

Sécurisation des applications Web

Mission 2A: Vulnérabilités liées à l'authentification et à la gestion des sessions – Énumération des logins

I Présentation générale.....	2
1 Le risque A2 du top 10 d'OWASP 2017.....	2
2 Conséquences.....	2
3 Bonnes pratiques.....	2
II Objectifs et architecture générale de l'activité.....	3
III Premier défi : énumération des logins.....	4
1 Objectif.....	4
2 A vous de jouer.....	4
3 Bonnes pratiques.....	5

I Présentation générale

1 Le risque A2 du top 10 d'OWASP 2017

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 implémentées, 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 et/ou poivrage) ;
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 et/ou poivrage 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).

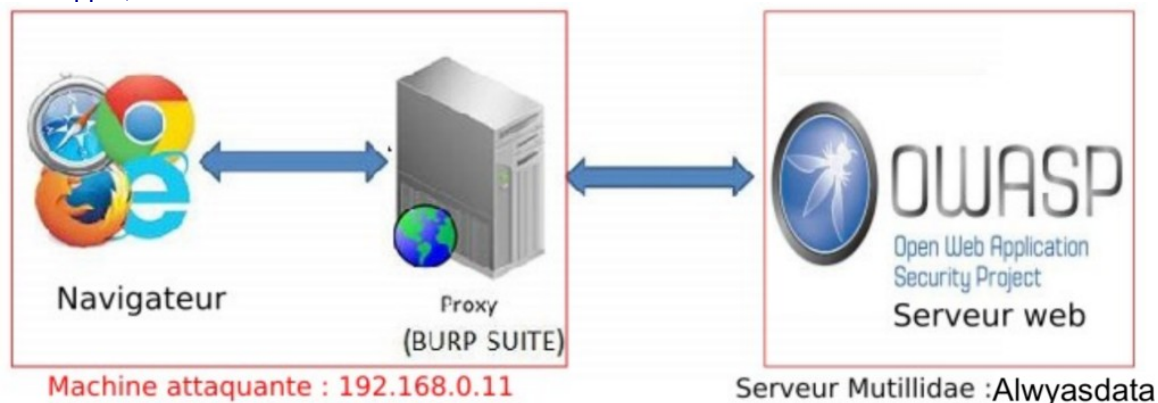
II Objectifs et architecture générale de l'activité

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 (A, B et C) peuvent être réalisés de manière indépendante. Chaque défi est associé à un dossier documentaire.

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



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 :

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

- 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.

2 A vous de jouer

Les questions suivantes peuvent être traitées en suivant les étapes décrites dans le dossier documentaire n°1 (page 6, 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.

Dossier documentaire

Dossier 1 : Énumération des logins

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.

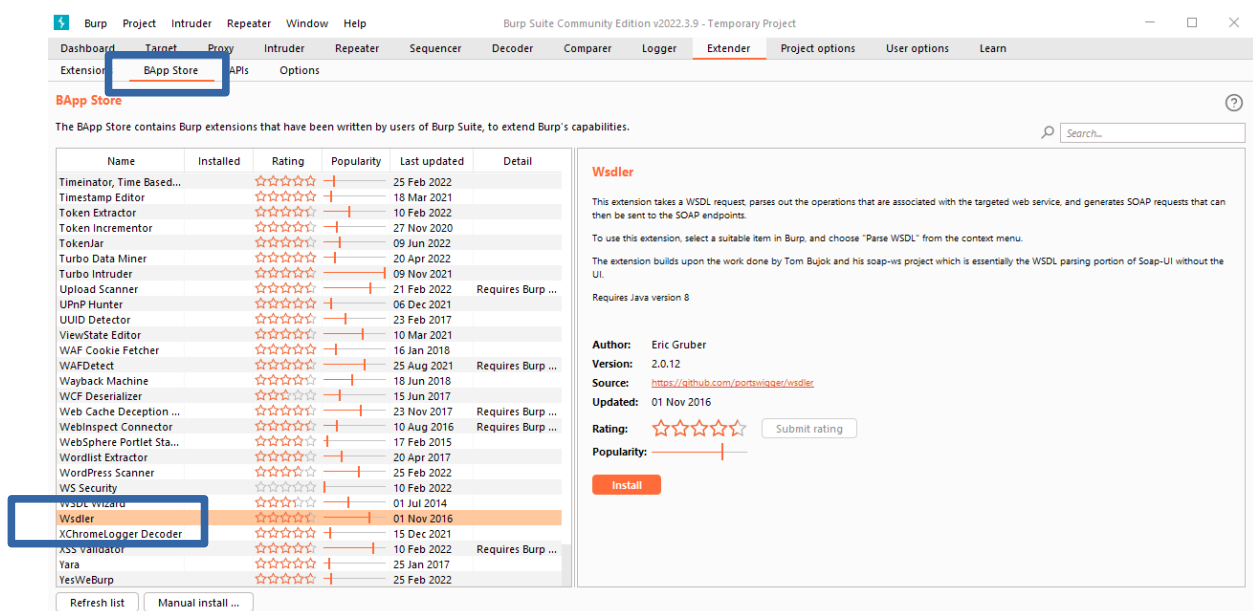
Étape n°0 (préalable) : Installation de l'extension Wsdler

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

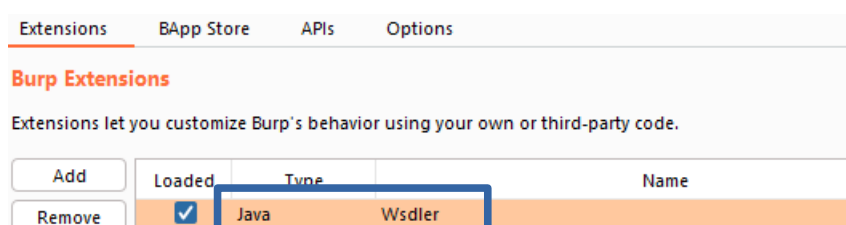
Cette extension intercepte les requêtes WSDL et les opérations associées au service web cible. Il est alors possible de générer des requêtes SOAP qui pourront être envoyées au service web.

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.



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

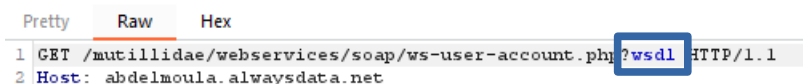


Étape n°1 : Test d'une requête et d'une réponse sur un login inexistant

Positionner le proxy à **intercept is 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**.



```
1 GET /mutillidae/webservices/soap/ws-user-account.php?wsdl HTTP/1.1
2 Host: abdelmoula.alwaysdata.net
```

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 **Extensions > Wsdler > Parse WSDL**.

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

Decoder	Comparer	Logger	Extender	Project options	User options	Learn	Wsdler
Dashboard		Target	Proxy	Intruder	Repeater	Sequencer	
ws-user-account ✕							
Operation		Binding		Endpoint			
getUser		ws-user-accountBinding		http://abdelmoula.alwaysdata.net/mutillidae/webser..			
createUser		ws-user-accountBinding		http://abdelmoula.alwaysdata.net/mutillidae/webser..			
updateUser		ws-user-accountBinding		http://abdelmoula.alwaysdata.net/mutillidae/webser..			
deleteUser		ws-user-accountBinding		http://abdelmoula.alwaysdata.net/mutillidae/webser..			

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**.



```
11 Content-Type: text/xml; charset=UTF-8
12 Host: abdelmoula.alwaysdata.net
13 Content-Length: 459
14
15 <?xml version='1.0' encoding='UTF-8' ?>
16 <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">
17 <soapenv:Header/>
18 <soapenv:Body>
19 <urn:getUser soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
20 <username xsi:type="xsd:string">gero et</username>
21 </urn:getUser>
22 </soapenv:Body>
23 </soapenv:Envelope>
```

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 **Send** 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).

The screenshot shows the Burp Suite interface with a target set to `http://abdelmoula.alwaysdata.net`. The **Request** tab is active, showing a POST request to `/mutillidae/webservices/soap/ws-user-account.php`. The **Response** tab is also visible, showing an XML response. The XML structure is as follows:

```
<?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">
        <accounts message="User gero et does not
          exist)" />
      </return>
    </ns1:getUserResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

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	Paste
1	901	HTTP/1.1 200 OKD...	Load

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](#)

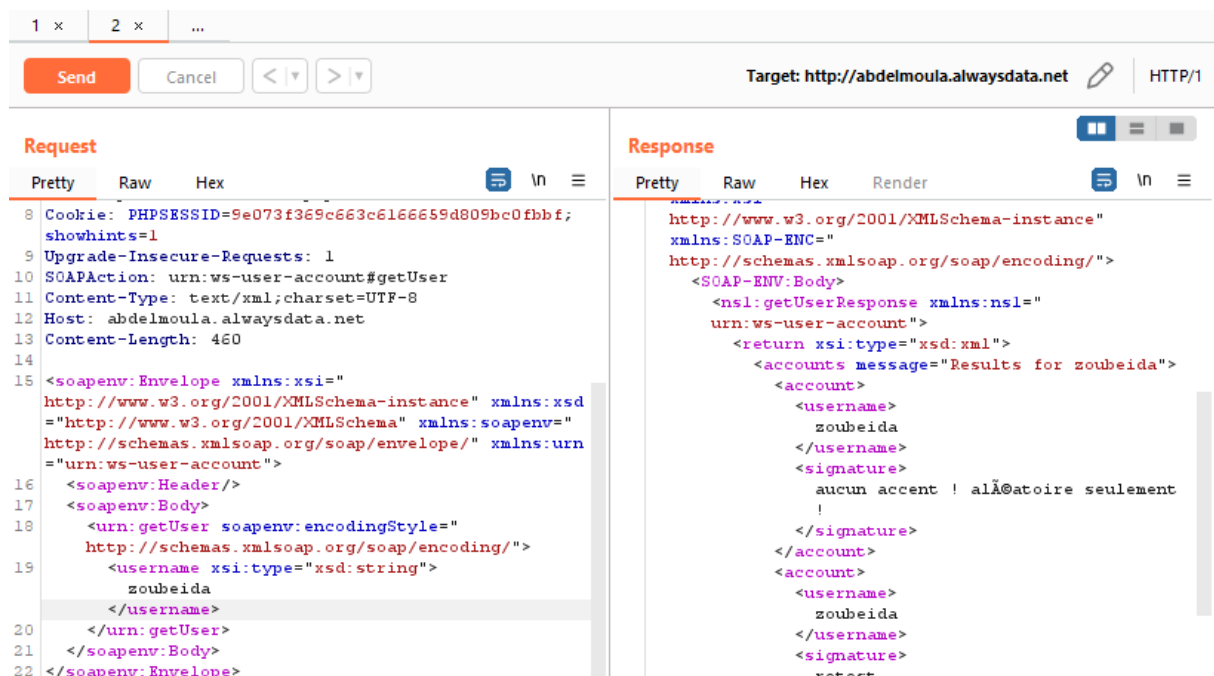
Reproduire ensuite l'ensemble des manipulations de l'étape n°2 avec un login existant. Pour cela, positionner le proxy à **intercept off** puis ouvrir la page suivante :

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

Positionner ensuite le proxy à **intercept is on**, puis cliquer sur le lien *Extensions > Wsdler > 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 (*zoubeida* dans la capture d'écran ci-dessous).



Il faut alors cliquer sur le bouton **Send** 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.

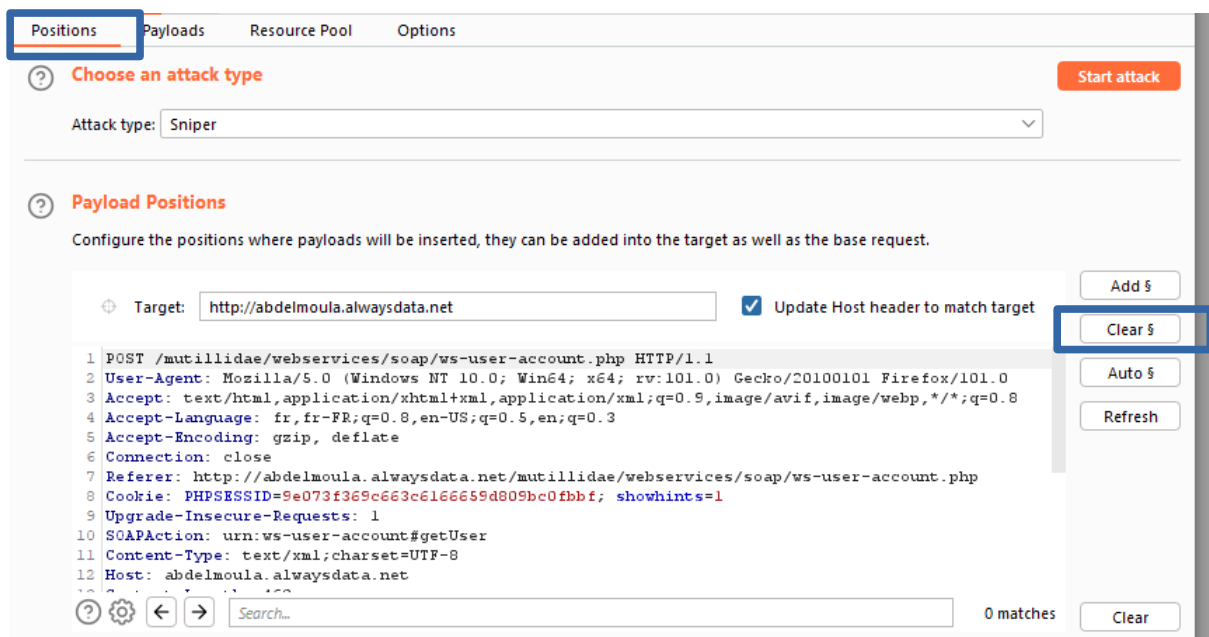
La fenêtre de gauche correspond à la réponse obtenue en cas de login inexistant. Celle de droite en cas de login existant.

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



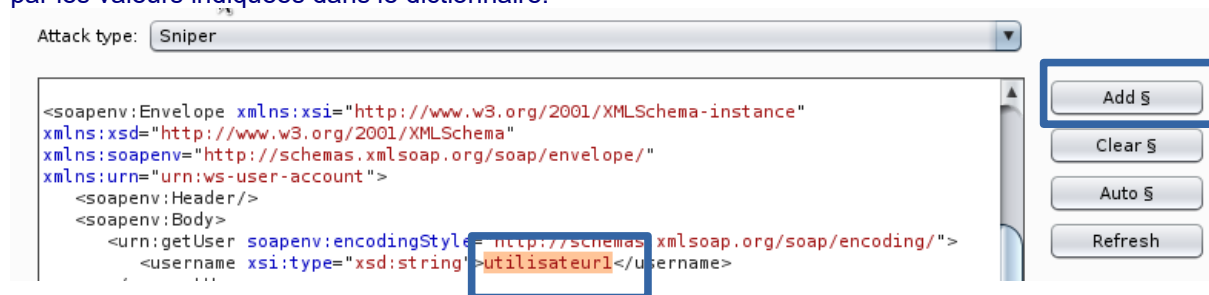
Étape n°3 : Énumération des logins

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**.

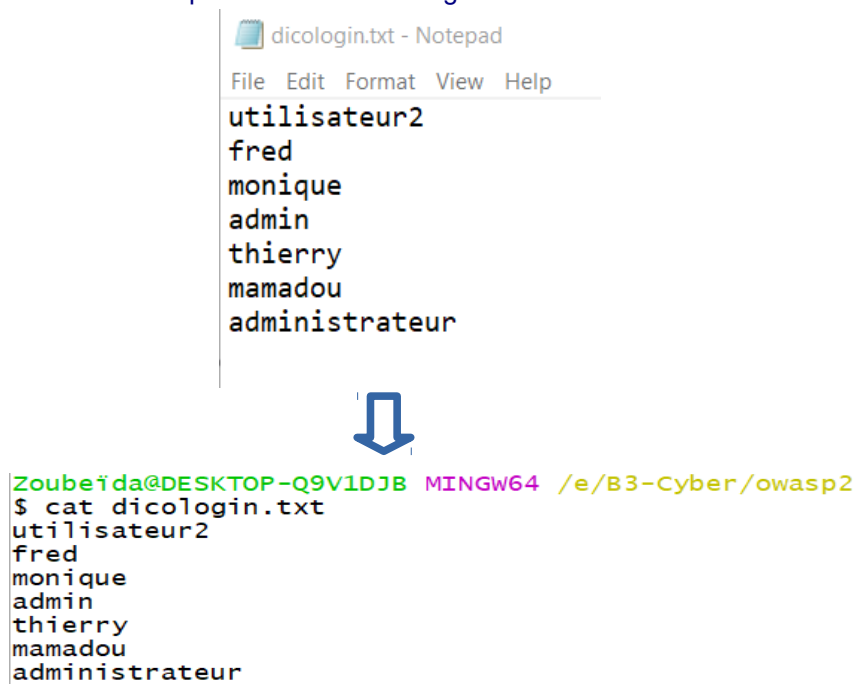


Toujours dans cette fenêtre, il faut sélectionner avec un double clic de souris la valeur correspondant au login (*utilisateur1* dans cette capture d'écran) et cliquer sur le bouton **Add**. Nous travaillerons donc avec une seule variable, d'où le mode **Sniper**. D'autres modes existent et permettent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

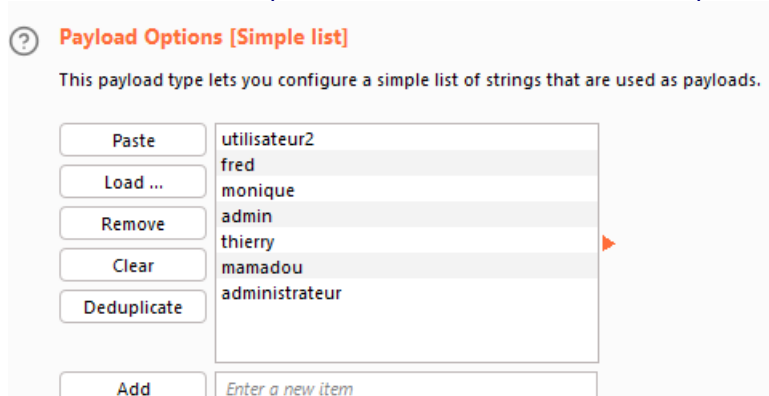
Une fois la valeur sélectionnée, Bupsuite testera en boucle différents logins en remplaçant la variable par les valeurs indiquées dans le dictionnaire.



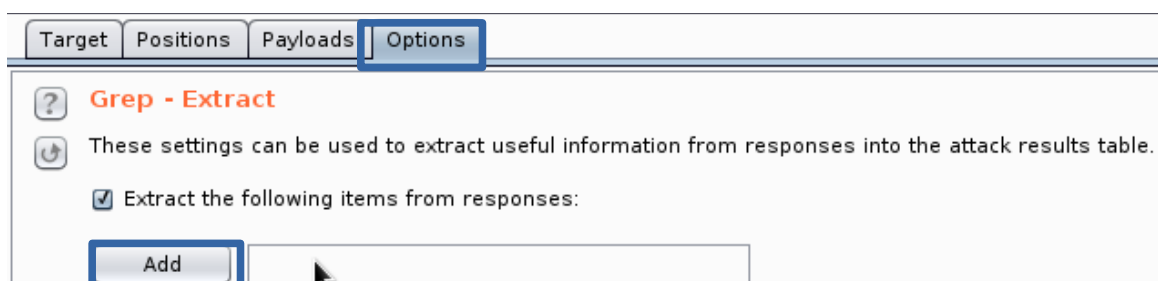
Puis, cliquer sur l'onglet **Payload** et charger un dictionnaire de login. Ce dictionnaire peut être créé à l'aide d'un éditeur de texte comportant une liste de login.



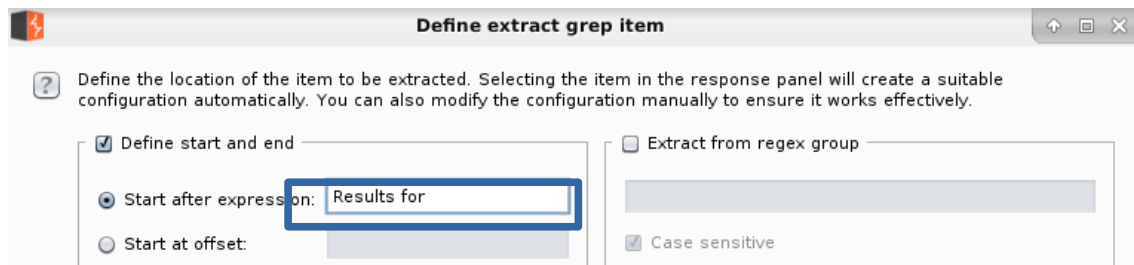
La sélection du dictionnaire se fait en cliquant sur le bouton **Load** de la rubrique **Payload options**.



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.

Requ...	Payload	Status	Error	Timeout	Length	Results for	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	1089	zoubeida"><accou...	
1	utilisateur2	200	<input type="checkbox"/>	<input type="checkbox"/>	908		
2	fred	200	<input type="checkbox"/>	<input type="checkbox"/>	900		
3	monique	200	<input type="checkbox"/>	<input type="checkbox"/>	903		
4	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	978	admin"><account>...	
5	thierry	200	<input type="checkbox"/>	<input type="checkbox"/>	903		
6	mamadou	200	<input type="checkbox"/>	<input type="checkbox"/>	973	mamadou"><accou...	

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.