CONTRIBUER À METASPLOIT : GUIDE DU DÉBUTANT

Davy Douhine (@ddouhine)

mots-clés : ??????????????????

Inutile de présenter le framework d'exploitation Metasploit, conçu par HD Moore et désormais maintenu par la société Rapid7. Devenu une trousse à outils incontournable pour les tests d'intrusion en quelques années, il est très largement utilisé par la communauté de la sécurité informatique.

C'est d'ailleurs probablement cette communauté qui est à l'origine de ce succès, car elle contribue grandement au développement de l'outil.

Mais comment peut-on contribuer au projet ? Est-ce à la portée de tout le monde ? En prenant un exemple concret de soumission d'exploit, nous allons essayer de répondre à ces questions.

1 Pas d'exploit pas de chocolat

Pour exploiter une vulnérabilité, il faut le plus souvent un *exploit*, aussi simple soit-il. C'est ce petit bout de code qui va déclencher une vulnérabilité pour ensuite exécuter sur la cible une charge utile qui réalisera les actions menant à l'objectif de l'attaquant. Les pentesteurs et autres curieux plus ou moins bienveillants utilisent ou écrivent régulièrement des *exploits*.

Mais écrire un exploit qui « marchotte » en Perl ou en Python et l'utiliser pour une mission est une chose. L'affûter et le rendre public en est une autre. Cette question : publier ou non, je me la suis posée il y a quelque temps lors d'un test d'intrusion et je suppose que d'autres se la pose quotidiennement.

La petite histoire commence par un échec. En l'occurrence un test d'intrusion raté. Il faut dire que tous les ingrédients sont réunis : un sacré périmètre, peu de temps, une seule vulnérabilité intéressante, mais aucun exploit connu. Alors surgit un dilemme : doit-on bâcler le test d'intrusion en utilisant la formule consacrée dans le rapport « un attaquant pourrait développer un exploit pour cette vulnérabilité » ou écrire cet exploit ? Dans un élan de courage et d'optimisme, j'ai préféré ne pas employer le conditionnel et j'ai donc opté pour la *seconde voie*.

Mais calmons d'emblée les ardeurs des aficionados d'OllyDbg, vous n'aurez pas le moindre opcode dans cet article. Nous allons présenter une vulnérabilité qui ne nécessite pas de manipuler la mémoire ou les registres.

La vulnérabilité en question, **CVE-2011-0647 [1]**, intitulée « EMC Replication Manager Command Execution »

Fig. 1 : Extrait du rapport Nessus mentionnant la vulnérabilité.

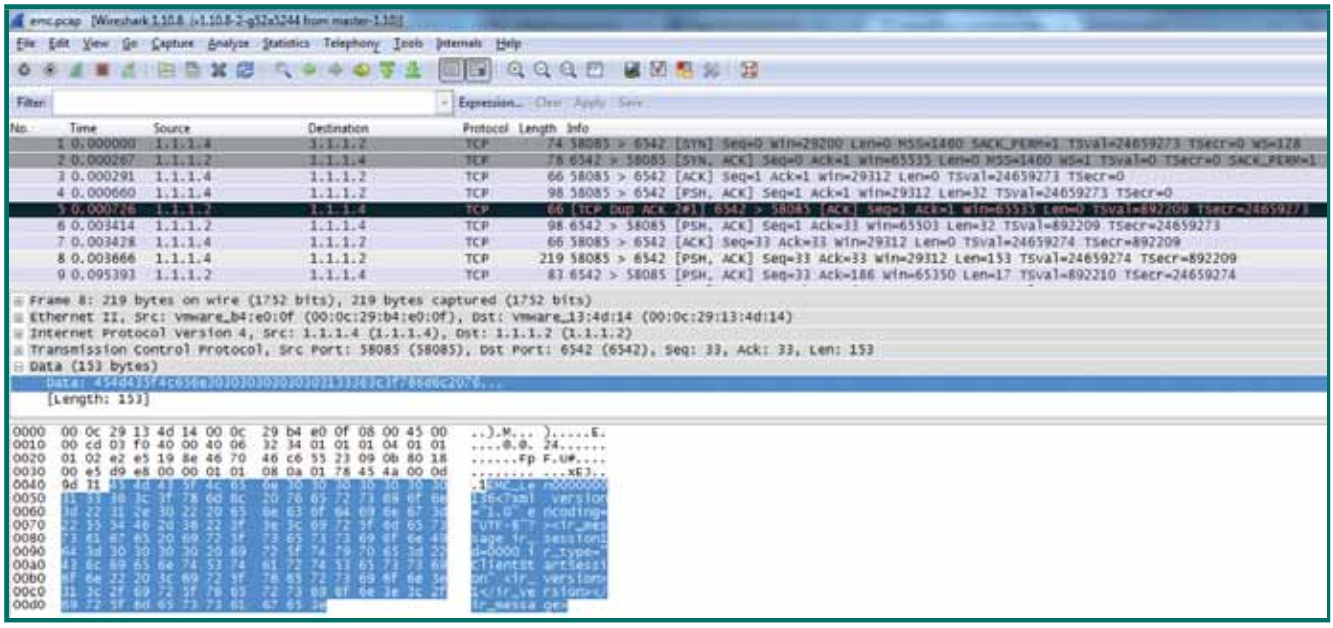


Fig. 2 : Extrait d'une communication avec le démon EMC Replication Manager Client Control.

impacte l'outil de réplication EMC Replication Manager **[2]**. Son score CVSS de 10 et sa belle couleur violette sous Nessus la rendent plutôt attrayante.

Le synopsis de Nessus dit que l'hôte distant exécute le démon EMC Replication Manager Client Control et que la version de ce logiciel est affectée par une vulnérabilité d'exécution distante de commande. Réussir à l'exploiter permettrait d'exécuter des commandes sur la cible et donc de rendre le rapport un peu plus illustré. Nessus précise même que c'est la gestion du message **RunProgram** qui pose problème. Mais comment envoyer ce type de message ?

Malheureusement, ce n'est pas en fouillant du côté des plugins de Nessus que nous aurons la réponse, car le plugin détectant la vulnérabilité fait partie de la fraction qui est compilée. Nous n'aurons donc pas accès facilement au code source en NASL.

Toutefois si Nessus a su détecter la vulnérabilité à distance et que le protocole n'utilise pas de chiffrement, une capture réseau devrait permettre d'identifier le fameux message. La chance est avec nous : pas de chiffrement. Les secrets du plugin ne resteront pas inviolés très longtemps. En effet, la capture permet de comprendre le protocole d'échange : on se dit bonjour, on initie une session, puis on demande le lancement d'un programme.

Après quelques tentatives, nous avons un exploit fonctionnel :

```
root@kali:/home/davy# perl emc_poc.pl --dst 1.1.1.2 --port 6542
--mode run --cmd c:\\windows\\system32\\calc.exe
Mode: ping
[*] We send hello...
[*] We send a ping...
[*] We send the endstring to disconnect...
[*] We get back:
```

```
1RAWHELLO00000000000000000000000000EMC_Len0000000240
<?xml version="1.0" encoding="UTF-8"?>
<ir_message ir_sessionId="00000" ir_requestId="00000" ir_
type="Status" ir_status="0">
<ir_statusType>final</ir_statusType>
<ir_statusTime>2014 07 26 11:49:50</ir_statusTime><errno>0</errno>
</ir_message>
```

Fig. 3 : Calc.exe est bien exécuté sur la cible.

2 Exploit standalone VS module Metasploit

L'exploit, écrit en Perl, est suffisant pour apporter une conclusion différente à la mission. Nous ne parlerons pas au conditionnel dans le rapport et nous pourrons ajouter l'exploit en annexe. Mais devons-nous en rester là pour autant ?

Le pentester doit travailler vite et les frameworks d'exploitation permettent de gagner un temps fou, car non seulement ils rassemblent des centaines d'exploits fiabilisés et prêts à l'emploi, mais ils fournissent également des outils puissants pour automatiser les tâches de reconnaissance, d'attaques et surtout de post-exploitation.

Intégrer l'exploit sur Metasploit permettrait par exemple de bénéficier de la puissance de meterpreter. Le script Perl pourrait très bien délivrer le premier étage de la charge utile de meterpreter, mais une approche plus élégante serait de créer un exploit Metasploit. Bien sûr, il serait possible de faire ça dans son coin, mais ajouter

l'exploit au repository officiel Metasploit permettrait aux autres utilisateurs d'en bénéficier. En tant qu'utilisateur régulier de Metasploit, cela serait aussi une sorte de remerciement pour les autres contributeurs.

À noter que si dans ce cas l'*exploit* initial a été écrit en Perl, il aurait peut-être été plus simple de l'écrire directement en Ruby pour Metasploit. En effet, Metasploit propose de nombreuses fonctions qui peuvent faciliter la vie : que vous cherchiez à générer un fichier ou des requêtes HTTP, Metasploit saura faire.

Lors d'une discussion avec Juan Vazquez, un des contributeurs de Rapid7 les plus prolifiques, je décide de me lancer dans la soumission de cet exploit. Comme il le fait remarquer, même si l'exploit ne sera probablement pas très utile sur Internet, il pourrait l'être lors de tests d'intrusion internes.

3 Se construire un environnement de développement Metasploit

Les contributions au projet Metasploit se font par le biais de leur GitHub. Metasploit met à disposition un wiki **[4]** décrivant les étapes nécessaires à l'installation d'un environnement minimal permettant de contribuer. Un premier conseil : lisez-le. J'aurais vraiment aimé qu'il soit aussi complet lorsque je me suis intéressé au sujet pour la première fois. Un deuxième conseil : n'optez pas pour une Kali. Ce fut mon premier choix par facilité, mais à l'installation des prérequis j'ai vite compris qu'il faudrait revoir ce choix pour ne pas perdre trop de temps sur cette première étape. À ce moment, j'ai bien tenté de savoir s'il n'y avait pas d'autres moyens de soumettre un exploit. Réponse sur IRC **[5]** : « github is the right way ».

Après avoir installé une distribution Ubuntu comme conseillé, j'ai installé git puis créé un compte GitHub. Ensuite, il suffit de configurer git pour qu'il soit lié au compte GitHub. Toutes les étapes sont détaillées sur le wiki ; inutile de revenir dessus. Une fois les prérequis installés correctement on peut entreprendre le développement du module (voir encadré).

Les modules Metasploit !

Dans le langage Metasploit, un module est un script Ruby. Les modules auxiliary s'utilisent de manière autonome. Par exemple, le module `auxiliary/scanner/portscan/tcp` va effectuer un balayage de ports, afficher les résultats puis rendre la main. Les modules exploit s'utilisent conjointement avec d'autres modules. Par exemple, le module `exploit/windows/browser/ms14_012_textrange`, une fois la vulnérabilité exploitée, va utiliser un module payload (ex : `payload/windows/meterpreter/reverse_tcp`) pour délivrer une charge utile.

4 Écriture, soumission et correction

Chaque module est composé de deux parties. La première contient des éléments d'information (description, références, etc.) ainsi que des éléments de configuration (architecture processeur ciblée, taille maximale de la charge utile, port, etc.). Ils sont affichés dans `msfconsole` lorsque l'on tape `info`. Les éléments qui sont à la main de l'utilisateur sont affichés lorsque l'on tape `show options`. La seconde partie du module contient la partie exécutable qui est généralement découpée en fonctions.

Pour débuter l'écriture d'un nouveau module, il est conseillé de se baser sur un module existant fournissant des fonctionnalités similaires. Utilisez de préférence un module récent, car en utilisant un module vieux de plusieurs années vous allez peut-être utiliser des fonctions qui ont été remplacées par d'autres. Vous vous risquez donc à devoir réécrire une partie du module après sa relecture par les équipes Rapid7.

Si vous écrivez un module exploitant une vulnérabilité de type corruption de mémoire comme un débordement de tampon et que vous utilisez Immunity Debugger ou Windbg pour déboguer la vulnérabilité, vous pouvez utiliser mona **[6]** pour vous préparer un squelette d'exploit au format Metasploit en tapant : `!mona skeleton`.

Lorsque vous souhaitez tester une première version de votre module, vous pouvez déposer le fichier directement dans l'arborescence Metasploit. Dans notre cas, elle se trouve ici : `/home/davy/git/metasploit-framework/modules/`.

Notre exploit peut donc être déposé dans : `/home/davy/git/metasploit-framework/modules/exploits/windows/emc/replication_manager_exec.rb`.

À ce stade, vous aller enfin pouvoir tester votre module en grandeur nature. Il est temps de lancer `msfconsole`.

Vont s'en suivre des allers-retours (dont le nombre est inversement proportionnel à votre niveau de compréhension de Ruby, de Metasploit et de la vulnérabilité) entre votre éditeur de texte et `msfconsole`. Après chaque modification du module, au lieu de recharger Metasploit dans son ensemble, il suffit de taper `reload` pour que le fichier soit rechargé.

Lorsque le module est fonctionnel et stable, il faut vérifier sa conformité avec les critères d'écriture de Metasploit.

Il y a sûrement des problèmes d'indentation, des espaces en trop en fin de ligne, etc. L'outil `msftidy` va vous aider dans la détection de ces erreurs de style.

```
ruby /home/davy/git/metasploit-framework/tools/msftidy.rb /home/davy/git/
metasploit-framework/modules/exploits/windows/emc/replication_manager_exec.rb
```

Des vérifications supplémentaires ont été ajoutées récemment **[7]** pour respecter le Ruby Style Guide **[8]**. L'objectif de tous ces contrôles est d'augmenter la maintenabilité de Metasploit en standardisant au maximum les modules.

SANS

www . hsc-formation . fr

SANS Institute

La référence mondiale en matière de formation et de certification à la sécurité des systèmes d'information

FORMATIONS INTRUSION
Cours SANS Institute
Certifications GIAC

SEC 504
Techniques de hacking, exploitation de failles et gestion des incidents

SEC 542
Tests d'intrusion des applications web et hacking éthique

SEC 560
Tests d'intrusion et hacking éthique

SEC 642
Tests d'intrusion avancés des applications web et hacking éthique

SEC 660
Tests d'intrusion avancés, exploitation de failles et hacking éthique

Dates et plan disponibles
Renseignements et inscriptions
par téléphone
+33 (0) 141 409 700
ou par courriel à:
formations@hsc . fr

GIAC SANS

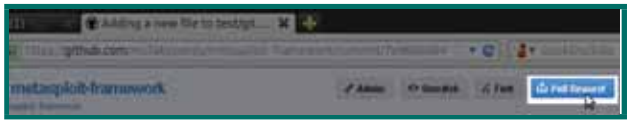
HSC

Attention, il est conseillé de mettre à jour son Metasploit avant de lancer `msftidy` pour être sûr d'avoir la dernière version de l'outil.

Par mesure de précaution, une des premières étapes qui est réalisée automatiquement par Rapid7 lors d'une *pull request* est un contrôle `msftidy`. Mais autant être propre dès le départ. Une fois que le module passe `msftidy` sans erreur, le moment est venu de le rendre public en effectuant une *pull request*.

```
ssh -T github
cd /home/davy/git/metasploit-framework
git push origin master
git checkout master
git checkout -b replication_manager_exec
git add modules/exploits/windows/emc/replication_manager_exec.rb
git commit -m "added aux module for EMC Replication Manager Command Execution"
git push origin replication_manager_exec
```

Finalement, après s'être connecté sur GitHub il suffit de cliquer sur le bouton ***pull*** pour que la *pull request* soit émise.



Le module est maintenant à la main des développeurs de Rapid7 qui vont effectuer leurs revues puis le tester.

Si l'application vulnérable est simple à trouver, ils testeront le module en conditions réelles. Si par contre, elle est payante et qu'ils ne peuvent pas se la procurer, ils demanderont des preuves à l'auteur du module (copie d'écran ou capture réseau).

S'ils n'ont rien à redire sur les fonctionnalités et le style du module, ils l'intégreront directement en « mergeant » la *pull request* dans la branche master. Le module sera instantanément disponible pour tous les utilisateurs.

Ou alors ils peuvent faire des remarques pour corriger ou améliorer certaines parties. Ce fut mon cas. Comme il le fait avec beaucoup de débutants, Juan Vazquez de l'équipe Rapid7 m'a aidé à réécrire et à perfectionner ce module. J'ai initialement soumis un bête module *auxiliary* qui n'était rien de plus que la transformation en Ruby du script Perl qui avait été écrit pour le test d'intrusion. Mais, comme Juan me l'a judicieusement fait remarquer, pour les vulnérabilités de type exécution de commande système, il faut systématiquement essayer d'écrire un exploit de manière à bénéficier des sessions meterpreter et donc de tous les outils de post-exploitation. La plupart du temps, ce type de vulnérabilité permet d'aboutir sans encombre à un module de type exploit. Effectivement, en utilisant le mixin `CmdStagerVBS` pour délivrer la charge utile, notre session meterpreter s'établit à tous les coups.

La cinématique d'exploitation est la suivante : le module lance le *handler metasploit*, se connecte sur le service EMC, dit bonjour, initie une session EMC, puis demande le lancement de `cmd /c #{CmdStagerVBS}`. Cette commande écrit sur le système de fichier de la cible le premier étage de la charge sous la forme d'un fichier VBS qui est ensuite exécuté par `CScript`. Ce premier étage va se connecter au *handler* pour récupérer le restant de la charge utile.

Une fois les remarques prises en compte, il faudra effectuer une nouvelle *pull request*. Si cette fois le code est satisfaisant, le module est ajouté à la branche master par Rapid7 **[10]** et vous recevez une notification de « Land ». Champagne !

Quelques mois plus tôt, Juan avait presque intégralement réécrit un autre module que j'avais coécrit avec un collègue **[9]**. Il n'est donc pas avare de son temps et de ces conseils.

5 Comment contribuer autrement ?

Combien d'exploits s'ennuient à mourir sur des secteurs de disques durs ou de clés USB alors qu'ils pourraient passer du bon temps à satisfaire des palanquées de pentesteurs et à attiser la curiosité de quelques RSSI ?

Vous l'avez lu, intégrer un module est assez simple et rapide, alors partagez en soumettant les vôtres !

Et même sans disposer d'exploit sous la main, il est possible de contribuer. Car si les exploits sont la matière première de Metasploit, ils seraient inoffensifs s'il n'était pas possible de les combiner avec des charges utiles efficaces. Alors si vous êtes doués en écriture de shellcodes, vous pouvez apporter votre aide sur les modules de type payload. Par exemple, cette *pull request* **[11]** propose une nouvelle version de meterpreter pour OS X qui sera, à terme, compatible avec les iOS **[12]**.

D'autres contributions, techniquement plus accessibles, peuvent être intéressantes aussi comme un module de post-exploitation ou un auxiliary.

Et finalement, même sans écrire la moindre ligne de code vous pouvez aussi participer en documentant le projet ou en réalisant des screencasts. ■

Références

[1] http://cvedetails.com/cve/2011-0647

[2] http://france.emc.com/storage/replication-manager.htm

[3] https://github.com/rapid7/metasploit-framework

[4] https://github.com/rapid7/metasploit-framework/wiki/Setting-Up-a-Metasploit-Development-Environment

[5] https://freenode.net/#metasploit

[6] http://redmine.corelan.be/projects/mona

[7] https://community.rapid7.com/community/metasploit/blog/2014/07/17/weekly-metasploit-update-embedded-device-attacks-and-automated-syntax-analysis

[8] https://github.com/bbatsov/ruby-style-guide

[9] https://www.rapid7.com/db/modules/exploit/unix/webapp/spip_connect_exec

[10] https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/emc/replication_manager_exec.rb

[11] https://github.com/rapid7/metasploit-framework/pull/34827

[12] http://sourceforge.net/p/metasploit/mailman/message/32514602/