Activité 2 : Vulnérabilités liées à l'authentification et à la gestion des sessions

ÉLÉMENTS DE CORRECTION

I Rappel des objectifs à atteindre

Comprendre les problématiques de codage sécurisé. L'activité commence par une description des principales vulnérabilités liées à ce deuxième volet du top 10 d'OWASP. Des bonnes pratiques de codages sont présentées en tant que contre-mesure. Coté démonstrations, trois défis sont à relever sur la plateforme Mutillidae via le logiciel BurpSuite :

- une énumération des logins ;
- une force brute des mots de passe sur un login découvert ;
- l'exploitation d'un cookie d'identification prévisible afin de se loguer avec le compte d'un autre utilisateur.

Attendu : chaque étudiant doit rédiger une documentation en expliquant les manipulations réalisées et faire des captures d'écrans.

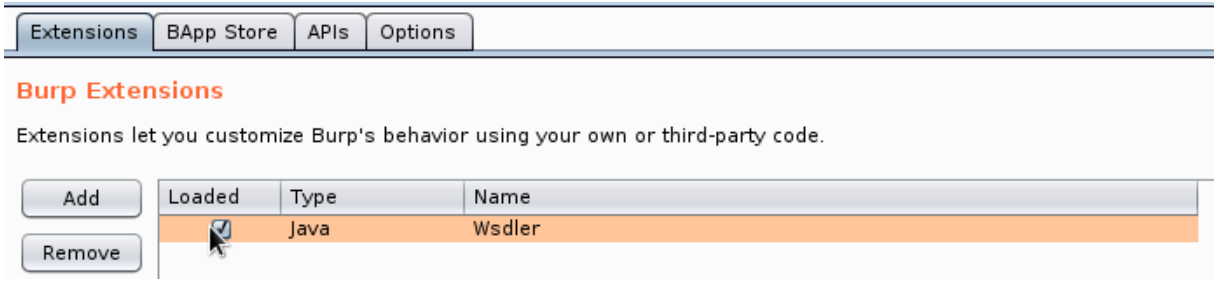
II Premier défi : Énumération des logins

Travail à faire 1 *Énumération des logins via un codage non sécurisé.*

Le but de cette première série de questions est d'étudier le comportement de l'application via un codage non sécurisé afin de lancer l'attaque visant à énumérer des logins valides.

Q1. Commencer par installer l'extension Wsdler en suivant les manipulations décrites dans l'étape n°1. Puis, positionner le niveau de sécurité à 0.

Attendu : Preuve de l'installation de l'extension dans la documentation.



Q2. Tester un exemple de requête et de réponse à l'aide d'un login non valide en suivant les manipulations décrites dans l'étape n°2 (parse de la page wsdl, envoi au répéteur, génération de la réponse et envoi au comparateur).

*Attendu : lorsque les manipulations sont terminées, les onglets **répéteur** et **comparateur** contiennent le résultat des travaux demandés (voir captures d'écrans dans la documentation). Il faut être attentif aux réponses fournies par le répéteur. Attention à ne pas fermer ces onglets lorsque la question est terminée. Tout l'intérêt de BurpSuite est de pouvoir conserver les résultats de plusieurs requêtes afin de pouvoir les exploiter par la suite.*

Q3. Tester un exemple de requête et de réponse à l'aide d'un login valide en suivant les manipulations décrites dans l'étape n°3 (parse de la page wsdl, envoi au répéteur, modification avec un login valide, génération de la réponse et envoi au comparateur).

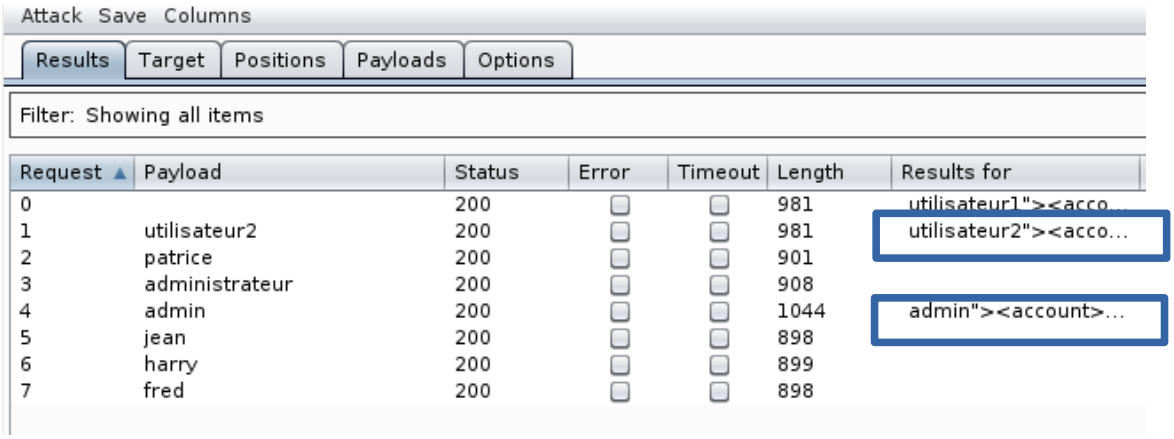
Attendu : les manipulations demandées sont presque identiques à celles de la question précédente. Il faut modifier le login proposé par défaut (gero et) par un login valide, c'est à dire un login dont on est certain qu'il correspond à un compte existant. Cela peut être le login du compte standard de l'attaquant dans le cas d'une attaque interne à l'entreprise.

Q4. Créer un dictionnaire de login sur votre machine cliente. Pour cela, ouvrir un éditeur de texte et saisir des logins les uns en dessous des autres et enregistrer votre fichier.

Attendu : les étudiants doivent disposer d'un dictionnaire de logins qu'ils peuvent créer à l'aide de l'éditeur de texte de leur choix (vim, nano). Les logins doivent s'empiler les uns en dessous des autres. L'enregistrement du fichier peut se faire dans le répertoire personnel de l'utilisateur utilisant la machine cliente dotée du logiciel BurpSuite.

Q5. Lancer l'énumération et relever les logins valides en suivant les manipulations décrites dans l'étape n°4.

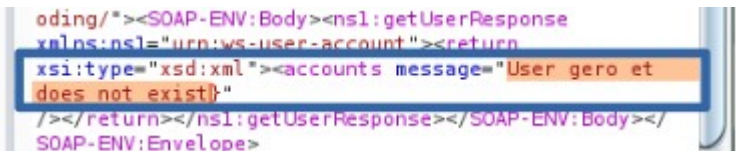
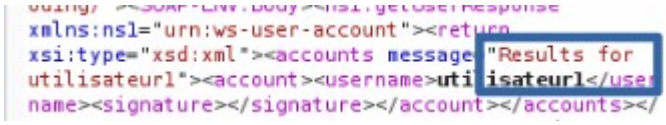
*Attendu : C'est à ce moment qu'il faut lancer l'attaque. Le résultat doit faire apparaître une liste de logins pour lesquels la colonne **Results for** est renseignée. Cette liste correspond au résultat de l'énumération.*



*Le login de la première ligne (utilisateur1) est celui associé à notre test pour l'obtention d'une réponse de succès. Il y a donc deux autres logins valides dans le résultat de notre énumération : **utilisateur2** et **admin**.*

Q6. *A l'aide du comparateur, expliquer quelles sont les lignes de la réponse sur lesquelles l'attaquant a pu s'appuyer pour lancer l'attaque ?*

*Attendu : C'est la chaîne de caractère **Résults for** qui a servi de filtre au lancement de l'attaque. En cas d'échec, le message est différent.*

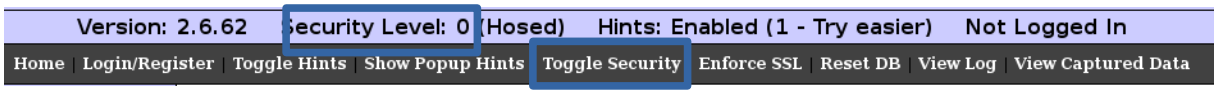
Login incorrect	
Login correct	

Travail à faire 2 Codage sécurisé et analyse du code source

Le but de cette deuxième partie est de tester à nouveau l'attaque après activation du codage sécurisé et de comprendre l'encodage mis en place.

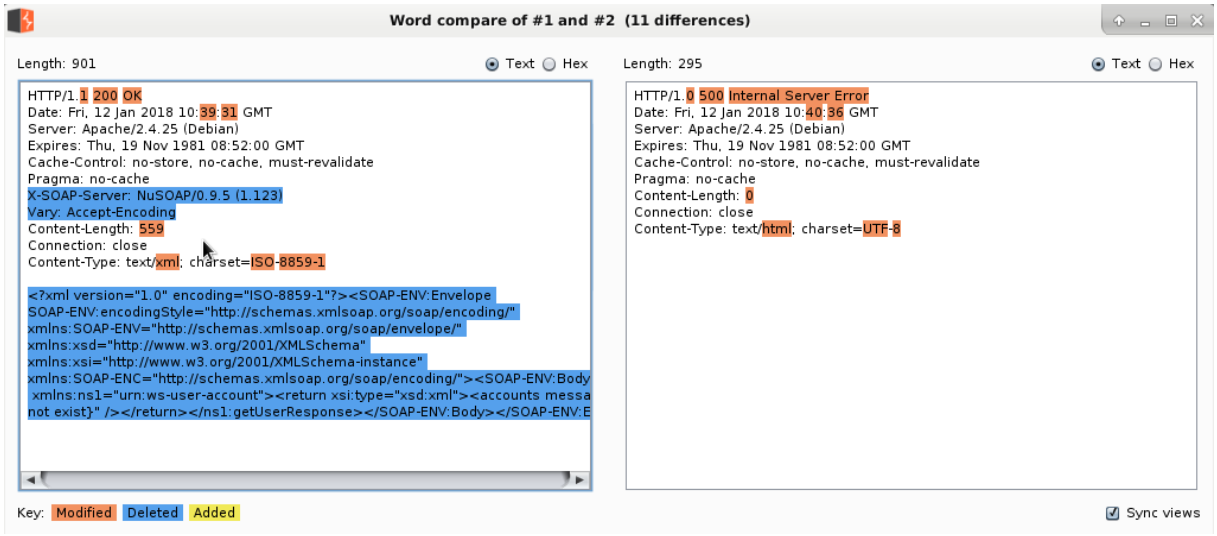
Q1. Fermer puis relancer BurpSuite. Positionner le niveau de sécurité à 5 et relancer l'attaque en suivant les étapes 1 à 5.

Attendu : Relance de l'attaque en ayant adapté le niveau de sécurité en cliquant sur le bouton Toogle security



Q2. Les informations affichées par le comparateur sont-elles exploitables pour tenter une énumération ?

Attendu : Voici ce qu'on obtient au niveau du comparateur :



Le résultat n'est pas exploitable et l'attaque ne peut pas aboutir.

Q3. *Chercher dans le code source de la page ws-user-account.php (située dans /var/www/html/mutillidae/webservices/soap/) le codage mis en place permettant d'obtenir un encodage sécurisé. Expliquer le rôle de l'instruction EncodeforHTML.*

Attendu : La page ws-user-account.php nous montre les éléments suivants :

*Une fonction nommée **doXMLEncodeQueryResults** est appelée. Cette fonction travaille sur la variable **EncodeOutput** initialisée à la valeur VRAI si le codage est sécurisé.*

```
case "2":
case "3":
case "4":
case "5": // This code is fairly secure
        $lEncodeOutput = TRUE;
        break;
```

Cette fonction encode la sortie à l'aide de l'instruction EncodeForHTML. Cela explique pourquoi les sorties du comparateur ne sont pas exploitables.

```
        function doXMLEncodeQueryResults($pUsername, $pQueryResult, $pEncodeOutp
ut){

                $lResults = "<accounts message=\"Results for {$pUsername}\">";
                $lUsername = "";
                $lSignature = "";

                while($row = $pQueryResult->fetch_object()){

                        $pEncodeOutput?$lSignature = $lUsername = $Encoder->enco
deForHTML($row->username):$lUsername = $row->username;;

                        if(isset($row->mysignature)){
                                $pEncodeOutput?$lSignature = $Encoder->encodeFor
HTML($row->mysignature):$lSignature = $row->mysignature;
                        }// end if
--Plus--(63%)
```

EncodeForHTML encode une chaîne d'entrée pour une sortie HTML sécurisée afin d'empêcher l'exploitation des messages de retour (rendus HTML indésirables). L'encodage permet de changer de format de représentation des données. L'utilisation d'un encodage pour traiter les retours HTML est une bonne pratique de codage.

III Deuxième défi : Force brute sur un mot de passe

Travail à faire 3 Force brute d'un mot de passe.

Dans un premier temps, l'objectif est de se placer coté attaquant en lançant une force brute pour découvrir le mot de passe d'un compte.

Q1. Commencer par préparer l'attaque en suivant les manipulations décrites dans l'étape n°1.

Attendu : Captures d'écrans de l'interception de la connexion avec le compte admin. Cette première partie ne présente pas de difficultés particulières. Les captures d'écrans du dossier documentaire font office de correction.

Q2. Lancer la force brute en suivant les manipulations décrites dans l'étape n°2.

Attendu : Captures d'écrans montrant au minimum :

- *la création du dictionnaire de mots de passe ;*
- *le positionnement de l'attaque en mode sniper avec une variable associée au mot de passe ;*
- *le résultat de l'attaque avec le code de status 302 lorsque le mot de passe est trouvé.*

Travail à faire 4 Codage sécurisé et analyse du code source

Le but de cette deuxième partie est de tester à nouveau l'attaque après activation du codage sécurisé et de comprendre les contre-mesures permettant de limiter ou de contrer l'attaque.

Q1. Fermer puis relancer BurpSuite. Positionner le niveau de sécurité à 5 et relancer l'attaque en suivant les étapes 1 et 2. La force brute a t-elle échouée ?

Attendu : L'attaque est encore un succès.

Q2. Observez le code source de la page *register.php* (création d'un nouveau compte) et indiquer si avec un codage de niveau 5, un utilisateur peut créer un compte avec un mot de passe non sécurisé ?

Attendu : Rien n'interdit un utilisateur de créer un compte avec un mot de passe non sécurisé (password, admin7…). Le niveau de sécurité 5 sur cette page correspond à des contrôles permettant d'éviter d'autres types d'attaques (script, injections sql…). Pour limiter ce type d'attaque, le

développeur doit interdire la création de comptes avec des mots passe non sécurisés en ajoutant des fonctions qui :

- *testent les mots de passe choisis au moment de leur saisie en indiquant leur caractère sécurisé ou non. Un message rappelant la définition d'un mot de passe sécurisé doit s'afficher.*
- *empêcher le choix de mots de passe non sécurisés.*

Q3. Coté administrateur système, quelles sont les mesures permettant de détecter et de contrer ce type d'attaque ?

Attendu : Cette question fait le lien avec l'option SISR. Pour détecter l'attaque, un IDS (système de détection des intrusions) peut être configuré. Un IPS (système de prévention des intrusion) peut bloquer les attaques de type force brute. Des outils comme fail2ban (voir le coté labo sur l'installation et la sécurisation d'ownCloud) exploitent les journaux de logs afin de bannir les adresses IP à l'origine de ces attaques.

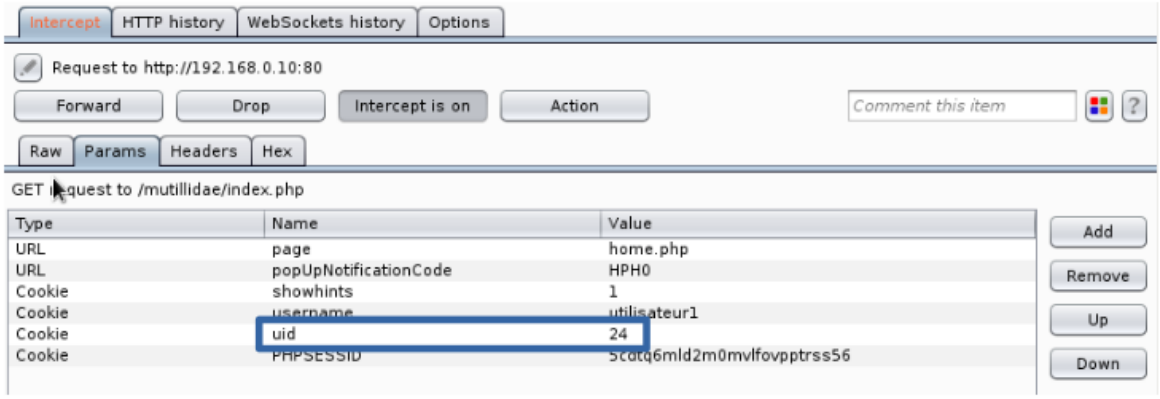
IV Troisième défi : Vol de session

Travail à faire 5 Vol de session.

Dans un premier temps, l'objectif est de se placer coté attaquant en volant la session d'une victime.

Q1. Commencer par intercepter le cookie d'un utilisateur correctement authentifié en suivant les manipulations décrites dans l'étape n°1.

*Attendu : Capture d'écran montrant le cookie nommé **uid**.*



Q2. Voler la session d'une victime en modifiant l'identifiant associé au cookie intercepté. Pour cela, suivre les manipulations décrites dans l'étape n°2.

Attendu : Après modification de la valeur du cookie et clic sur le bouton Forward du proxy, il faut se retrouver connecté en tant que qu'admin.

Travail à faire 6 Codage sécurisé et analyse du code source

Le but de cette deuxième partie est de tester à nouveau l'attaque après activation du codage sécurisé et de comprendre les contre-mesures permettant de contrer l'attaque.

Q1. Fermer puis relancer BurpSuite. Positionner le niveau de sécurité à 5 et relancer l'attaque en suivant les étapes 1 et 2. La modification du cookie *uid* a t-elle une conséquence ?

Attendu : L'attaque échoue. La modification du cookie uid n'a pas de conséquences. Visiblement ce cookie ne peut plus être exploité avec un niveau de sécurité 5.

Q2. Observez le code source de la page *index.php* (page d'accueil) et relever les différence avec le codage de niveau de sécurité 0 et 1.

Attendu : La page index.php comprend un bloc nommé REACT TO CLIENT SIDE ATTACK. Lorsqu'on se situe en niveau de sécurité 5, on ne se fie plus aux données « client » pour valider l'authentification mais on utilise des sessions encodées.

```
                case "2":
                case "3":
                case "4":
                case "5": // This code is fairly secure
                        /* If we are secure, then we do not rely on any client i
nput
                         * to make authorization decisions. Authorization tokens
 should
                         * never be stored on the client. We use the SESSION in
secure mode.
                         *
                         * Also, when we create an HTTP header, we encode any ou
tput to
                         * prevent response splitting. The critical chars in res
ponse splitting
                         * are CR-LF. Dont fall for filtering. Just encode it al
--Plus--(53%)_
```

```
l.
                         */
                        header('Logged-In-User: '.$Encoder->encodeForHTML($_SESS
ION['logged_in_user']), TRUE);
                break;
        }// end switch
        /* ----------------------------------------
     * END REACT TO CLIENT SIDE CHANGES
     * ---------------------------------------- */
```

En modifiant le cookie uid, on reste connecté sous le même utilisateur qu'avant la modification.

Q3. *Expliquer les différences entre un cookie et une session. Conclure sur les bonnes pratiques de codage concernant le suivi des utilisateurs identifiés.*

Attendu : Les données d'un cookie sont stockées coté client. Si ce dernier est utilisé comme identifiant prévisible des utilisateurs alors il peut être modifié et envoyé au serveur avec la modification. Avec les sessions, les données résident sur le serveur et leur durée de vie est limitée. L'encodage des sorties permet de brouiller les pistes d'une éventuelle exploitation malveillante.