLEASE! </h1>
<form method="GET">
<input  id="message" name="message">
<input type="submit" value="Leave a message">
<form>
<h2>All the messages left by guests </h2>
<div id="show">
<?php
    if(!empty($_GET['message'])){ $str= $_GET["message"] ; $str = htmlspecialchars($str);
        $see = "<img src='".$str."' width=30% height=30%> "; echo $see; } ?>
</div>
</body>
</html>
```

Exploit :
'onerror='alert("XSS")"

Defense : PHP HTML Purifier à télécharger sur ce site : http://htmlpurifier.org/ et à dezipper dans le même dossier que le programme suivant :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>XSS Rule 6(Defense)</title> </head> <body>
<h1>Welcome to our Guest Book, Leave us a Message PLEASE! </h1>
<form method="GET">
<input  id="message" name="message">
<input type="submit" value="Leave a message">
<form>
<h2>All the messages left by guests </h2>
<div id="show">
<?php
include('./htmlpurifier-4.13.0/library/HTMLPurifier.auto.php');
$purifierConfig = HTMLPurifier_Config::createDefault();
$purifierConfig->set('Core.Encoding', 'UTF-8');
$purifierConfig->set('HTML.Allowed',
'a[href|title],img[title|src|alt],em,strong,cite,blockquote,code,ul,ol,li,dl,dt,dd,p,br,h1,h2,h3,h4,h5,h6,sp
an,*[style]');
$purifier = new HTMLPurifier($purifierConfig);
if(!empty($_GET['message'])){
    $str= $_GET["message"] ;
    $see = "<img src='".$str."' width=30% height=30%> ";
    echo $purifier->purify($see);
}
?>

</div>
</body>
</html>
```

RULE #7 - Avoid JavaScript URLs

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-7-avoid-javascript-urls

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /><title>XSS Rule 7(Vulnérabilité)</title>
</head>
    <body>
        <h1>Cliquez pour vous y rendre</h1>
        <a id="link" href="">Site</a>
        <br>
        <script>
            var url = prompt("Entrez l'url du site sur lequel vous souhaitez vous rendre");
            document.getElementById('link').href=url;
        </script>
    </body>
</html>
```

Exploit :

javascript:alert('XSS !');

Defense:

```
<!DOCTYPE html>
<html lang="fr">
    <head>
      <meta charset="utf-8" /> <title>XSS Rule 7(Defense)</title> </head>
    <body>
        <h1>Cliquez pour vous y rendre</h1>
        <div id="list">
            <a id="link" href="">Site</a>
        </div>
        <br>
        <script>
            var url = prompt("Entrez l'url du site sur lequel vous souhaitez vous rendre");
            //var result = toString(url.protocol); // Retourne:"https:"
            //console.log(result);
            if(validURL(url)){
                document.getElementById('link').href=url;
            }else{
                url =""
                alert("Protection XSS, lien non conforme supprimé");
                var parentElement = document.getElementById("list");
                var imgElement = document.getElementById('link');
                parentElement.removeChild(imgElement);
            }

            function validURL(str) {
                var pattern = new RegExp('^(https?:\\/\\/)?'+ // protocol
                    '((([a-z\\d]([a-z\\d-]*[a-z\\d])*)\\.)+[a-z]{2,}|'+ // domain name
                    '((\\d{1,3}\\.){3}\\d{1,3}))'+ // OR ip (v4) address
                    '(\\:\\d+)?(\\/[-a-z\\d%_.~+]*)*'+ // port and path
                    '(\\?[;&a-z\\d%_.~+=-]*)?'+ // query string
                    '(\\#[-a-z\\d_]*)?$','i'); // fragment locator
                return !!pattern.test(str);
            }
        </script>
    </body> </html>
```

DOM based XSS Prevention Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html

Vulnerability :

```
<?php

$name = "username";
$value = "Ossama";
$expires = time() + (86400 * 30) ;//30 days
setcookie($name , $value , $expires );?>

<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /><title>DOM Based XSS(Vulnérabilité)</title>
</head> <body>
      <h1>DOM BASED XSS DEMO</h1>
      <div id="result">Fail</div>
      <script>
        var num = document.URL.split("num=")[1];
        document.querySelector("#result").innerHTML = eval(num);
      </script>
    </body>
</html>
```

Exploit à ajouter à la fin de l'url :

?num=alert(document.cookie)

Defense:

```
<?php
$name = "username";
$value = "Ossama";
$expires = time() + (86400 * 30) ;//30 days
setcookie($name , $value , $expires );?>


<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /> <title>DOM Based XSS(Defense)</title><script
src="https://github.com/iBananos/Secu-Web2021/blob/main/src/DOM_BASED_XSS/bundle.js"></script> </head>
    <body>
      <h1>DOM BASED XSS DEMO</h1>
      <div id="result" >Fail</div>
      <script >
        var num = document.URL.split("num=")[1];
        document.querySelector("#result").innerHTML =
ESAPI.encoder().encodeForJavaScript(ESAPI.encoder().encodeForHTML(num));
       </script>
    </body>
</html>
```

Session Management Cheat Sheet - httponly-attribute

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#httponly-attribute

Vulnerability :

```
<?php
    if(!empty($_GET['name'])){
        $name = $_GET['name'];
        setcookie("name", $name, time()+3000);}?>
<!DOCTYPE html><html lang="fr"> <head><meta charset="utf-8" /><title>Session Management 1</title></head>
<body>
        <?php
            if (!empty($_GET['name'])) {
                echo "<h1>Bonjour ".$_GET["name"]." voici votre liste de courses</h1>
                <form type='get' action=''>
                <input type='text' name='keyword' />
                <input type='hidden' name='name' value='".$_GET['name']."' />
                <input type='submit' value='Ajouter' /></form>";
                if (!empty($_GET['keyword'])){
                    echo "<br>Vous avez ajouté : ".$_GET['keyword']; }
            } else {
                echo "<h1>Veuillez entrer votre nom</h1>
                <form type='get' action=''>
                <input type='text' name='name'/>
                <input type='submit' value='Valider'/>
                </form>"; }?> </body></html>
```

Exploit à mettre dans le deuxième input qui s'affiche après avoir entré son nom :

```
<script>
  if(document.cookie == "") {
    alert("Aucun cookie httpOnly activé");
  } else {
    alert("Vous venez de vous faire voler vos cookies :\n" + document.cookie + "\nDommage...");
  }
</script>
```

Defense:

```
<?php
    if(!empty($_GET['name'])){
        $name = $_GET['name'];
        setcookie("name", $name, time()+3000, null, null, false, true); }?>
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /> <title>Session Management 1</title> </head>
    <body>
        <?php
            if (!empty($_GET['name'])){
                echo "<h1>Bonjour ".$_GET["name"]." voici votre liste de courses</h1>
                <form type='get' action=''>
                <input type='text' name='keyword' />
                <input type='hidden' name='name' value='".$_GET['name']."' />
                <input type='submit' value='Ajouter' /></form>";
                if (!empty($_GET['keyword'])){
                    echo "<br>Vous avez ajouté : ".$_GET['keyword']; }    }
            else {
                echo "<h1>Veuillez entrer votre nom</h1>
                <form type='get' action=''>
                <input type='text' name='name'/>
                <input type='submit' value='Valider'/>
                </form>"; } ?> </body></html>
```

Session Management Cheat Sheet - samesite-attribute

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#samesite-attribute

Vulnerability le problème est qu'il est nécessaire d'avoir un certificat approuvé pour avoir comme scheme HTTPS, et donc tester l'attribut same-site :

```
<?php
    header('Access-Control-Allow-Origin: *');
    header('Access-Control-Allow-Methods: GET');
    header('Access-Control-Allow-Headers: *');
    header('Access-Control-Allow-Credentials: true');
    if(!empty($_GET['name'])){ $name = $_GET['name'];
        // Attribut SameSite => None
       $options = array ('expires' => time() + 300000, 'path' => '', 'domain' => '', 'secure' => true,
       'httponly' => true, 'samesite' => 'None');
        setcookie("name", $name, $options);}?>
<!DOCTYPE html><html lang="fr"> <head><meta charset="utf-8" />
 <title>Session Management - Attribut SameSite (Vulnerability)</title> </head> <body>
        <?php
            if (!empty($_GET['name'])) {
                echo "<h1>Bonjour ".$_GET["name"]." voici votre liste de courses</h1>
                <form type='get' action=''>
                <input type='text' name='keyword' />
                <input type='hidden' name='name' value='".$_GET['name']."' />
                <input type='submit' value='Ajouter' /></form>";
                if (!empty($_GET['keyword'])){
                    echo "<br>Vous avez ajouté : ".$_GET['keyword']; }
            } else {
                echo "<h1>Veuillez entrer votre nom</h1>
                <form type='get' action=''>
                <input type='text' name='name'/>
                <input type='submit' value='Valider'/>
                </form>";}?></body></html>
```

Exploit : Nous n'avons pas pu tester, le but ici est de pouvoir envoyer une requête qui contiendra les cookie si SameSite est à None.
<img src='https://Invalid-URL.png' onerror="fetch('https://localhost:8000', { credentials: 'include' })">

Defense: Ici nous ne devrions pas pouvoir les envoyer car SameSite est à Strict

```
<?php
    header('Access-Control-Allow-Origin: *');
    header('Access-Control-Allow-Methods: GET');
    header('Access-Control-Allow-Headers: *');
    header('Access-Control-Allow-Credentials: true');
    if(!empty($_GET['name'])){ $name = $_GET['name'];
        // Attribut SameSite => Strict
        $options = array ('expires' => time() + 300000, 'path' => '', 'domain' => '',
        'secure' => true, 'httponly' => true, 'samesite' => 'Strict');
        setcookie("name", $name, $options);}?>
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" />
<title>Session Management - Attribut SameSite (Defense)</title> </head> <body>
        <?php
            if (!empty($_GET['name'])) {
                echo "<h1>Bonjour ".$_GET["name"]." voici votre liste de courses</h1>
                <form type='get' action=''>
                <input type='text' name='keyword' />
                <input type='hidden' name='name' value='".$_GET['name']."' />
                <input type='submit' value='Ajouter' /></form>";
                if (!empty($_GET['keyword'])){
                    echo "<br>Vous avez ajouté : ".$_GET['keyword']; } }
            else {
                echo "<h1>Veuillez entrer votre nom</h1>
                <form type='get' action=''>
                <input type='text' name='name'/>
                <input type='submit' value='Valider'/>
                </form>"; } ?> </body></html>
```

Session Management Cheat Sheet - secure-attribute

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#secure-attribute

Vulnerability :

```php
<?php
    // On démarre une nouvelle session
    $sess_name = session_name();
    if (session_start()) {
        // Attribut Secure => false, HttpOnly => true
        setcookie($sess_name, session_id(), null, '/', null, false, true);  }
    $id_session = session_id();     // id de session?>
<!DOCTYPE html><html> <head> <title>Session Management - Attribut Secure</title><meta charset="utf-8">
</head>
    <body>
        <h1>Service d'authentification</h1>
        <?php
            if($id_session){
                echo "Votre ID de session est : ".$id_session."<br><br>
                L'attribut secure est sur false, une personne malintentionnée
                pourrait capturer les cookies en écoutant le réseau avec wireshark par exemple"; } ?>
    </body></html>
```

Exploit : Capture d'écran de wireshark

```
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7\r\n
Cookie: PHPSESSID=ksdc4248g5ahndu507jjmi1rr0\r\n
   Cookie pair: PHPSESSID=ksdc4248g5ahndu507jjmi1rr0
```

Defense:

```php
<?php
    // On démarre une nouvelle session
    $sess_name = session_name();
    if (session_start()) {
        // Attribut Secure => true, HttpOnly => true
        setcookie($sess_name, session_id(), null, '/', null, true, true);}
    // id de session
    $id_session = session_id();?>

<!DOCTYPE html><html> <head> <title>Session Management - Attribut Secure</title> <meta charset="utf-8">
</head>

    <body>
        <h1>Service d'authentification</h1>
        <?php
            if($id_session){
                echo "Votre ID de session est : ".$id_session."<br><br>
                L'attribut secure est sur true, une personne malintentionnée
                ne peut donc pas capturer les cookies si vous êtes sur https";
            }
        ?>
    </body>
</html>
```

XML External Entity Prevention Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /> <title>XXE</title> </head> <body>
        <h1>Votre base de données</h1>
        <h3>Veuillez ajouter des données au format XML (Exemple : &ltname&gtJean&lt/name&gt)</h3>
        <form action="" method="post">
            <textarea style="width: 50%; height: 4%;" rows="10" cols="33" maxlength="1000"
name="data"></textarea><br>
            <input type="submit" value="Ajouter" name="submit" />
        </form><br><br>
        <?php
            error_reporting(E_ERROR | E_PARSE);
            if (isset($_POST["submit"]) && !empty($_POST["data"])) {$xmlfile = $_POST["data"];
                $dom = new DOMDocument(); $dom->loadXML($xmlfile); $contenu = simplexml_import_dom($dom);
                echo 'Vous avez ajouté :<pre>'; print_r($contenu); echo '</pre>';
                if ($contenu == "") {echo "XML non valide"; }} ?>
    </body>
</html>
```

Exploit :

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM
"https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/Common-Credentials/best15.txt">]>
<produit>&xxe;</produit>

OU ALORS UN FICHIER LOCAL :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:/nameFile"> ]>
<produit>&xxe;</produit>
```

Defense:

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /> <title>XXE</title> </head> <body>
        <h1>Votre base de données</h1>
        <h3>Veuillez ajouter des données au format XML (Exemple : &ltname&gtJean&lt/name&gt)</h3>
        <form action="" method="post">
            <textarea style="width: 50%; height: 4%;" rows="10" cols="33" maxlength="1000" name="data">
            </textarea><br>
            <input type="submit" value="Ajouter" name="submit" />
        </form><br><br>
        <?php
            error_reporting(E_ERROR | E_PARSE);
            if (isset($_POST["submit"]) && !empty($_POST["data"])) {
                $xmlfile = $_POST["data"];
                libxml_disable_entity_loader(true);
                $dom = new DOMDocument();
               $dom->loadXML($xmlfile, LIBXML_NOENT | LIBXML_DTDLOAD);
                $contenu = simplexml_import_dom($dom);
                echo 'Vous avez ajouté :<pre>';
                print_r($contenu);
                echo '</pre>';
                if ($contenu == "") {
                    echo "XML non valide";
                }}?>
    </body>
</html>
```

SQL Injection Prevention Cheat Sheet

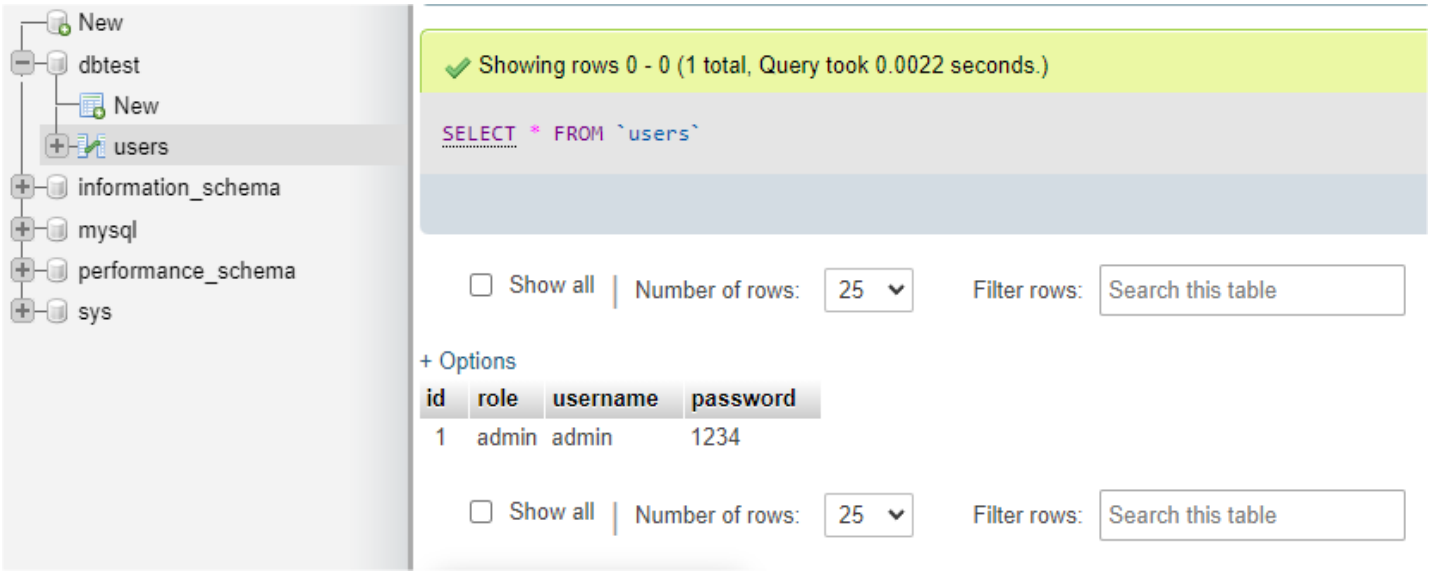
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /><title>SQL Injection</title> </head>   <body>
        <h1>Service d'authentification</h1>
        <form action="" method="post">
            Pseudo :    <input type="text" name="uid" > <br><br>
            Mot de passe :  <input type="password" name="pwd" > <br><br>
            <button type="submit" name="submit">Log In</button>
        </form>
        <?php
            if (isset($_POST["submit"]) && !empty($_POST["uid"]) && !empty($_POST["pwd"])) {
                // Récupération des données des champs de connection
                $username = $_POST["uid"]; $pwd = $_POST["pwd"];
                // Connection à la BD
                $mysqli = new mysqli("localhost", "root", "", "dbTest");
                // Vérification de la connection
                if ($mysqli -> connect_errno) {
                    echo "Failed to connect to MySQL: " . $mysqli -> connect_error; exit();}
                // Requête SQL
                $sql = "SELECT id, role FROM users WHERE username='$username' AND password='$pwd'";
                echo "<br>Requête SQL : ".$sql."<br>";
                // On exécute la requête
                if ($result = $mysqli->query($sql)) {
                    // Si résultat supérieur à 0, alors identifiant correct
                    if ($result->num_rows > 0) {
                        while ($row = $result->fetch_assoc()) { echo "<br>Identifiant correct, vous êtes
désormais connecté en tant que ".$row["role"].".";}}else { echo "<br>Mauvais identifiant, veuillez
réessayer.";}
                    $result->close(); }}?></body></html>
```

Exploit :
Dans Pseudo, taper : **' OR 1=1 OR 1='**
Dans Mot de passe, taper un mot de passe au hasard.

Pour réaliser cette attaque, il faut avoir créer dans MySQL une base de donnée “dbtest” contenant une table “users” avec les champs “id”, “role”, “username”, et “password”. Vérifier l’username et le mot de passe pour se connecter à la base de données en cas de problème.



Defense:

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /> <title>SQL Injection</title> </head> <body>
        <h1>Service d'authentification</h1>
        <form action="" method="post">
            Pseudo :    <input type="text" name="uid" > <br><br>
            Mot de passe :  <input type="password" name="pwd" > <br><br>
            <button type="submit" name="submit">Log In</button> </form>
        <?php
            if (isset($_POST["submit"]) && !empty($_POST["uid"]) && !empty($_POST["pwd"])) {
                // Récupération des données des champs de connection
                $username = $_POST["uid"];$pwd = $_POST["pwd"];
                // Connection à la BD
                $mysqli = new mysqli("localhost", "root", "", "dbTest");
                // Vérification de la connection
                if ($mysqli -> connect_errno) {echo "Failed to connect to MySQL: " . $mysqli ->
connect_error;
                    exit(); }
                // On prépare la requête SQL
                $stmt = $mysqli->prepare("SELECT id, role FROM users WHERE username=? AND password=?");
                $stmt->bind_param("ss", $username, $pwd);
                $stmt->execute();// On exécute la requête
                // On obtient les résultat de la requête
                $res = $stmt->get_result(); $row = $res->fetch_assoc();
                if ($res->num_rows > 0) {// Si résultat supérieur à 0, alors identifiant correct
                    echo "<br>Identifiant correct, vous êtes désormais connecté en tant que
".$row["role"].".";}
                else {echo "<br>Mauvais identifiant, veuillez réessayer.";}
                $res->close(); }?> </body> </html>
```

Password Storage Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Vulnerability :

```
<!DOCTYPE html><html lang="fr"> <head> <meta charset="utf-8" /> <title>Password Storage</title>    </head>
    <body>
        <h1>Inscription</h1>
        <form action="" method="post">
            Pseudo :    <input type="text" name="uid" > <br><br>
            Mot de passe :  <input type="password" name="pwd" > <br><br>
            <button type="submit" name="submit">Valider</button>
        </form>
        <br>
        <?php
            if (isset($_POST["submit"]) && !empty($_POST["uid"]) && !empty($_POST["pwd"])) {
                $username = $_POST["uid"];
                $password = $_POST["pwd"];
                echo "Votre mot de passe à été stocké de cette manière : ".hash("sha256", $password);  }  ?>
    </body></html>
```

Exploit : script python à exécuter avec un fichier dictionnaire.txt (fournit dans l'archive)

```
import hashlib

passwordFile = "dictionnaire.txt"      ## Fichier contenant les mots de passe
targetHash = "828e48a1fa87e36637809921d2b280b197efdbb643919d02b25d9b5466fb4e1d"     ## Hash a cracker

def main() :
    with open(passwordFile) as fp:
        for password in fp :
            password = password.replace("\n", "")
            print(password, " -> ", hashlib.sha256(password.encode()).hexdigest())
            if hashlib.sha256(password.encode()).hexdigest() == targetHash :
                print("Mot de passe trouvé : " + password)
                fp.close()
                exit()
    print("Aucun mot de passe trouvé !")

if __name__ == "__main__" :
    main()
```

Defense:

```
<!DOCTYPE html><html lang="fr">  <head>  <meta charset="utf-8" />  <title>Password Storage</title>  </head>
    <body>
        <h1>Inscription</h1>
        <form action="" method="post">
            Pseudo :    <input type="text" name="uid" > <br><br>
            Mot de passe :  <input type="password" name="pwd" > <br><br>
            <button type="submit" name="submit">Valider</button>
        </form>
        <br>
        <?php
            if (isset($_POST["submit"]) && !empty($_POST["uid"]) && !empty($_POST["pwd"])) {
                $username = $_POST["uid"];
                $password = $_POST["pwd"];
                echo "Votre mot de passe à été stocké de cette manière : ".password_hash($password,
PASSWORD_ARGON2I);  }?> </body></html>
```