

Création d'un Honeypot pour la Détection d'URLs Malicieuses

Julian COUX, Tom JEANNESSON, Hugo BRUCKER

January 20, 2025

Introduction

Un **honeypot** est un dispositif de sécurité conçu pour simuler une cible vulnérable et attirer les attaquants. Son but principal est d'enregistrer leurs activités, d'analyser leurs comportements, et dans notre cas, d'identifier des URLs malicieuses. En pratique, un honeypot agit comme un leurre qui donne l'illusion d'une faiblesse sécuritaire à exploiter. Lorsqu'un attaquant interagit avec le honeypot, toutes ses actions, telles que les commandes exécutées ou les fichiers demandés, sont enregistrées et analysées.

L'objectif ici est de déployer un honeypot capable de détecter des activités malveillantes, comme des tentatives d'accès à distance ou des téléchargements de fichiers via des outils tels que `wget`, afin de récupérer et valider les URLs associées. Tout notre travail est disponible ici.

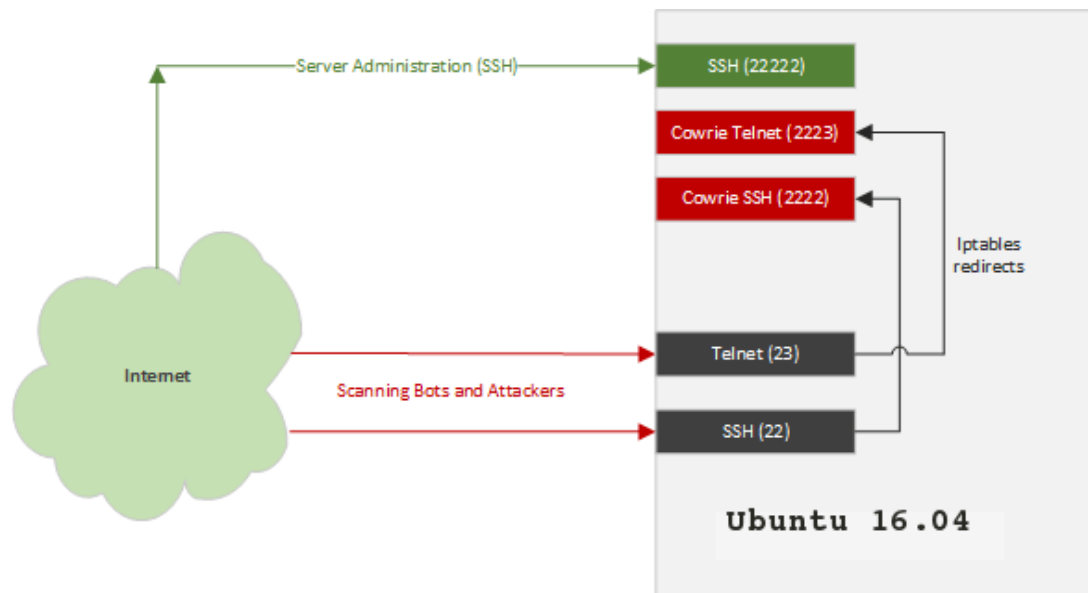


Figure 1: Cowrie HoneyPot

Outils pour créer le honeypot

Introduction à Cowrie

Afin de mettre en ligne notre premier honeypot, nous avons choisi d'utiliser Cowrie. Cowrie est un honeypot interactif simulant un shell SSH/Telnet. Conçu pour attirer les

attaquants, Cowrie enregistre les activités malveillantes, notamment les tentatives de connexion, les commandes saisies et les transferts de fichiers. Celui-ci permet de simuler un shell en Python, grâce à lui, un attaquant qui entre sur le serveur aura l'impression d'être dans un vrai shell, mais sera en réalité dans le honeypot.

Pour résumer, voici quelques avantages de Cowrie :

- Il simule un environnement réaliste (exemple : un système Linux vulnérable).
- Il collecte des logs détaillés.
- Il est facile à déployer via Docker.

Le choix de cet outil fut donc assez simple pour nous dans un premier temps.

Environnement technique

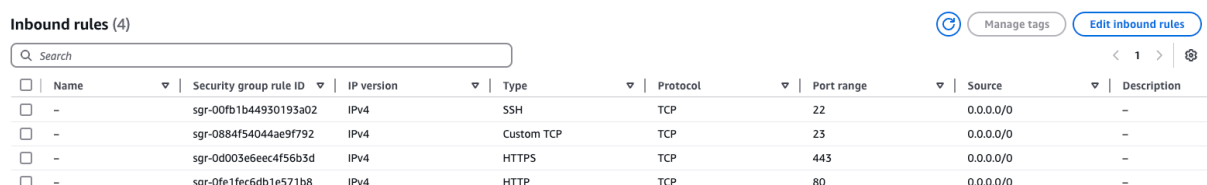
Voici la stack technique que nous avons choisi pour déployer le honey pot :

- **AWS** : Fournit l'infrastructure cloud pour exécuter Cowrie.
- **Docker Compose** : Simplifie le déploiement en créant des conteneurs pour Cowrie, MySQL, et Grafana.
- **MySQL et Grafana** : Pour stocker les logs et les visualiser.

Étapes du déploiement

1. Création d'une instance AWS :

- Utiliser une instance EC2 de type t3.small (nous avons testé avec une instance plus petite, mais celle-ci manquait de mémoire).
- Configurer les règles de groupe de sécurité internes pour autoriser les connexions SSH et les ports nécessaires (par exemple, 22, 80, 443). Voici un lien documentant comment le faire: <https://gist.github.com/tusharf5/89a094de01321880fdf44bf1d4e4ea4c>
- Configurer les groupes de sécurité AWS pour laisser entrer les attaquants.



<input type="checkbox"/>	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
<input type="checkbox"/>	-	sgr-00fb1b44930193a02	IPv4	SSH	TCP	22	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0884f54044ae9f792	IPv4	Custom TCP	TCP	23	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0d003e6eac4f56b3d	IPv4	HTTP5	TCP	443	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0fe1fec6db1e571b8	IPv4	HTTP	TCP	80	0.0.0.0/0	-

Figure 2: Groupes de sécurité

2. Installation de Docker Compose :

Docker est très simple à installer sur une instance ubuntu, voici comment le faire : <https://docs.docker.com/engine/install/ubuntu/>

3. Configuration de Cowrie avec docker-compose :

Créer un fichier `docker-compose.yml` :

```

1 version: "3"
2 services:
3   cowrie:
4     image: ghcr.io/tomjeannesson/ensimag3a-cowrie/cowrie:latest
5     ports:
6       - "22:22"
7       - "23:23"
8       - "2222:2222"
9       - "2323:2323"
10
11   db:
12     restart: always # if something brokes, it will automatically restart
13     image: ghcr.io/tomjeannesson/ensimag3a-cowrie/db:latest
14     container_name: db
15     volumes:
16       - cowrie-db:/var/lib/mysql:z
17     ports:
18       - "3306:3306"
19     environment:
20       - MYSQL_ROOT_PASSWORD=root
21       - MYSQL_PASSWORD=root
22       - MYSQL_DATABASE=cowrie
23       - MYSQL_USER=cowrie
24
25   grafana:
26     image: grafana/grafana:latest
27     restart: always
28     volumes:
29       - grafana:/var/lib/grafana:z
30     environment:
31       # - GF_SERVER_DOMAIN=${SERVICE}
32       # - GF_SERVER_ROOT_URL=https://${SERVICE}/
33       - GF_USERS_ALLOW_SIGN_UP=false
34       - GF_ANALYTICS_CHECK_FOR_UPDATES=false
35       - GF_ANALYTICS_REPORTING_ENABLED=false
36     labels:
37       - "traefik.enable=true"
38       - "traefik.http.routers.django.rule=Host(`${SEB_DOMAIN}`)"
39       - "traefik.http.routers.django.entrypoints=websecure"
40       - "traefik.http.routers.django.tls.certresolver=myresolver"
41       - "traefik.http.services.django.loadbalancer.server.port=3000"
42     expose:
43       - "3000"
44
45   traefik:
46     image: "traefik:v3.1.3"
47     container_name: traefik
48     command:
49       - "--log.level=INFO"
50       - "--api.insecure=false"

```

```

51 - "--providers.docker=true"
52 - "--providers.docker.exposedbydefault=false"
53 - "--entrypoints.web.address=:80"
54 - "--entrypoints.websecure.address=:443"
55 - "--entrypoints.web.http.redirections.entryPoint.to=websecure"
56 - "--entrypoints.web.http.redirections.entryPoint.scheme=https"
57 - "--certificatesresolvers.myresolver.acme.httpchallenge=true"
58 - "--certificatesresolvers.myresolver.acme.httpchallenge.entrypoint=web"
59 - "--certificatesresolvers.myresolver.acme.email=tomjeanesson@gmail.com"
60 - "--certificatesresolvers.myresolver.acme.storage=/letsencrypt/acme.json"
61 ports:
62 - "80:80"
63 - "443:443"
64 volumes:
65 - "/var/run/docker.sock:/var/run/docker.sock:ro"
66 - /traefik-letsencrypt:/letsencrypt
67
68 volumes:
69   cowrie-etc:
70   cowrie-var:
71   cowrie-db:
72   grafana:

```

Nous utilisons traefik comme reverse proxy pour assurer une connexion en https avec le dashboard grafana.

Une fois ce fichier créé, il suffit de lancer le projet avec

```

1 docker compose up --build -d

```

Récupération des logs dans une base de données

Cowrie génère des logs détaillés contenant les commandes exécutées, les adresses IP des attaquants, et les fichiers téléchargés. Ces logs sont écrits dans un fichier ou envoyés vers une base de données MySQL.

Configuration de Cowrie pour MySQL

Voici le fichier de configuration Cowrie que nous avons utilisé `cowrie.cfg` :

```

1 [telnet]
2 enabled = True
3 # listen 23/tcp and 2323/tcp
4 listen_endpoints = tcp:23:interface=0.0.0.0 tcp:2323:interface=0.0.0.0
5
6 [ssh]
7 # listen 22/tcp and 2222/tcp
8 listen_endpoints = tcp:22:interface=0.0.0.0 tcp:2222:interface=0.0.0.0
9

```

```

10 [output_mysql]
11 enabled = true
12 host = db
13 database = cowrie
14 username = cowrie
15 password = root
16 port = 3306
17 debug = true

```

Ceci nous permet de récupérer automatiquement tous les logs de Cowrie et de venir les récupérer dans un dashboard Grafana.

Visualisation des résultats

Afin de visualiser les données facilement et avoir une vue sur les commandes les plus testées. Nous avons choisi Grafana pour faire un dashboard complet qui affiche toutes les données importantes.

Voici notre dashboard principal:

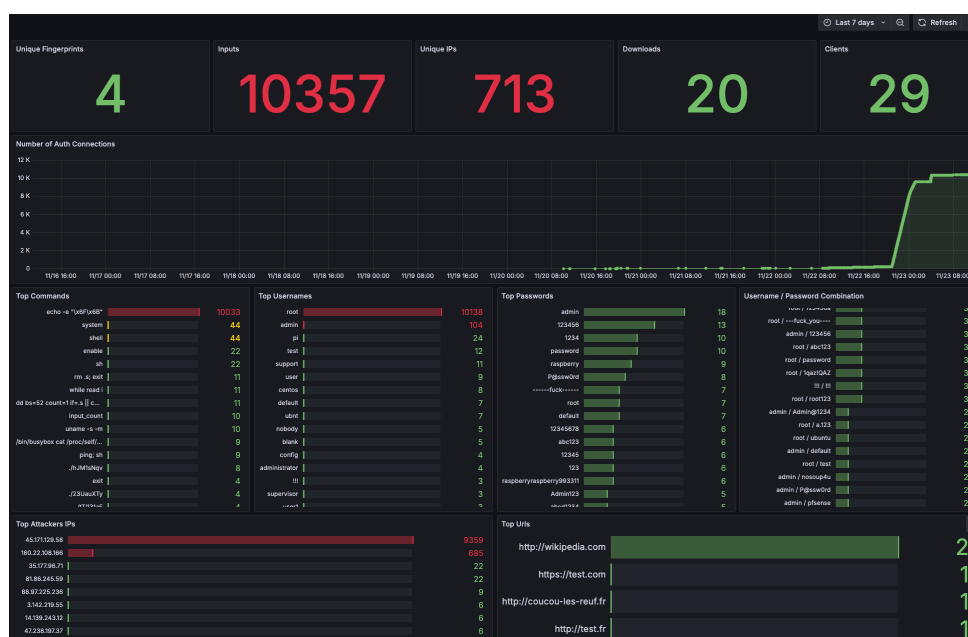


Figure 3: Dashboard Grafana

Comme nous pouvons le voir, il y a tout d'abord une série d'information en haut, tel que le nombre d'entrées (lignes de la bdd), le nombre d'adresses IP uniques, ...

En dessous, un historique des visites de notre honeypot (les connexions via ssh). On peut voir que aux dernières nouvelles, on référençait 12000 visites. Sachant que la plupart d'entre elles sont simplement des tests ssh, sans but d'attaquer le serveur (pour le moment).

Ensuite on a plusieurs graphiques qui mettent en avant les informations qui reviennent le plus, tel que les noms d'utilisateurs, les mots de passe, les commandes et les couples username/password. Ces informations sont essentielles pour étudier le comportement des attaquants et mieux connaître leurs techniques.

Enfin, en bas du dashboard, on a 2 graphiques. Un qui répertorie les adresses IP les plus courantes et l'autre les URLs les plus souvent trouvées. C'est cette section qui est

importante pour le projet et dans laquelle on pourra récupérer les URLs potentiellement malicieuses.

Pour intégrer Grafana à notre serveur, voici comment nous avons procédé :

- Connecter Grafana à la base MySQL.
- Créer le dashboard personnalisé avec les éléments présentés.

Geomap avec coordonnées IP

Par la suite, nous avons décidé d'enrichir notre tableau de bord avec une carte du monde permettant de visualiser l'origine géographique des attaquants.

Pour ce faire, nous avons intégré le plugin geomap de Grafana et développé un script Python pour convertir les adresses IP en coordonnées géographiques.

Voici un exemple de son utilisation :

```
1 import requests
2
3 def get_geo(ip):
4     url = f"http://ip-api.com/json/{ip}"
5     response = requests.get(url)
6     return response.json()
7
8 print(get_geo("8.8.8.8"))
```

Nous avons constaté que la majorité des attaquants provenaient de grandes capitales, principalement situées en Asie de l'Est, sur la côte Est de l'Amérique du Nord, ainsi qu'en Europe.

Ci-dessous, une capture d'écran partielle de la carte (la version complète ayant été perdue, comme mentionné dans la section *Difficultés*) :



Figure 4: Geomap Grafana

Traitement des URLs

Récupération des URLs

Concernant la récupération des URLs malicieuses, nous utilisons deux méthodes issues des logs de notre honeypot.

1. Récupération via les logs des commandes tapées

La première consiste à filtrer toutes les commandes tapées par les attaquants et récupérer tous ce qui ressemble à un lien. En général, ce sont les commandes qui contiennent `curl` ou `wget`.

Voici la requête SQL nécessaire à mettre dans Grafana afin de récupérer toutes les URLs tapées :

```
1 SELECT COUNT(input) AS input_count, input
2   FROM cowrie.input
3  WHERE input LIKE 'http%'
4   GROUP BY input
5  ORDER BY input_count DESC
6   LIMIT 50;
```

Cette méthode nous permet de récupérer toutes les URLs que le attaquants exécutent volontairement dans notre serveur.

Le problème étant que sur cowrie, le nombre d'URLs tapées par les attaquants reste assez faible. Car avant de taper des commandes contenant des URLs malicieuses, ils tapent quelques commandes pour tester le serveur sur lequel ils sont. Et potentiellement détecter que c'est un HoneyPot. Malheureusement, dans sa configuration actuelle, Cowrie se fait trop facilement démasquer.

Nous avons donc besoin d'une nouvelle méthode pour récupérer des URLs avec les informations que nous avons déjà.

2. Reverse DNS pour récupérer les noms de domaine

Cette seconde méthode consiste à utiliser les adresses IP que nous avons recueilli pour faire du reverse DNS et récupérer leur nom de domaine. L'objectif à cela est d'essayer de cibler des noms de domaine malicieux qui peuvent appartenir à des groupes malveillants. De plus, nous cherchons à savoir si ces noms de domaine sont rattachés à des site web malicieux (URL malicieuse).

Pour cela, nous utilisons une api en Python, qui permet, en donnant l'adresse IP, de récupérer le nom de domaine attaché à celle-ci. Voici le code nécessaire pour cela :

```
1 ips = cursor.fetchall()
2
3 for index, ip_row in enumerate(ips):
4     ip = ip_row[0]
5     success = False
6     while not success:
7         try:
8             response = requests.get(f"{API_URL}-{ip}")
```

```

9         data = response.json()
10        success = True
11    except requests.exceptions.JSONDecodeError:
12        print("An error occurred, retrying in 1s.")
13        sleep(1)
14
15    if data["status"] == "success":
16        latitude = data["lat"]
17        longitude = data["lon"]
18        domain = socket.getnameinfo((ip, 0), 0)[0]

```

Cette méthode ne retourne pas de vraies adresses URLs, mais plutôt des hostname. Nous avons tout de même jugé intéressant d'essayer de cibler les noms de domaine qui revenaient souvent et de voir si ceux-ci peuvent cacher un site malicieux.

Parmi tous les noms de domaines que nous avons récupérés, nous avons fait un script (`best-domains.py`) afin de repérer ceux qui sont le plus souvent présents dans les hostnames. Voici certains intéressants que nous avons sélectionnés (parmi les 50 plus présents des 1500 hostnames) :

Nom de domaine	Nombre d'apparitions
spectrum.com	46
stretchoid.com	44
kyivstar.net	37
shadowserver.org	21
verizon.net	9
bufferover.run	8

Validation des URLs

Une fois les URLs malicieuses récupérées à partir des logs du honeypot, il est essentiel de les valider en les comparant à des bases de données reconnues pour répertorier les menaces en ligne. Sans le faire, les données récupérées ne valent rien. Cependant, ce processus pose plusieurs défis en termes d'automatisation que nous allons expliciter.

Exemple de scripts de comparaison

La première API que nous avons testé est celle de PhishTank. Elle permet de vérifier si une URL donnée est classifiée comme site de phishing. Voici un exemple de script Python illustrant cette démarche :

```

1  import requests
2  p = PhishTank()
3  result = p.check("http://example.com")
4
5  if result.in_database:
6      if result.valid:
7          print("{url} is a phish!".format(url=result.url))
8      else:
9          print("{url} is not a phish!".format(url=result.url))

```



```

10 else:
11     print("{url} is not in the PhishTank database".format(url=result.url))

```

Dans cet exemple, l'URL est soumise à l'API de PhishTank pour vérifier si elle est présente dans leur base de données. Si l'URL est identifiée, une alerte est déclenchée.

Cependant, le service était indisponible au moment où nous avons fait ces recherches, et nous avons donc cherché d'autres solutions.

Register

X New user registration temporarily disabled.

Figure 5: PhishTank

Nous nous sommes donc tournés vers VirusTotal, qui offre également une API permettant d'analyser des URLs. Voici un exemple d'utilisation que nous avons mis en place:

```

1  import json
2  from time import sleep
3
4  import requests
5  from domains import domains
6
7  apikey = "<YOUR_KEY>"
8  for domain in domains:
9      print(f"Checking {domain}")
10     url = "https://www.virustotal.com/vtapi/v2/domain/report"
11     passed = False
12     while not passed:
13         try:
14             params = {
15                 "apikey": apikey,
16                 "domain": domain,
17             }
18             response = requests.get(url, params=params)
19             response_json = json.loads(response.content)
20             passed = True
21         except json.decoder.JSONDecodeError:
22             sleep(5)
23
24     result = response_json.get("detected_referrer_samples", [])
25     d_list = []
26     for line in result:
27         d_list.append(line["positives"])
28
29     final_result = sum(d_list)
30     if final_result == 0:
31         print("The Domain Not Malicious! ")

```

```
32     elif final_result > 0:
33         print("The Domain Is Malicious ! ")
```

Nous rencontrons le problème qu'un grand nombre de domaines n'existent plus:

```
{'response_code': 0, 'verbose_msg': 'Domain not found'}
```

Ceci est dû au fait que notre méthode de récupération d'URL se focalise sur les serveurs attaquants, qui sont éphémères, et n'hébergent que rarement des scripts malicieux.

Ceci dit, nous avons tout de même réussi à dénicher des domaines potentiellement malicieux:

```
"BitDefender domain info": "This URL domain/host was seen to
                             host badware at some point in time",
ou
'alphaMountain.ai category': 'Information Technology, Suspicious'
```

Au final, nous estimons qu'environ 0.5% des urls sont malicieux, et 1% le sont potentiellement, selon l'API de VirusTotal. Une liste des noms de domaine que nous avons trouvé est disponible dans le dossier malicious.json, à la racine de notre projet, et les données récoltées de l'API sont visibles ici.

Pourquoi ce processus est difficile à automatiser ?

Bien que ces scripts permettent une validation partielle, automatiser entièrement le processus pose plusieurs problèmes :

1. Nature éphémère des URLs malveillantes : Les attaquants utilisent des domaines temporaires qui deviennent rapidement inactifs ou sont remplacés par d'autres. Ainsi, les URLs récupérées par le honeypot ne sont souvent plus actives au moment de leur validation.

2. Limites des APIs : Les services comme VirusTotal ou PhishTank imposent des restrictions sur le nombre de requêtes par minute/jour. Cela rend difficile l'analyse d'un grand volume de données collectées.

3. Variabilité des formats : Les journaux du honeypot contiennent parfois des données incomplètes ou malformées (par exemple, des URLs tronquées), ce qui complique leur traitement automatique.

4. Faible couverture des bases existantes : Les bases d'URLs malicieuses ne sont pas exhaustives, et de nombreuses menaces émergentes ne sont pas encore référencées. Cela peut conduire à des faux négatifs.

En résumé, bien que les outils disponibles permettent une certaine automatisation, la validation complète et fiable des URLs reste un processus complexe nécessitant une intervention manuelle ou des solutions plus sophistiquées pour améliorer l'efficacité.

Optimisation du Honeypot

Pour améliorer l'efficacité et l'attractivité de notre honeypot, plusieurs optimisations ont été envisagées et mises en œuvre. Ces optimisations visaient à attirer davantage d'attaquants tout en réduisant le temps et les efforts nécessaires pour traiter et analyser les données recueillies.

Tout d'abord, un choix technique que nous avons fait, nous a permis d'augmenter nos chances d'avoir du trafic sur notre serveur. En effet, nous avons utilisé de Elastic

BeanStalk sur AWS pour générer un nom de domaine. Celui-ci est alors directement introduit dans les DNS par AWS, ce qui le rend plus facilement repérable par des attaquants.

1. Publication de l'URL du Serveur

Après plusieurs jours de fonctionnement, nous avons constaté que notre honeypot n'attirait pas suffisamment d'attaquants. Nous avons alors envisagé une méthode simple pour les inciter à visiter notre serveur.

Nous avons inséré l'URL de notre serveur dans le fichier README de plusieurs repository publics sur GitHub, afin de le rendre plus accessible aux attaquants. Cette approche repose sur le fait que le crawling, une pratique courante chez les attaquants, permet de repérer les URL de serveurs potentiellement vulnérables. En publiant notre "appât" au sein de la vaste toile d'Internet, nous avons maximisé les chances d'être détectés.

Cette stratégie s'est avérée efficace, car nous avons rapidement observé une augmentation significative des visites sur notre serveur, validant ainsi notre hypothèse :



Figure 6: Timeline ajout URL sur Git

2. Automatisation de la Validation des URL Capturées

Plusieurs scripts Python ont été développés pour vérifier automatiquement les URL collectées (comme présenté), en s'appuyant sur des API. Cela a remplacé la vérification manuelle, réduisant ainsi les efforts nécessaires et accélérant l'analyse des données.

Difficultés rencontrées

Durant le développement de ce projet, nous avons rencontré plusieurs défis majeurs qui ont retardé l'atteinte de nos objectifs.

Premièrement, le **déploiement** de Cowrie sur AWS s'est avéré complexe en raison de conflits liés à la configuration des ports. Cette étape initiale a nécessité un temps d'ajustement important avant de stabiliser l'environnement.

Ensuite, après deux mois de fonctionnement sans incident, notre serveur a subitement **crashé**, entraînant la **perte de toutes les sauvegardes** et fichiers de configuration, y compris ceux du tableau de bord Grafana. Cet événement imprévu a doublé notre charge de travail, car nous avons dû tout reconfigurer depuis le début. Durant la période d'activité du serveur, nous avons constaté un nombre important de tentatives de **DDoS**, avec des milliers de connexions en un temps réduit, sans activité de notre part. Nous supposons qu'une de ces attaques est à l'origine du crash de l'instance AWS.

Par ailleurs, la majorité des URLs capturées s'avéraient **éphémères**, ce qui les rendait souvent impossibles à tester après leur collecte. Le processus de validation, difficile à automatiser, devait être effectué manuellement, ce qui était chronophage. Malheureusement, les URLs devenaient fréquemment obsolètes avant même que leur analyse puisse être menée à terme.

Enfin, bien que **Cowrie** soit un outil performant, il reste relativement **facile à détecter** par des attaquants expérimentés. Nous avons observé que, dès leur arrivée sur le serveur, certains exécutent des commandes spécifiques qui semblaient destinées à évaluer la légitimité de l'environnement. Ces comportements ont révélé que Cowrie était rapidement identifié, limitant ainsi son efficacité. Dans les sections suivantes, nous explorerons des alternatives permettant de concevoir un honeypot moins facilement détectable.

Résultats obtenus

Durant les deux mois d'exécution de notre projet (avant le crash du serveur), nous avons collecté un total de **1500 URLs et noms de domaines**. Parmi ces derniers, une fraction significative a été analysée pour leur potentiel caractère malicieux. Grâce à **VirusTotal**, nous avons confirmé que **0,5 %** des URLs étaient effectivement malveillantes, avec une estimation totale d'environ **1 %** de malicieux (certaines analyses n'ayant pas pu être finalisées en raison des limitations des API). *Ces résultats ont été détaillés dans la section "Validation des URLs".*

En parallèle, notre honeypot a enregistré près de **65 000 connexions** à notre serveur, incluant un nombre important de tentatives de DDoS qui ont contribué à ce chiffre élevé. Nous avons également accumulé une quantité importante de données avec **autant de logs de commandes exécutées** sur notre serveur, offrant une base riche pour l'analyse des comportements des attaquants.

Les erreurs que nous avons faites

Au terme de ce projet, nous avons pris conscience de plusieurs erreurs que nous avons commises. Celles-ci incluent :

- **Temps excessif consacré à Cowrie** : Bien que Cowrie soit un outil intéressant pour se familiariser avec les honeypots, il n'est pas le plus performant. Si nous devions recommencer, nous éviterions d'utiliser sa version émulée, car elle est trop facilement détectable. Nous aurions plutôt créé notre propre honeypot et utilisé Cowrie uniquement pour la collecte des logs du serveur.
- **Les IP des machines attaquantes ne contiennent pas les scripts malveillants** : Nous avons initialement cherché à obtenir les noms de domaine des attaquants dans le but de retracer leur origine et d'accéder à un ensemble d'URLs malveillantes. Cependant, nous avons constaté que la plupart de ces domaines étaient soit éphémères, soit issus de services tiers. Cette approche ne répondait finalement pas aux besoins du projet.
- **Manque de diversité dans la recherche d'URLs** : Nous nous sommes principalement concentrés sur la recherche d'URLs via le honeypot. Or, comme l'ont également observé d'autres groupes, cette méthode s'est révélée peu efficace. Nous aurions dû diversifier nos approches en développant, par exemple, des outils de crawling ou de phishing.
- **Absence de sauvegardes régulières de la base de données** : Comme mentionné lors de notre présentation orale, nous avons subi un crash serveur qui a entraîné la perte de 65 000 lignes dans la base de données, ainsi que des configurations de Grafana. Cette perte de données a généré un retard important, retard que nous aurions pu éviter avec des sauvegardes plus régulières.
- **Répartition insuffisante des tâches** : En explorant différentes techniques pour rechercher des URLs, nous aurions pu mieux répartir les tâches au sein de l'équipe. Une répartition plus précise des responsabilités aurait permis à chacun de se concentrer sur un aspect spécifique de la collecte d'URLs, multipliant ainsi les résultats par trois.

Futures améliorations

Comme mentionné précédemment, notre honeypot pourrait être significativement optimisé pour être plus efficace et moins détectable par les attaquants. Voici quelques pistes d'améliorations identifiées :

1. Rendre le honeypot moins détectable

Actuellement, notre honeypot est trop facilement repéré par les attaquants. Une solution serait de modifier sa structure pour permettre aux attaquants d'exécuter de vraies commandes dans un environnement "poubelle", qui leur serait laissé volontairement. Pour ce faire, nous pourrions déployer un container Docker dédié. Ce container serait isolé et contrôlé, offrant l'illusion d'un accès complet au serveur. Les attaquants croiraient ainsi infecter réellement la machine, tandis que nous conserverions un contrôle total. En cas de besoin, le container pourrait être redémarré pour effacer les modifications apportées.

2. Mise en place d'un système de sauvegarde automatique

Pour éviter de perdre les données critiques en cas d'incident, un système de sauvegarde automatique de la base de données pourrait être déployé. Cela garantirait l'existence de sauvegardes régulières et sécurisées, minimisant ainsi les risques liés à une éventuelle panne ou attaque.

3. Validation automatique des URLs malicieuses

Une amélioration majeure serait de développer d'autres scripts de validation automatique des URLs capturées. Il serait intéressant d'améliorer nos scripts pour faire de la classification d'URLs (en fonction de leur champ d'action par exemple). De plus, ces informations pourraient être centralisées sur un site web dédié, répertoriant les URLs malicieuses et les rendant accessibles pour d'autres chercheurs ou organisations de cybersécurité.

Conclusion

Ce projet a été une expérience enrichissante, tant sur le plan technique qu'organisationnel.

La réalisation du honeypot avec Cowrie nous a permis de simuler un environnement réel et d'étudier les méthodes des attaquants.

Grâce à la visualisation des données avec Grafana, nous avons pu analyser les comportements des intrusions et étudier en profondeur les URLs malicieuses. Bien que nous en ayons récupéré un grand nombre, peu étaient réellement malicieuses.

Ce projet nous a aussi formés sur les bonnes pratiques de sécurité, notamment la nécessité de ne pas utiliser de mots de passe faciles, et nous a sensibilisés à l'importance de la protection des serveurs.

1 Bibliographie

References

- [1] How to Setup "Cowrie - An SSH honeypot". <https://medium.com/threatpunter/how-to-setup-cowrie-an-ssh-honeypot-535a68832e4c>
- [2] Honey Pot SSH : Cowrie". *Mise en place et étude d'un Honey Pot SSH (Cowrie)* <https://www.it-connect.fr/mise-en-place-et-etude-dun-honey-pot-ssh-cowrie/>
- [3] Outil Cowrie". *Cowrie GitHub repository* <https://github.com/cowrie/cowrie>
- [4] Create a HoneyPot. *How to create an HoneyPot* <https://medium.com/@ecojumper30/creating-a-honeypot-f2b4cc33385a>
- [5] T-Pot tool. *The All In One Multi Honeypot Platform* <https://github.com/telekom-security/tpotce>
- [6] Vidéo Micode. *Comment piéger des centaines de pirates informatiques ?* <https://www.youtube.com/watch?v=kE3rGmoS0qs>
- [7] HoneyPots in Cybersecurity. *What Is a Honeypot?* <https://www.crowdstrike.com/en-us/cybersecurity-101/exposure-management/honeypots/>
- [8] Docker Inc. *Docker Engine Install Documentation*. Docker Docs. <https://docs.docker.com/engine/install/ubuntu/>
- [9] Grafana Labs. *Grafana Documentation*. Grafana Docs. <https://grafana.com/docs/grafana/latest/>
- [10] Amazon Web Services. *Amazon EC2 Documentation*. AWS Docs. <https://docs.aws.amazon.com/ec2/index.html>
- [11] IP-API. *IP Geolocation API*. <http://ip-api.com>
- [12] Grafana Labs. *Geomap Panel Documentation*. Grafana Docs. <https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/geomap/>
- [13] Oracle Corporation. *MySQL Documentation*. Oracle Docs. <https://dev.mysql.com/doc/>