

Sécurisation des applications Web

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

Propriétés	Description
Intitulé long	Exploitation d'une plateforme d'apprentissage des vulnérabilités des applications web
Intitulé court	Sécurisation des applications web
Formation concernée	BTS Services Informatiques aux Organisations
Matière	SLAM4 : Réalisation et maintenance de composants logiciels.
Présentation	<p>Ce Côté labo a pour objectif d'exploiter la plateforme d'apprentissage Mutillidae (OWASP) afin de se familiariser avec les principales vulnérabilités des applications web.</p> <p>Chaque activité couvre une problématique spécifique (SQLi, XSS, CSRF...) en référence au top 10 des vulnérabilités décrites par l'OWASP.</p> <p>Dans un premier temps, l'étudiant doit réaliser les attaques associées à chaque vulnérabilité.</p> <p>Dans un deuxième temps, l'objectif est d'analyser et de comprendre les codes sources des scripts présentés dans leur forme non sécurisée puis sécurisée en tant que contre-mesure.</p> <p>Cette deuxième activité traite des problématiques d'authentification et de gestion des sessions.</p>
Notions	<p>Activités supports de l'acquisition des compétences</p> <p>D4.1 – Maintenance d'une solution applicative</p> <ul style="list-style-type: none">• A4.2.1 Analyse et correction d'un dysfonctionnement, d'un problème de qualité de service ou de sécurité. <p>Savoir-faire</p> <ul style="list-style-type: none">• Programmer un composant logiciel.• Adapter un composant logiciel.• Valider et documenter un composant logiciel. <p>Savoirs associés</p> <ul style="list-style-type: none">• Techniques de sécurisation.
Prérequis	Commandes de base d'administration d'un système Linux, langages PHP et JavaScript. Avoir lu la présentation et réalisé les installations nécessaires à l'activité 1.
Outils	Deux machines éventuellement virtualisées sont nécessaires avec Linux comme système d'exploitation.
	Site officiel : https://www.owasp.org
Mots-clés	OWASP, Mutillidae, BurpSuite, vulnérabilités, SQLi, XSS, IDOR.
Durée	Une heure pour cette activité.
Auteur(es)	Patrice DIGNAN, avec la relecture, les tests et les suggestions de Hervé Le Guern et de Yann BARROT.
Version	v 1.0
Date de publication	juillet 2018

I Présentation générale.....	2
1 Le risque A2 du top 10 d'OWASP.....	2
2 Conséquences.....	2
3 Bonnes pratiques.....	3
II Objectifs et architecture générale de l'activité.....	3
III Premier défi : énumération des logins.....	5
1 Objectif.....	5
2 A vous de jouer.....	5
3 Bonnes pratiques.....	6
II Deuxième défi : Force brute sur un mot de passe.....	7
1 Objectif.....	7
2 A vous de jouer.....	7
3 Bonnes pratiques.....	7
III Troisième défi : Vol de session.....	8
1 Objectif.....	8
2 A vous de jouer.....	8
3 Bonnes pratiques.....	8
IV Conclusion.....	9

I Présentation générale

1 Le risque A2 du top 10 d'OWASP

Lors du développement d'une application, le codage des fonctions liées à l'authentification et à la gestion des sessions (cookie de session) peuvent être incorrectement implémentés, permettant ainsi à des attaquants de compromettre des mots de passe et des identifiants de session.

2 Conséquences

En cas de force brute sur des mots de passe ou de vol d'identifiant de session (session hijacking), une personne malveillante peut s'identifier avec le compte d'un autre utilisateur voire avec celui de l'administrateur. Les conséquences peuvent être particulièrement graves sur une application manipulant des données hautement confidentielles (applications médicales, bancaires...). Par ailleurs, le **Règlement général sur la protection des données** (RGPD) confère à la CNIL des missions supplémentaires et un pouvoir de contrôle et de sanction accru en matière de protection des données ce qui renforce l'obligation des entreprises d'assurer la sécurité des données manipulées.

Si le codage des fonctions liées à l'authentification et à la gestion des sessions est mal implémenté, l'application web risque d'offrir **les vulnérabilités suivantes** :

1. **Tests d'authentification possibles sur des listes de login et de mots de passe : énumération des logins valides puis force brute des mots de passe ;**
2. Identification à l'aide de mots de passe par défaut encore actifs lors de la phase de déploiement de l'application : certaines applications web comportent des comptes avec des mots de passe par défaut (glpi/glpi pour l'application GLPI ou nagios/nagiosadmin pour l'application nagios ou admin/cisco sur un routeur Cisco WRV215 , etc.) ;
3. Création autorisée de comptes utilisateurs avec des mots de passe non sécurisés tels que admin ou password1 ;
4. Codage non sécurisé des fonctionnalités permettant à un utilisateur de retrouver son mot de passe en cas d'oubli ;
5. Mots de passe écrits en dur dans du code source, mots de passe non chiffrés ou faiblement hashés (absence de fonction de salage) ;
6. Absence ou mauvais codage des fonctions gérant les authentifications multi-formes pour les applications très sensibles en terme de confidentialité des informations ;
7. **Mauvaise implémentation des cookies de session : cookie d'identifiant de session prévisible**, exposition des sessions ID dans l'URL, absence de rotation des sessions ID après un succès d'authentification ou après une déconnexion, pas de timeout sur les sessions ID.

Côté mise en pratique, cette deuxième activité exploite quelques exemples des vulnérabilités décrites dans les points 1. et 7.

3 Bonnes pratiques

Les bonnes pratiques suivantes peuvent être mises en place en tant que limitations ou contre-mesures des vulnérabilités présentées en amont :

1. Le développeur ne doit pas indiquer la raison d'un échec d'authentification : login incorrect ou mot de passe incorrect. Il faut simplement indiquer qu'il y a un échec d'authentification sans donner plus de détails par une phrase du type : échec d'authentification ;
2. Il faut coder des fonctions qui imposent un changement de mot de passe lors de la première connexion et supprimer les comptes inutiles comportant des mots de passe par défaut lors du déploiement de l'application ;
3. Il faut interdire les mots de passe non sécurisés (mots de passe du dictionnaire) en testant la solidité des mots de passe au moment de la création des comptes (codage qui impose une longueur minimale, la présence de caractères spéciaux...) ;
4. Il faut s'assurer que les fonctions permettant de retrouver un mot de passe en cas d'oubli ne présentent pas un codage trop laxiste (demande d'une couleur préférée par exemple) ;
5. Externaliser le stockage des mots de passe et les stocker sous forme chiffrée : ne pas stocker des mots de passe en clair dans du code source, utiliser des fonctions de salage lorsque les mots de passe sont hashés afin de prévenir les attaques du type table arc-en-ciel (rainbow table¹) ;
6. Les applications manipulant des informations hautement confidentielles doivent comporter des modules d'authentifications multi-formes en plus du traditionnel login/mot de passe : possession d'un objet pour déchiffrer un contenu, biométrie, géolocalisation... ;
7. Générer des cookies d'identifiant de sessions non prévisibles, qui changent après un succès d'authentification, les désactiver après une déconnexion et programmer une durée de validité (timeout).

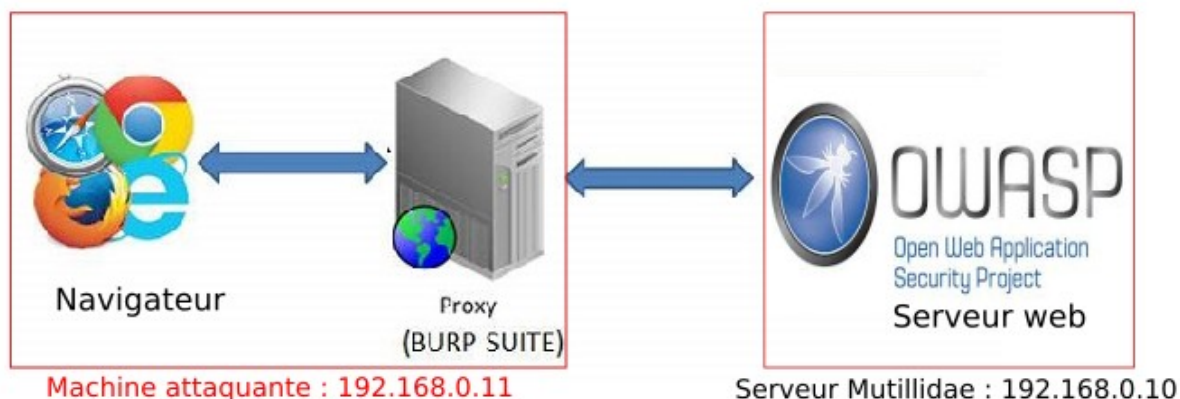
II Objectifs et architecture générale de l'activité

Trois défis sont proposés pour illustrer la sécurisation de l'authentification et des sessions :

- l'énumération des logins : il s'agit d'énumérer des logins valides à partir d'un dictionnaire ;
- une attaque par force brute sur un login valide : l'attaquant étant certain qu'un login est valide, il peut tenter une force brute sur le mot de passe ;
- un vol de session à l'aide d'un cookie d'identification prévisible afin de s'identifier à l'aide du compte d'un autre utilisateur sans connaître son login et son mot de passe.

Ces trois défis peuvent être réalisés de manière indépendante. Chaque défi est associé à un dossier documentaire.

Pour rappel, l'environnement de travail est le suivant :



1 Rainbow table : attaque permettant de cracker l'empreinte (hash) d'un mot de passe.


Le serveur Mutillidae propose un site web conçu pour identifier et tester les failles de sécurité identifiées par l'OWASP. Il est possible pour chacune d'entre elles, de définir le niveau de sécurité appliqué.

Notre démarche consistera, pour les failles de type A2 d'OWASP :

- à partir de la version non sécurisée de la page concernée et à mettre en évidence la faille de sécurité.
- nous constaterons ensuite que dans la version sécurisée de cette page fournie par Mutillidae, l'attaque n'est plus possible.
- l'étude des mécanismes de sécurisation utilisés, donc du code de la page associée, permettra de dégager des bonnes pratiques de programmation.

Quant à la machine attaquante, elle comprend un navigateur ainsi que le proxy BurpSuite qui permet d'intercepter les requêtes avant de les envoyer au serveur.

III Premier défi : énumération des logins

 **Remarque importante** : attention à ne pas utiliser la **version 2.6.60** de Mutillidae pour la réalisation de ce défi suite à la constatation d'un bug.

1 Objectif

L'objectif est d'obtenir une liste de logins valides testés à l'aide d'un dictionnaire. Lorsque le développeur indique la raison d'un échec d'authentification (login incorrect), un attaquant peut profiter de ces messages d'échec afin de tester des listes de login en comparant les réponses obtenues entre un succès et un échec d'authentification.

L'énumération des logins est envisageable, quel que soit le type d'interaction entre les clients et le serveur : nous aurions pu appliquer la démarche décrite plus bas pour une authentification "classique" basée sur des pages HTML/PHP ; nous avons choisi de travailler sur une authentification passant par un **service web** basé sur un **protocole de communication de type SOAP**.

Les services web sont de plus en plus utilisés. Ils facilitent la communication entre applications hétérogènes : ils servent beaucoup pour interconnecter les systèmes d'informations ; on les retrouve aussi dans les services mis à disposition par un cloud.

Mutillidae permet d'aborder un certain nombre de problèmes de sécurité posés par ces services web : l'énumération des logins est donc le premier que nous rencontrerons.

Pour aller plus loin sur les services web :

OWASP propose une page dédiée aux services web et aux problèmes de sécurités associés :
https://www.owasp.org/index.php/Web_Services.

[Les services Web](#) sur *Openclassrooms*

2 A vous de jouer

Les questions suivantes peuvent être traitées en suivant les étapes décrites dans le dossier documentaire n°1 (page 10, énumération des logins).

Travail à faire 1 Énumération des logins en mode non sécurisé.

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

- Q1.** Commencer par installer l'extension Wsdler en réalisant les manipulations décrites dans l'étape n°1. Puis, positionner le niveau de sécurité à 0.
- Q2.** Tester un exemple de requête et de réponse à l'aide d'un login non valide en réalisant les manipulations décrites dans l'étape n°2 (parse de la page wsdl, envoi au répéteur, génération de la réponse et envoi au comparateur).
- Q3.** Tester un exemple de requête et de réponse à l'aide d'un login valide en réalisant 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).
- 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.
- Q5.** Lancer l'énumération et relever les logins valides en réalisant les manipulations décrites dans l'étape n°4.
- 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 ?

Travail à faire 2 Énumération des logins en mode 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 2 à 4.
- Q2.** Les informations affichées par le comparateur sont-elles exploitables pour tenter une énumération ?
- 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*.

3 Bonnes pratiques

Les bonnes pratiques de codage permettant de limiter ou d'éviter ce type d'attaque sont les suivantes :

- ne pas indiquer la cause d'un échec d'authentification mais se limiter à un affichage indiquant l'échec d'authentification sans donner plus de détails (login incorrect ou mot de passe incorrect) ;
- encoder les messages de sortie pour éviter qu'un attaquant puisse les exploiter.

II Deuxième défi : Force brute sur un mot de passe

1 Objectif

Réaliser une force brute du mot de passe associé au compte administrateur précédemment découvert (login admin).

2 A vous de jouer

Les questions suivantes se traitent en suivant les étapes décrites dans le dossier documentaire n°2 (page 18, force brute d'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 côté attaquant en lançant une force brute pour découvrir le mot de passe d'un compte.

- Q1.** Commencer par préparer l'attaque en réalisant les manipulations décrites dans l'étape n°1.
- Q2.** Lancer la force brute en suivant les indications de l'étape n°2.

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 échoué ?
- Q2.** Observez le code source de la page *register.php* (création d'un nouveau compte) et indiquez si avec un codage de niveau 5, un utilisateur peut créer un compte avec un mot de passe non sécurisé ?
- Q3.** Côté administrateur du système attaqué, quelles sont les mesures permettant de détecter et de contrer ce type d'attaque ?

3 Bonnes pratiques

Les bonnes pratiques suivantes permettant de limiter ou d'éviter ce type d'attaque :

- limitation de l'attaque : le développeur doit rajouter des fonctions permettant de tester la sécurité d'un mot de passe au moment de la création d'un compte en empêchant l'utilisation d'un mot de passe non sécurisé ;
- empêcher l'attaque : côté administrateur système, les systèmes de prévention des intrusions peuvent bloquer les attaques de type force brute.

III Troisième défi : Vol de session

1 Objectif

Se retrouver authentifié avec le compte d'un autre utilisateur via un cookie d'identification prévisible sans connaître le login et le mot de passe de la victime.

2 A vous de jouer

Les questions suivantes se traitent en suivant les étapes décrites dans le dossier documentaire n°3 (page 21, vol de session).

Travail à faire 5 Vol de session.

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

- Q1.** Commencer par intercepter le cookie d'un utilisateur correctement authentifié en réalisant les manipulations décrites dans l'étape n°1.
- Q2.** Voler la session d'une victime en modifiant l'identifiant associé au cookie intercepté. Pour cela, réaliser les manipulations décrites dans l'étape n°2.

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 ?
- Q2.** Observez le code source de la page *index.php* (page d'accueil) et relever les différences avec le codage de niveau de sécurité 0 et 1.
- Q3.** Expliquer les différences entre un cookie et une session. Conclure sur les bonnes pratiques de codage concernant le suivi des utilisateurs identifiés.

3 Bonnes pratiques

Les bonnes pratiques permettant d'éviter ce type d'attaque sont les suivantes :

- utiliser des sessions avec des valeurs générées et non prévisibles ;
- utiliser des ID de sessions qui sont modifiés après un succès d'authentification, désactivés après une déconnexion et qui ont une durée de vie limitée (timeout) ;
- encoder les ID de sessions.

IV Conclusion

Aperçu général des mesures de défense

En résumé, les principales mesures de défense concernant les problématiques d'authentification de gestion des sessions sont les suivantes :

Authentification

- Ne pas révéler des messages d'erreur ou de succès trop explicites ;
- Ne jamais inscrire, dans du code source, des mots de passe non chiffrés ou pas assez chiffrés, externaliser le stockage des mots de passe ;
- Renforcer la politique de gestion des mots de passe (durée de vie, longueur, caractères spéciaux, majuscules).

Gestion des sessions

- Générer des jetons de sessions non prévisibles ;
- Programmer des règles d'expiration et de rotation des ID de sessions (après un succès d'authentification ou une déconnexion) ;
- Utiliser des fonctions permettant d'encoder les identifiants de session.

Dossier documentaire

Dossier 1 : Énumération des logins



Remarque importante : attention à ne pas utiliser la **version 2.6.60** de Mutillidae pour la réalisation de ce défi suite à la constatation d'un bug.

La démarche permettant de réaliser l'attaque est la suivante :

1. Dans un premier temps, l'attaquant va observer le code renvoyé par le serveur suite à un échec d'authentification.
2. L'étape précédente est répétée avec un login existant : pour cela, il peut utiliser son propre compte standard ;
3. L'attaquant peut alors comparer les différences sur les codes de retour obtenus afin de relever un extrait pertinent qu'il pourra exploiter comme filtre pour réaliser son attaque ;
4. Enfin, il ne reste plus qu'à utiliser un dictionnaire comportant des logins à tester en utilisant le filtre précédemment repéré. L'ensemble des logins valides obtenus correspond au résultat de notre énumération.

Étape n°0 (préalable) : Installation de l'extension Wsdler

Dans un premier temps, il faut enrichir BurpSuite d'une extension nommée Wsdler.

Cette extension intercepte les requêtes WSDL et les opérations associées au service web cible. Il est alors possible de générer des requêtes SOAP qui pourront être envoyées au service web.

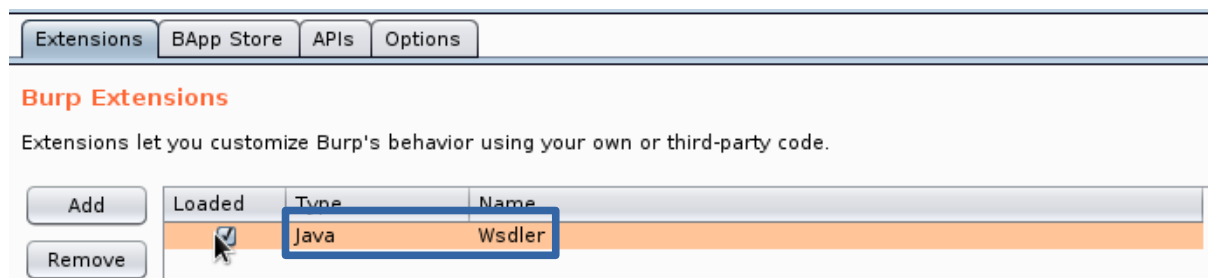
L'extension est décrite plus en détail par son développeur sur son site : <https://blog.netspi.com/hacking-web-services-with-burp/>

Pour commencer, lancer BurpSuite, aller dans l'onglet **Extender** puis dans **BApp Store**. Sélectionner l'extension **Wsdler** et l'installer. Le bouton **Installer** est situé en bas de la fenêtre de droite.

The screenshot shows the Burp Suite Community Edition v1.7.29 interface. The 'BApp Store' tab is active, displaying a list of extensions. The 'Wsdler' extension is highlighted. The details for 'Wsdler' are shown on the right, including its author (Eric Gruber), version (2.0.12), and update date (01 nov. 2016). The extension is described as 'essentially the WSDL parsing portion of Soap-UI without the UI.' and 'Requires Java version 8'. There is a 'Submit rating' button and a 'Reinstall' button.

Name	Insta...	Rating	Popu...	Last upd...	Detail
Software Version Re...		☆☆☆☆	—	12 déc. ...	Requires ...
Software Vulnerabilit...		☆☆☆☆	—	17 juil. 2...	Requires ...
SpyDir		☆☆☆☆	—	08 févr. ...	
SQLiPy SQLmap Inte...		☆☆☆☆	—	15 déc. ...	
Swagger Parser		☆☆☆☆	—	18 août ...	
ThreadFix		☆☆☆☆	—	25 janv. ...	Requires ...
TokenJar		☆☆☆☆	—	25 janv. ...	
UUID Detector		☆☆☆☆	—	23 févr. ...	
WAFDetect		☆☆☆☆	—	08 févr. ...	
Wayback Machine		☆☆☆☆	—	25 mai 2...	
WCF Deserializer		☆☆☆☆	—	15 juin 2...	
Web Cache Deceptio...		☆☆☆☆	—	23 nov. ...	Requires ...
WebInspect Connector		☆☆☆☆	—	10 août ...	Requires ...
WebSphere Portlet S...		☆☆☆☆	—	17 févr. ...	
What-The-WAF		☆☆☆☆	—	02 oct. 2...	
Wordlist Extractor		☆☆☆☆	—	20 avr. ...	
WSDL Wizard		☆☆☆☆	—	01 juil. 2...	
Wsdler	✓	☆☆☆☆	—	01 nov. ...	
XChromelLogger Dec...		☆☆☆☆	—	25 janv. ...	

Une fois l'installation terminée, vérifier que l'extension s'affiche dans l'onglet **Extensions**.



Étape n°1 : Test d'une requête et d'une réponse sur un login inexistant

Positionner le proxy à **intercept off** puis ouvrir la page suivante :

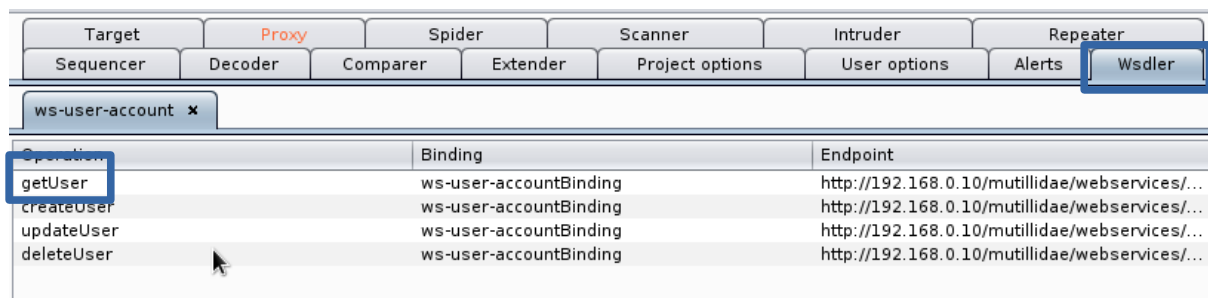
OWASP 2017 => A2 : Broken Authentication and Session Management => Username Enumeration => Lookup User (SOAP Web Service).

Positionner le proxy à **intercept on**, puis cliquer, dans le navigateur, sur le lien **View the WSDL**.

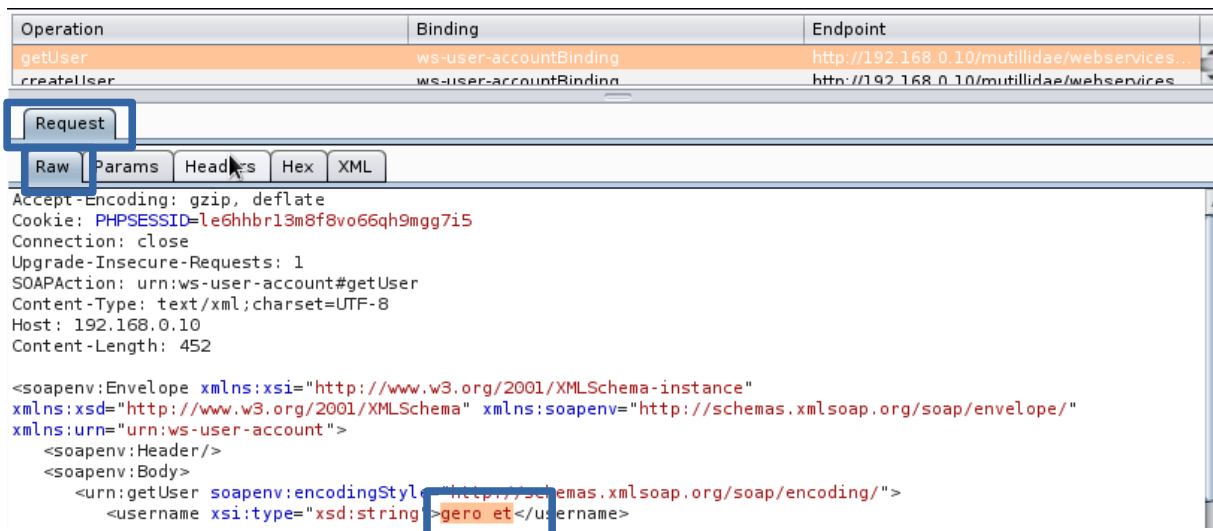


Faire un clic droit à l'intérieur de cette fenêtre de capture (Raw) en positionnant la souris dans la partie en fond blanc et cliquer sur **Parse WSDL**.

Vérifier que l'onglet **Wsdler** de BurpSuite s'enrichit des opérations suivantes :



Pour notre objectif d'énumération, c'est l'opération **getUser** qui nous intéresse. Cliquer sur **getUser** et observer le code de la requête et plus particulièrement le contenu de la balise **username**. Cette balise contient une valeur par défaut correspondant à un login qui n'existe pas dans la liste des logins valides des comptes déjà existants (gero et). Cette requête est donc idéale pour tester le comportement de notre application sur un **login inexistant**.



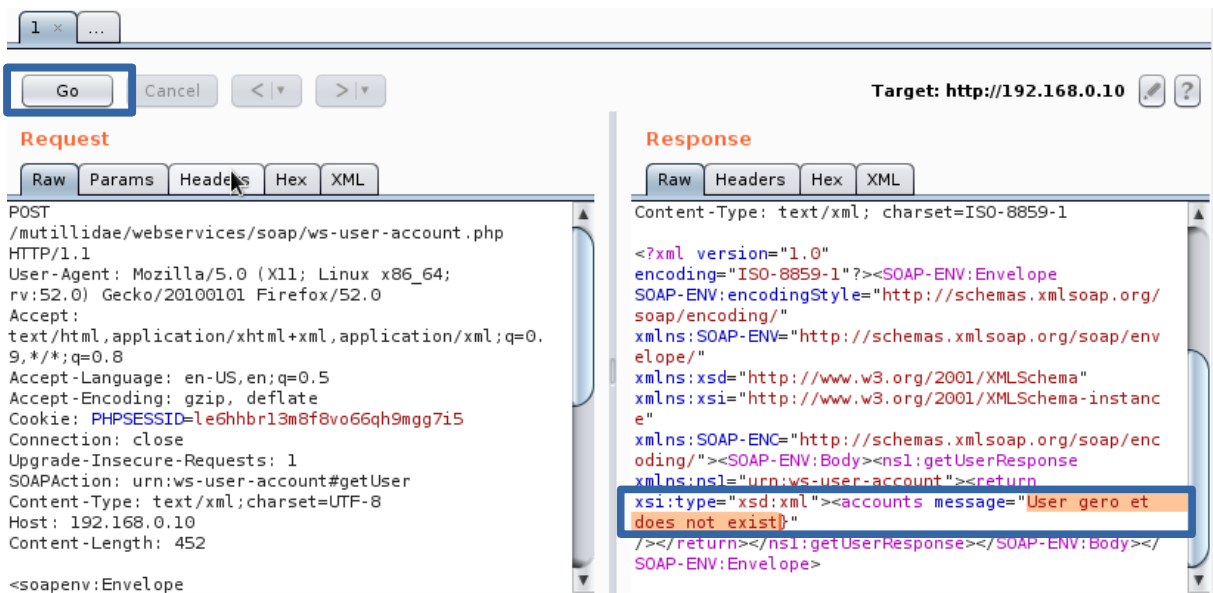
Faire un clic droit dans la fenêtre correspondant à la requête associée à un login inexistant (Raw) et cliquer sur **Send to Repeater**.

L'onglet **Repeater** de BurpSuite ajoute un premier sous onglet correspondant à notre requête.

Dans cet onglet, la partie **Request** correspond à la requête traitée et l'onglet **Raw** indique le flux capturé suite à cette requête.

Cliquer sur le bouton **Go** pour observer la réponse correspondante.

Observez le code de la réponse et plus particulièrement la phrase indiquant que l'utilisateur (login) n'existe pas (User gero et does not exist).



Faire un clic droit dans la fenêtre de la réponse (fenêtre de droite) et cliquer sur **Send to Comparer**. L'onglet **Comparer** de BurpSuite s'enrichit de notre première réponse correspondant à un login inexistant.

Comparer ?

This function lets you do a word- or byte-level comparison between different data. You can load, paste, or send data here from other tools and then select the comparison you want to perform.

Select item 1:

#	Length	Data
1	901	HTTP/1.1 200 OKD...

Étape n°2 : Test d'une requête et d'une réponse sur un login existant

Préalable : créer un nouvel utilisateur sous Mutillidae nommé **utilisateur1** en lui affectant un mot de passe. Pour cela, cliquer sur le lien **Please register here** dans la page d'authentification. Pour les besoins du TP, créer aussi un autre utilisateur nommé **utilisateur2**.

Dont have an account? [Please register here](#)

Reproduire ensuite l'ensemble des manipulations de l'étape n°2 avec un login existant. Pour cela, positionner le proxy à **intercept off** puis ouvrir la page suivante :

OWASP 2017 => A2 : Broken Authentication and Session Management => Username Enumeration => Lookup User (SOAP Web Service).

Positionner ensuite le proxy à **intercept on**, puis cliquer sur le lien **View the WSDL**.

Comme précédemment, faire un clic droit dans la fenêtre de capture du proxy et cliquer sur **Parse WSDL**. Puis, dans l'onglet **Wsdler** de BurpSuite, cliquer sur **getUser** et envoyer la requête au **répéteur** (Send to Repeater) par un clic droit.

Le répéteur de BurpSuite offre maintenant un deuxième onglet correspondant à notre nouvelle requête. C'est à ce moment là qu'il faut remplacer la valeur **gero et** par un login valide (*utilisateur1* dans la capture d'écran ci-dessous).

The screenshot shows the Burp Suite Repeater tool interface. The 'Request' tab is active, displaying a SOAP envelope with the following XML structure:

```
<soapenv:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:ws-user-account">
  <soapenv:Header/>
  <soapenv:Body>
    <urn:getUser
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <username
        xsi:type="xsd:string">utilisateur1</username>
      </urn:getUser>
    </soapenv:Body>
  </soapenv:Envelope>
```

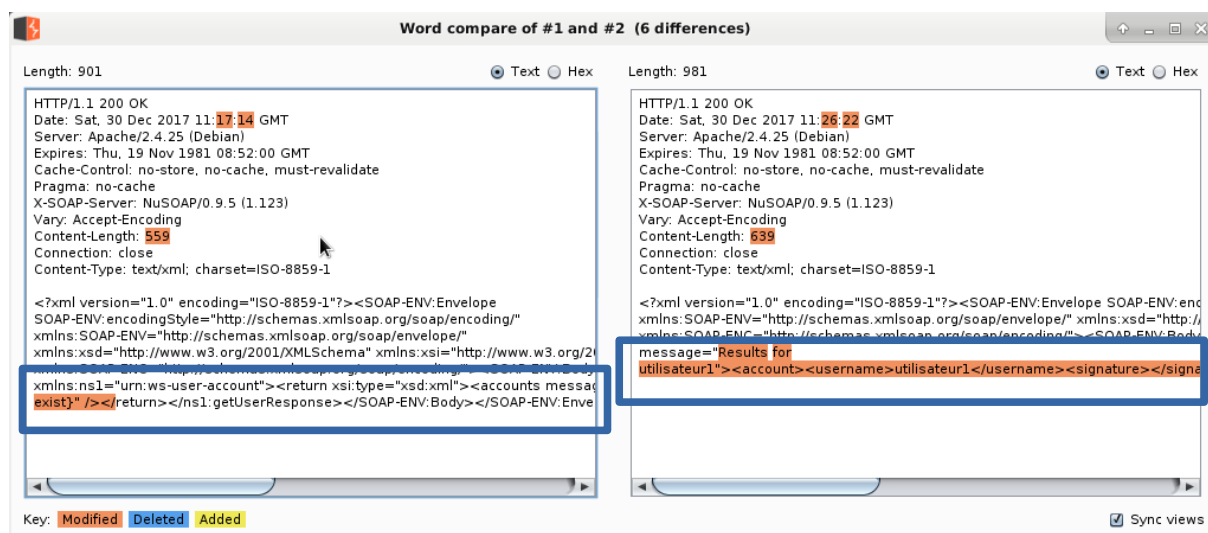
The 'Response' tab is also active, showing the following XML response:

```
<?xml version="1.0"
  encoding="ISO-8859-1"?><SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:getUserResponse
      xmlns:ns1="urn:ws-user-account">
      <return
        xsi:type="xsd:string">Results for utilisateur1</return>
    </ns1:getUserResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Il faut alors cliquer sur le bouton **Go** pour obtenir la réponse correspondant à un login valide. On observe la chaîne de caractère **Results for**. On peut alors envoyer la réponse au comparateur (Send to the Comparer) par un clic droit.

L'onglet Comparer de BurpSuite permet alors de comparer les réponses obtenues entre un login valide et un login inexistant. En cliquant sur le bouton **Words**, on peut observer les différences.

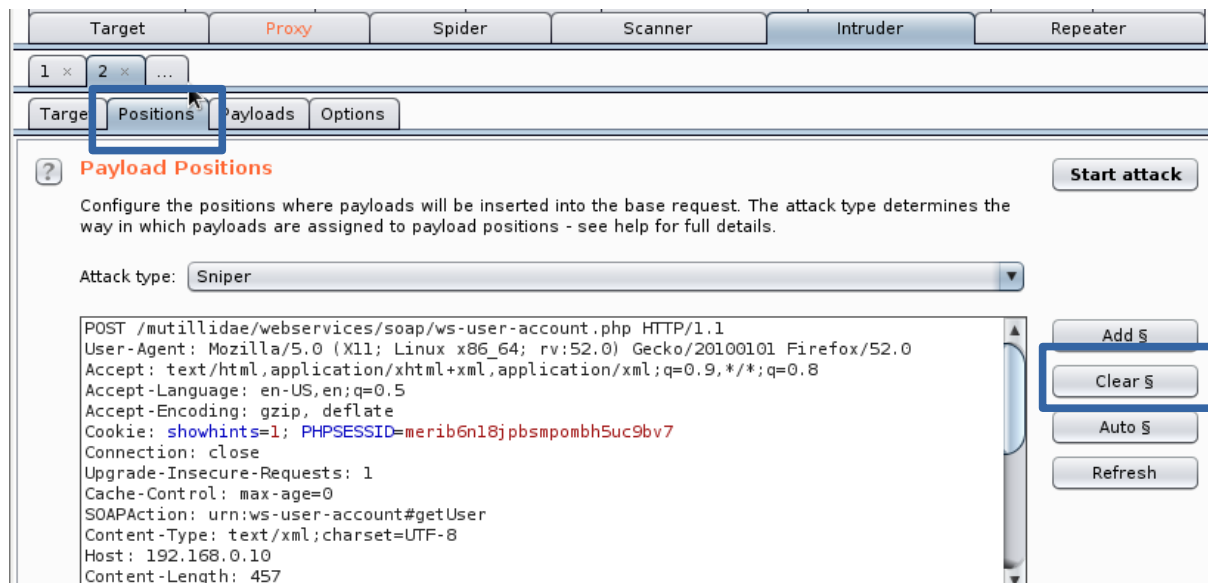
La fenêtre de gauche correspond à la réponse obtenue en cas de login inexistant. Celle de droite en cas de login existant.



C'est cette différence dans les messages qui s'affichent que nous allons exploiter.

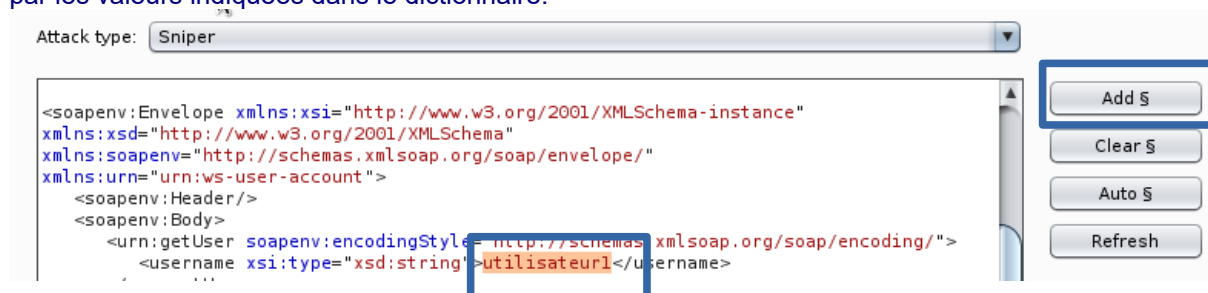
Étape n°3 : Énumération des logins

Revenir dans l'onglet Repeater de BurpSuite et dans la fenêtre de réponse (fenêtre de droite) correspondant au test sur un **login correct**, faire un clic droit et cliquer sur **Send to the Intruder**. Aller ensuite dans l'onglet **Intruder** de BurpSuite, et cliquer sur l'onglet **Positions** puis sur le bouton **Clear**.



Toujours dans cette fenêtre, il faut sélectionner avec un double clic de souris la valeur correspondant au login (*utilisateur1* dans cette capture d'écran) et cliquer sur le bouton **Add**. Nous travaillerons donc avec une seule variable, d'où le mode **Sniper**. D'autres modes existent et permettent de travailler avec plusieurs variables.

Une fois la valeur sélectionnée, Bupsuite testera en boucle différents logins en remplaçant la variable par les valeurs indiquées dans le dictionnaire.



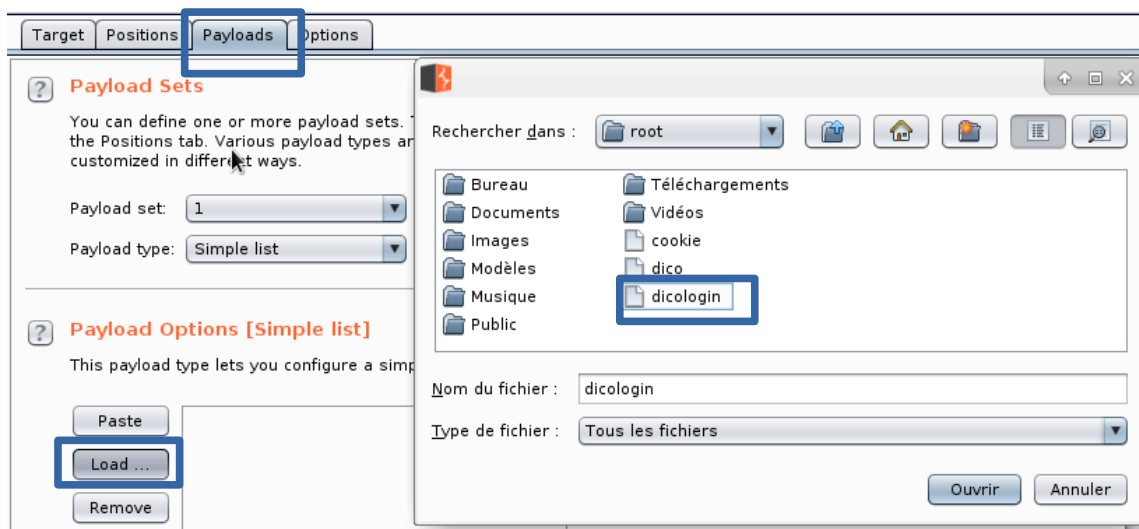
Puis, cliquer sur l'onglet **Payload** et charger un dictionnaire de login. Ce dictionnaire peut être créé à l'aide d'un éditeur de texte comportant une liste de login.

```
GNU nano 2.7.4                                Fichier : dicologin
utilisateur2
patrice
administrateur
admin
jean
harry
fred
```

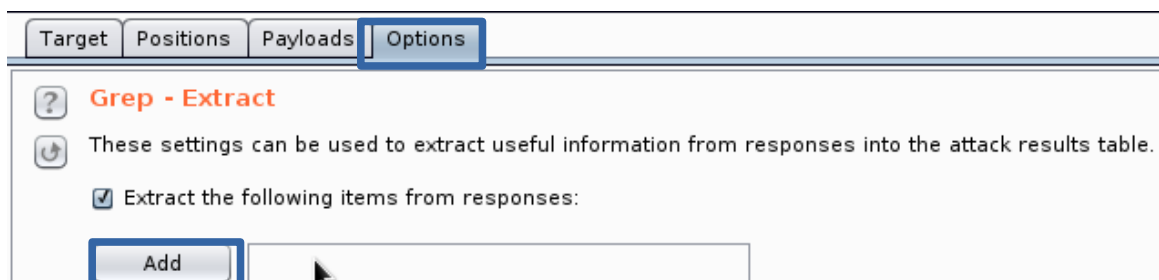


```
root@client-owasp:~# more dicologin
utilisateur2
patrice
administrateur
admin
jean
harry
fred
```

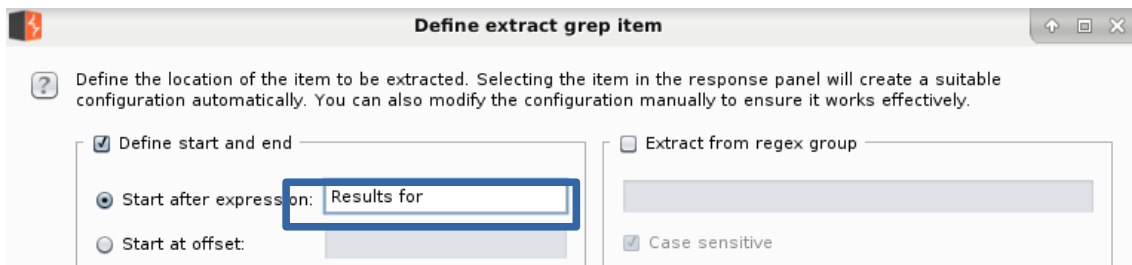
La sélection du dictionnaire se fait en cliquant sur le bouton **Load** de la rubrique **Payload options**.



Enfin, dans le dernier onglet **Options**, il faut ajouter un filtre dans la rubrique **Grep Extract**. Cliquer ensuite sur le bouton **Add**.



Dans la fenêtre suivante, il faut indiquer la chaîne de caractère correspondant à un login correct: **"Results for"**, puis valider en cliquant sur **OK**.



Il ne reste plus qu'à lancer l'attaque en cliquant sur **Start Attack** dans le menu **Intruder** de BurpSuite. A ce moment là, ne pas tenir compte du message d'avertissement sur les limitations de la version community.

La fenêtre de résultat de l'attaque correspond à notre énumération de logins.

Attack Save Columns						
Results Target Positions Payloads Options						
Filter: Showing all items						
Request ▲	Payload	Status	Error	Timeout	Length	Results for
0		200	<input type="checkbox"/>	<input type="checkbox"/>	981	utilisateur1"><acco...
1	utilisateur2	200	<input type="checkbox"/>	<input type="checkbox"/>	981	utilisateur2"><acco...
2	patrice	200	<input type="checkbox"/>	<input type="checkbox"/>	901	
3	administrateur	200	<input type="checkbox"/>	<input type="checkbox"/>	908	
4	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	1044	admin"><account>...
5	jean	200	<input type="checkbox"/>	<input type="checkbox"/>	898	
6	harry	200	<input type="checkbox"/>	<input type="checkbox"/>	899	
7	fred	200	<input type="checkbox"/>	<input type="checkbox"/>	898	

Tous les logins testés pour lesquels la colonne **Results for** est alimentée sont des logins valides sur lesquels une force brute du mot de passe peut être tentée.

Dossier 2 : Force brute d'un mot de passe

Étape n°1 : Préparation de l'attaque

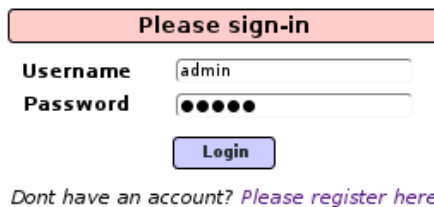
Une attaque en force brute consiste à tester en boucle des mots de passe présents dans un dictionnaire. L'outil Burpsuite teste alors chacun des mots de passe en observant le code de retour ce qui permet d'identifier un succès d'authentification.

Démarrer BurpSuite et positionner mutillidae sur le niveau de sécurité 0. Placer aussi le proxy en mode de non capture en cliquant sur **intercept off**. Ouvrir ensuite la page permettant de s'authentifier :

OWASP 2017 => A2 (Broken Authentication and Session Management) => Authentication Bypass => Via Brute Force => Login

Dans l'exemple qui suit, nous travaillons sur un compte dont le login est *admin*. Dans la version 2.6.62 de Mutillidae, un compte *admin* existe déjà avec le mot de passe *adminpass*. Les manipulations suivantes pourraient être faite avec n'importe quel compte existant dont on souhaite brute forcer le mot de passe.

Positionner le proxy BurpSuite à on (intercept on). Dans le champ **login**, saisir **admin** et dans le champ du mot passe, saisir n'importe quel mot de passe erroné, puis valider en cliquant sur le bouton **Login**. On peut saisir n'importe quel mot de passe erroné vu que l'objectif de cette étape est juste de positionner une variable.



Please sign-in

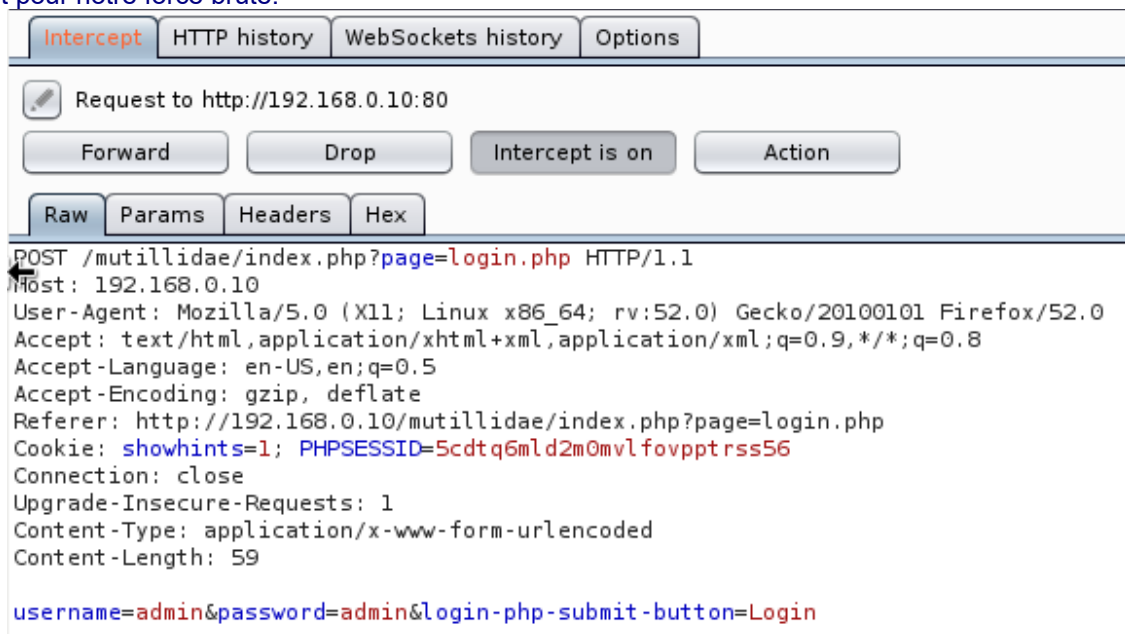
Username

Password

Login

Dont have an account? [Please register here](#)

Cliquer ensuite sur le bouton **Forward**. Dans la capture d'écran ci-dessous, c'est le mot de passe *admin* qui a été saisi. Peu importe puisque nous allons transformer ce mot de passe en variable de test pour notre force brute.



Intercept HTTP history WebSockets history Options

Request to http://192.168.0.10:80

Forward Drop Intercept is on Action

Raw Params Headers Hex

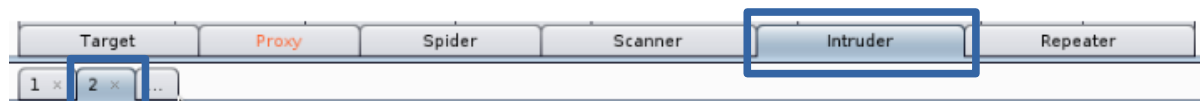
```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.0.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.10/mutillidae/index.php?page=login.php
Cookie: showhints=1; PHPSESSID=5cdtq6mld2m0mvlfovptrss56
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 59

username=admin&password=admin&login-php-submit-button=Login
```

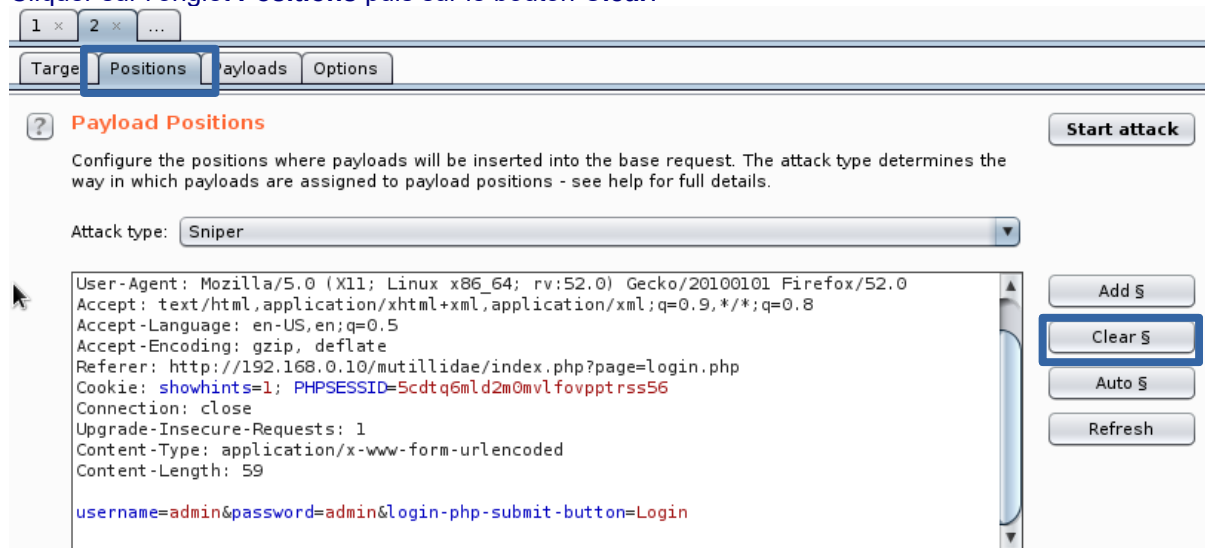
Étape n°2 : Lancement de l'attaque

Faire ensuite un clic droit à l'intérieur de cette fenêtre et cliquer sur **Send to Intruder**.

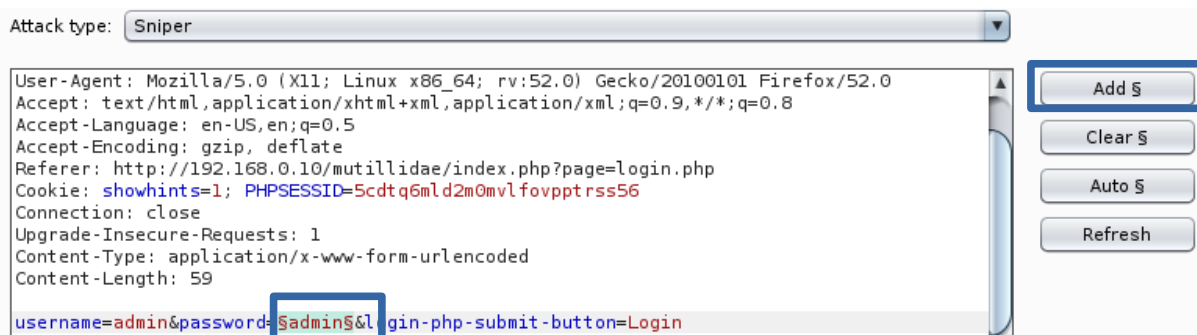
L'onglet Intruder de BurpSuite s'enrichit d'un sous onglet supplémentaire.



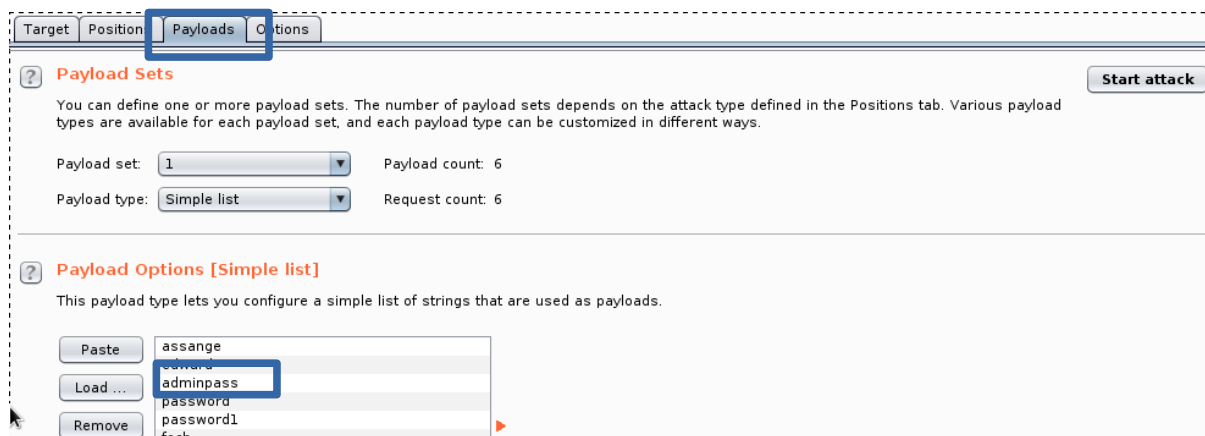
Cliquer sur l'onglet **Positions** puis sur le bouton **Clear**.



Puis, sélectionner par un double clic le mot de passe saisi (admin) et cliquer sur le bouton **Add**.



Cliquer ensuite sur l'onglet **Payload** et charger un dictionnaire en cliquant sur le bouton **Load** dans la section **Payload Options**. Il faut auparavant avoir créé ce dictionnaire.



Le lancement de l'attaque se fait en cliquant sur le bouton **Start Attack**.

Request ▲	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	55271	
1	assange	200	<input type="checkbox"/>	<input type="checkbox"/>	55271	
2	edward	200	<input type="checkbox"/>	<input type="checkbox"/>	55271	
3	adminpass	302	<input type="checkbox"/>	<input type="checkbox"/>	373	
4	password	200	<input type="checkbox"/>	<input type="checkbox"/>	55317	
5	password1	200	<input type="checkbox"/>	<input type="checkbox"/>	55317	
6	foch	200	<input type="checkbox"/>	<input type="checkbox"/>	55317	

Les lignes associées à un **code de status de 302** correspondent à un succès d'authentification. Le mot de passe du compte admin est donc *adminpass*.

Dossier 3 : Vol de session

Étape n°1 : Interception du cookie

Tout d'abord, commencer par se déconnecter de Mutillidae et positionner le proxy en non interception en cliquant sur intercept off. Redémarrer aussi le navigateur.

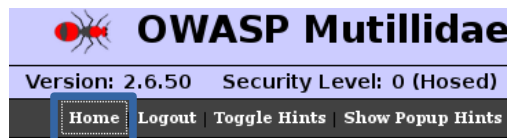
Démarrer BurpSuite et positionner Mutillidae avec le niveau de sécurité à 0. Puis ouvrir la page suivante :

OWASP 2017 => A2 : Broken Authentication and Session Management => Privilège Escalation => Login

Saisir un login et un mot de passe correspondant à un compte existant (*utilisateur1* dans cet exemple). Puis valider en cliquant sur le bouton **Login**.



Ensuite, positionner le proxy à **intercept on**. Cliquer ensuite sur le lien **Home** en haut de la page afin de naviguer en étant connecté.



Cliquer sur le bouton **Forward** du proxy et observer les cookies capturés dans l'onglet **Params** de l'intercepteur.

Type	Name	Value
URL	page	home.php
URL	popUpNotificationCode	HPH0
Cookie	showhints	1
Cookie	username	utilisateur1
Cookie	uid	24
Cookie	PHPSESSID	5cdq6mld2m0mvlfovptrrs56

Un des cookies capturé s'intitule **uid** et ressemble à une sorte de clé primaire.

On peut s'interroger sur le comportement de l'application si on rejoue la requête avec un numéro différent car après le numéro 24 on peut prévoir que l'utilisateur suivant est associé au numéro 25. De même on peut se demander à quel utilisateur est associé le numéro 1.

Étape n°2 : Vol de la session

Modifier la valeur du cookie **uid** en mettant la valeur 1. Pour cela, double cliquer sur la valeur du cookie et saisir la nouvelle valeur.

GET request to /mutillidae/index.php

Type	Name	Value
URL	page	home.php
URL	popUpNotificationCode	HPH0
Cookie	showhints	1
Cookie	username	utilisateur1
Cookie	uid	1
Cookie	PHPSESSID	5cdtq6mld2m0mvfvpptrss56

Puis cliquer sur le bouton **Forward** du proxy. On se retrouve connecté en tant qu'administrateur. Le cookie d'identification était donc prévisible.

Logged In Admin: admin (g0t r00t?)