dig .com AXFR +dnssec
Lister l'Internet grâce à DNSSEC

Aris Adamantiadis aris@badcode.be

Résumé. L'extension de sécurité DNSSEC, devenue incontournable dans l'Internet moderne, souffre d'un sérieux problème de conception pourtant connu des initiés : ses réponses NSEC et NSEC3 divulguent petit à petit le contenu des zones protégées par une technique nommée "Zone Walking". Cependant, certaines zones DNS résistaient encore à l'extraction de par leurs tailles conséquentes : les TLD tels que .com, .org et .fr qui contiennent des millions d'entrées.
Mon travail consiste à faire la revue des 1500 TLD pour évaluer les mitigations implémentées contre le zone walking, la conception d'un outil optimisé pour une utilisation quotidienne sur la totalité des TLD, ainsi que la tâche paradoxalement compliquées du crackage des noms de domaines.

1 Introduction

1.1 Motivations

La genèse de ce projet date de plusieurs années en arrière, lorsque j'ai utilisé l'outil *nsec3map* ([1]) lors de la phase de reconnaissance d'un test d'intrusion sur la zone DNS cible d'un client. J'ai testé nsec3map sur quelques TLD et eu la surprise que ça fonctionnait sur certains petits TLDs malgré les échecs sur des gros TLD comme .com.
Convaincu que cette expérience était sans intérêt, je l'ai laissée au placard jusqu'à ce que, après plusieurs discussions, je réalise qu'il y a un intérêt très concret en *threat intelligence* à posséder la liste de tous les noms de domaines valides sur internet, par exemple pour découvrir les nouveaux noms de domaines proches du nôtre, le typosquatting et les domaines de phishing. Il va de soi que ce projet n'a pas d'intention nuisible ou malveillante mais cela vaut la peine de le préciser.

Ce projet était aussi une bonne occasion d'appréhender la programmation *asyncio* sous Python, quelques touches d'OpenCL, et d'investir dans une RTX4090.[1]

L'objectif final est de publier une liste la plus complète possible des noms de domaine de niveau 2 (exemple.com) connus et de mettre ces listes à jour régulièrement de manière automatique.

[1] En note de frais, évidemment

1.2 Avertissement

La législation sur le scanning sur Internet est une zone grise, et il n'existe à ma connaissance pas de définition de ce qu'est un abus de ressources sur le réseau. Pour cette raison, j'ai pris un maximum de précautions pour réduire l'impact du scan sur les infrastructures DNS. L'outil *Malifar*, décrit plus loin dans le papier et disponible sur github, a le potentiel de générer beaucoup de trafic DNS s'il est utilisé sans surveillance. J'invite les potentiels utilisateurs à prendre cela en compte lors de la reproduction des résultats.

1.3 État de l'art

L'idée de faire du "Zone Walking" n'est pas neuve. DJB en avait déjà parlé en 2009 [7] et publié un outil basique. L'outil *nsec3map* a vu son premier commit en 2016. Le support pour le crackage des hashes [2] nsec3 a été ajouté en 2019 à *hashcat*. Certains demandes sur le github de *hashcat* [6] et conversations off-the-record me laissent supposer que je ne suis pas le seul à m'intéresser au sujet.
nsec3map Fonctionne plutôt bien sur des zones modestes (par exemple .be avec ses 500.000 entrées) mais est limité en performances réseau, en utilisation de la mémoire et en forçage des entrées nsec3. La littérature sur le crackage des hashs nsec3 de zones DNS est pratiquement inexistant.

Des projets de collecte des données TLD existent. Citons d'abord TLDR et TLDR-2 [2] qui visent à récolter des données via des transferts de zones, ensuite zone-walks [3] qui vise à collecter les données disponibles via NSEC.

1.4 Télécharger l'Internet : le bon, la brute et le truand

Trois méthodes s'offrent à nous pour le téléchargement des zones DNS. Certaines zones sont en open-access, de façons conditionnelles, inconditionnelles ou accidentelles.

Le bon L'ICANN, l'organisme gestionnaire des noms de domaines sur Internet, fournit un portail (Centralized Zone Data Service [14]) qui permet d'accéder aux zones DNS complètes de certains TLD participant à l'effort. Il est nécessaire de créer un compte, de faire une demande qui doit être approuvée, ce qui n'est pas trop dans l'esprit Hacker. Certains sites,

[2] désolé pour le franglais

comme `https://zonefiles.io/list/com/` vendent des bases de données de noms de domaine, mais leur business model et leur approche technique laissent planer un doute sur l'origine et la qualité de leurs données.

Certaines zones comme *.ch* et *.li*, gérées par SWITCH, sont accessibles via quelques commandes *dig* expliquées sur leur site web [16]

```

# filename ch_zonedata.key
key tsig-zonedata-ch-public-21-01 {
    algorithm hmac-sha512;
    secret "stZwEGApYumtXkh73qMLPqfbIDozWKZLkqRvcjKSpRnsor6A6M \
    xixRL6C2HeSVBQNfMW4wer+qjSOZSfiWiJ3Q==";
};

# filename li_zonedata.key
key tsig-zonedata-li-public-21-01 {
    algorithm hmac-sha512;
    secret "t8GgeCn+fhPaj+cRy1epox2Vj4hZ45ax6v3rQCkkfIQNg5fsxu \
    U23QM5mzz+BxJ4kgF/jiQyBDBvL+XWPE6oCQ==";
};

dig -k ch_zonedata.key @zonedata.switch.ch +noall +answer \
+noidnout +onesoa AXFR ch. > ch.txt
```

La brute À ma grande surprise, la technique décrite par Floyd [13] il y a 13 ans fonctionne encore aujourd'hui. Le transfert de zone (AXFR) est un type de requête DNS utilisée par les administrateurs pour copier une zone d'un serveur à un autre. Ces commandes sont normalement authentifiées (comme dans la commande plus haut) pour n'être accessibles que par des utilisateurs autorisés. Le script suivant énumère grossièrement les serveurs DNS autoritatifs de chaque zone et opère un transfert de zone AXFR.

Listing 1: dump_axfr.sh

```
#!/bin/bash

TLDs=$(curl 'https://data.iana.org/TLD/tlds-alpha-by-domain.txt' |
↪  grep -v '^#')
for tld in $TLDs
do
   echo "Doing TLD $tld"
   for ns in $(dig $tld NS +short)
   do
      #echo "$tld : $ns"
      ip4=$(dig $ns A +short)
      ip6=$(dig $ns AAAA +short)
      for ip in $ip4 $ip6
      do
         #echo "$tld: $ns: $ip"
         FILENAME="${tld}_${ip}.zone"
         dig axfr $tld @$ip > "$FILENAME"
         if grep -q "SOA" "$FILENAME" ; then
            echo "Match for $tld !"
         else
            rm -f "$FILENAME"
            #echo "No match for $tld"
         fi
      done
   done
done

```

Listing 2: Résultat du brute-force AXFR

```
$ ls *.zone
ARPA_170.247.170.2.zone      FJ_144.120.146.65.zone
ARPA_192.112.36.4.zone       FJ_2402:2940:100:100c::1.zone
ARPA_192.203.230.10.zone     FJ_2402:2940:100:100d::1.zone
ARPA_192.33.4.12.zone        GN_41.77.190.237.zone
ARPA_192.5.5.241.zone        GP_193.218.114.34.zone
ARPA_193.0.14.129.zone       MP_16.162.31.128.zone
ARPA_199.7.91.13.zone        MP_16.163.54.122.zone
ARPA_2001:500:12::d0d.zone   MP_75.101.129.89.zone
ARPA_2001:500:2::c.zone      MP_75.101.133.101.zone
ARPA_2001:500:2d::d.zone     MW_196.45.188.5.zone
ARPA_2001:500:2f::f.zone     MW_196.45.190.9.zone
ARPA_2001:500:a8::e.zone     MW_41.221.99.135.zone
ARPA_2001:7fd::1.zone        MW_41.87.2.154.zone
ARPA_2801:1b8:10::b.zone     MW_41.87.5.162.zone
BW_168.167.98.226.zone       NI_186.1.31.8.zone
ER_196.200.96.1.zone         XN--54B7FTA0CC_180.211.212.213.zone
ER_196.200.96.2.zone         XN--54B7FTA0CC_2407:5000:88:2::3.zone
FJ_144.120.146.1.zone
```

C'est l'approche prise par TLDR-2 [2], et au vu des archives disponibles sur Github, ça a l'air de fonctionner.

Le truand Nous l'aurons compris, la majeure partie des TLD devront être extraits par des techniques de Zone Walking.

Un petit rappel sur DNSSEC. DNSSEC (Domain Name System Security Extensions), standardisé en 1999, est une extension du protocole DNS conçue dans le but de rajouter une couche de sécurité au vénérable DNS. DNSSEC signe cryptographiquement les entrées DNS des zones afin d'empêcher un acteur malicieux d'en modifier le contenu sur le chemin. DNSSEC se base sur une hiérarchie dans laquelle les TLD peuvent déléguer la signature des zones aux domaines de second niveau. La conception de DNSSEC vise à protéger l'intégrité mais non la confidentialité des données des zones. Cependant une vulnérabilité sur la confidentialité des zones, connue sous le nom de Zone Walking, existe dans les premières versions de DNSSEC.

La première technique de Zone walking abuse la fonction NSEC de DNSSEC, décrite dans RFC 3845 [15].

Prenons une requête dns pour 'nexistepas.ch' en envoyons la au serveur autoritatif de la zone *.ch* :

```
$ dig nexistepas.ch A +dnssec @a.nic.ch

; <<>> DiG 9.18.18-0ubuntu0.22.04.1-Ubuntu <<>> nexistepas.ch A
↪  +dnssec @a.nic.ch
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 9005
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 6, ADDITIONAL:
↪  1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1232
;; QUESTION SECTION:
;nexistepas.ch.     IN  A

;; AUTHORITY SECTION:
ch.     900 IN  SOA a.nic.ch. dns-operation.switch.ch. 2024020400
↪  900 600 1209600 900
ch.     900 IN  RRSIG SOA 13 1 900 20240304221506 20240203220143
↪  30091 ch.
↪  Q2WrYq2qw5Bj51Idynu1s7G/+p+t0YUeRzxXKxlpc+dmDA664TTjTSSp
↪  Pb2ACr8pnPZN7krXZkR5yA5WQxZ9kg==
ch.     900 IN  NSEC  0-0.ch. NS SOA RRSIG NSEC DNSKEY
ch.     900 IN  RRSIG NSEC 13 1 900 20240304141705 20240203140142
↪  30091 ch.
↪  hBT85gA6WAknVCt4+2Pg+DvNfJ4gSz4D2EdnD2BXnU2BFdGFoHvxW+/P
↪  eb8pacxfq7G7U40DwbdOPhCeFIDJVA==
nexiskill.ch.   900 IN  NSEC  nexisvizzera.ch. NS RRSIG NSEC
nexiskill.ch.   900 IN  RRSIG NSEC 13 2 900 20240226035542
↪  20240127030211 30091 ch.
↪  jH7kPR+Fxmt9oUofkKdGp0iI2ZvvZv5M/w/mE4eor3hnPD5CCTA1U4xq
↪  wviTKyUTolEkLBPOuyr+NwDrouwRuw==
[...]
```

Dans cette réponse, on apprend qu'il s'agit d'un NXDOMAIN (le domaine demandé n'existe pas), ainsi que trois paires de réponses. Chaque réponse est accompagnée d'un RRSIG qui constitue une signature cryptographique de la réponse. Ces RRSIG, bien qu'essentiels pour la l'intégrité et l'authentification du protocole, ne nous sont d'aucune utilité. Le dernier enregistrement NSEC est le plus intéressant : il indique qu'il n'existe pas de noms de domaine entre "nexiskill.ch" et "nexisvizzera.ch". Implicitement, cela veut dire que ces deux noms de domaine existent. Un scanner NSEC tentera de trouver deux noms de domaine classés avant et après, pour obtenir une nouvelle réponse du serveur DNS, divulguant de nouveau une limite aux noms de domaines qui n'existent pas dans la zone.

```
$ dig nexisvizzeraa.ch A +dnssec @a.nic.ch | grep "IN    NSEC"
ch.     900 IN  NSEC  0-0.ch. NS SOA RRSIG NSEC DNSKEY
nexisvizzera.ch.  900 IN  NSEC  nexiswiss.ch. NS DS RRSIG NSEC
```

Un scanner NSEC efficace effectuera au minimum n requêtes pour couvrir en totalité une zone de n entrées.

Les faiblesses de NSEC ayant été un frein considérable à l'implémentation de DNSSEC, une solution a été mise en œuvre via la RFC 5155 [9], qui mandate une forme de hashage pour les zones qui le supportent. Le principe général de NSEC est gardé, sauf que le serveur DNS ne conserve que des hash (basés sur SHA1 + base32) des bornes entre chaque nom de domaine existant dans sa zone. Lorsqu'un client effectue une requête pour un nom de domaine inexistant, le nom est hashé puis comparé selon les mêmes rêgles que pour NSEC :

```
$ dig nexistepas.be. A +dnssec @b.nsset.be
[...]
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 41310
[...]
;; AUTHORITY SECTION:
n83q05hv1lpj19e1m5s6kph55sc5lh66.be. 600 IN NSEC3 1 1 0 -
↪  N840INOT5TF2DS37ILFA0FGOT0R535HO NS DS RRSIG
n83q05hv1lpj19e1m5s6kph55sc5lh66.be. 600 IN RRSIG NSEC3 8 2 600
↪  20240215005951 20240124150612 62823 be.
↪  dEG3BEoRWf3oRVl4Y5xvjcto00Qnl0E/yfl5kXviHxBelKkGbnLFQJfC
↪  p6jOcjElBe+sOcXBNuiX7jkzDJ0CKhcQisRggBzDof0EWQGJ+kPSRlQE
↪  MpO/wZGevLRHVI8z9mFEFbFZMQiLg52mauey02+FwS4Zy7VYaEHIGy01 A0w=
pdrpqgs0t19r8qul7c2h4bhnb7agh1b8.be. 600 IN NSEC3 1 1 0 -
↪  PDRRMAHC5LLBO4DHP1J3TFP2JERSM6G0 NS SOA RRSIG DNSKEY NSEC3PARAM
pdrpqgs0t19r8qul7c2h4bhnb7agh1b8.be. 600 IN RRSIG NSEC3 8 2 600
↪  20240214204813 20240123183525 62823 be.
↪  nPnQHEmJBECYie4Yo45Z12SYS+zCTzMJ1VNeoH5Fok2hTHgTp5MAsB9B
↪  QadbBiZKYxr+5t8Rowfj6pZYXA7HUD/zcnhcIKJWhJEwW1CqxdPg5Y5/
↪  W26qW0HyucC+qxqJpmfjLmvQy07qHGnNC1gsFcMiUCUnHb7/XOmaxj7y p+Q=
e43jlmfjm7b0ob359cku1oek7gqqrdgd.be. 600 IN NSEC3 1 1 0 -
↪  E43KEJDV5NITCUR0E43HTLAGRLHJPV44 NS DS RRSIG
e43jlmfjm7b0ob359cku1oek7gqqrdgd.be. 600 IN RRSIG NSEC3 8 2 600
↪  20240224010526 20240202102556 62823 be.
↪  ERjSUn0EWUUDTKIsLpLcI5M7fsyrxHhpQLEHV0TlOujrvx4ZZ7Z2/0pv
↪  ODoMpoZW3BddsxbMQ3iUayy+/IfR/q8NzoymZ8P3JuhIfvwsLz3Ppg//
↪  kk2tFcXq3XWgJhU8sTMdyqbB8UqOZDQvqORobhkHH9QV+lUNmxMRmzYy hNE=
[...]
```

Les informations importantes à regarder dans ce cas-ci sont les enregistrements NSEC3, en particulier celui-ci :

```
n83q05hv1lpj19e1m5s6kph55sc5lh66.be. 600 IN NSEC3 1 1 0 -
↪  N840INOT5TF2DS37ILFA0FGOT0R535HO NS DS RRSIG
```

Cet enregistrement indique que "nexistepas.be" est situé entre n83q05hv1lpj19e1m5s6kph55sc5lh66 et N840INOT5TF2DS37ILFA0FGOT0R535HO, et que la zone est protégée en "NSEC3 1 1 0 -". Ces champs correspondent respectivement au type de hashage (1=SHA1), aux flags (1=opt-out), le nombre d'itérations de hashage, et le salt (- = aucun). Nous pouvons vérifier que "nexistepas.be" hashé est bien situé entre ces deux bornes.

```
$ ./fastnsec3/nsec3hash.py nexistepas.be 0
nexistepas.be: n83v1t44d13qre431gkpm07n1r92soa4 [...]
```

Un scanner efficace doit effectuer au minimum n requêtes pour extraire n hashs, mais également précalculer un nombre conséquent de hashs pour pouvoir tomber entre des noms de domaines existant. Un TLD comme *.com* compte 6,8 millions de domaines protégés en NSEC3. Cela veut dire qu'en moyenne, chaque nom de domaine est distant d'un autre de $\frac{1}{6.8\times10^6}$ soit de $-log_2(\frac{1}{6.8\times10^6}) = 22.7$ bits, ce qui veut dire que la difficulté pour trouver un hash aléatoire entre deux hashs connus requiert environ $2^{22.7}$ opérations de hashage. Cette valeur est une moyenne, et un scanner sera limité par les intervalles les plus petits de la zone, qui peuvent aller jusqu'à 2^{45} opérations nécessaires en pratique.

2 État des lieux

J'ai fait tourner plusieurs scripts Python, pour collecter les paramètres SOA et DNSSEC de chaque TLD sur une période d'environ trois mois et les stocker dans une base de donnée. Ceci pour tenter de comprendre quel est le paysage de la protection des TLD vis à vis des attaques de zone walking. Une majorité écrasante de TLDs utilisent NSEC3 (Tableau 1). 13 TLDs implémentent DNS de façon à ne pas utiliser NSEC ou NSEC3, ou d'une manière qui ne permet pas le zone walking avec l'algorithme de base.

La plupart des TLD ne changent jamais de paramètres NSEC3. Certains changent le salt toutes les semaines, et deux TLD (.by et .xn–90ais) les changent toutes les 20 minutes ! (Tableau 2)

La grande majorité des TLDs n'utilisent pas de salt et un nombre d'itération très faible (0 ou 1). Une série de TLDs, manifestement gérés

Type de DNSSEC	Nombre de TLD
Pas de DNSSEC	196
NSEC	48
NSEC3	1299
Autre	13
Total	1448

Tableau 1. Distribution de DNSSEC dans les TLD

par la même entité, utilisent 100 itérations mais le même salt.

Ces paramètres n'ont pas d'effet concret pour protéger les zones, nous le verrons plus loin.

Une grande majorité de TLDs utilisent le opt-out flag (table 3). Ce champ, lorsqu'il est activé, autorise la zone DNS à fournir des sous-délégations non sécurisées et non signées. Cela veut dire que dans l'exemple "IN NSEC3 1 1 0 -" vu plus haut, la zone DNS ne fournira pas de preuve d'inexistence pour les noms de domaines non sécurisés. Le but est de réduire la quantité d'entrées à signer dans les énormes zones. Ces noms de domaines ne seront pas visibles par le Zone Walker, c'est donc une limitation assez importante de la technique car les très grosses zones utilisent le Opt Out.

3 Dumper NSEC3 Malifar

Le dumper NSEC3 Malifar [3] a été programmé en python avec asyncio. Les objectifs étaient les suivants :

- Programmation asynchrone pour les performances.
- Utilisation de TCP pour minimiser le nombre de flux parallèle sur chaque serveur.
- Configuration par TLD dans un fichier de configuration.
- Système de caching pour stocker les données de hashing intermédiaire afin de pouvoir répéter le listage quotidiennement.
- Stockage efficace des résultats pour le cracking.
- Aperçu temps réel de l'état du listage.
- Accélération du cracking des pré-hash par une machine tièrce avec GPU.
- Listage le plus "linéaire" possible pour minimiser l'empreinte mémoire.
- Fonction stop/restore.

[3] Malifar est une référence obscure à un jeu vidéo

TLD	Temps médian entre changement de Salt
by	0 days 00 :18 :05.357738
xn–90ais	0 days 00 :18 :05.383759
xn–kput3i	0 days 00 :18 :09.714327
ls	0 days 00 :34 :22.782460
xn–3ds443g	0 days 00 :35 :41.415095
ms	0 days 00 :35 :58.916927
xn–vuq861b	0 days 00 :53 :43.579054
pe	0 days 00 :55 :34.559593500
md	0 days 00 :55 :35.587601
xn–kprw13d	0 days 07 :00 :04.376266
mil	0 days 10 :07 :28.504282
mx	0 days 23 :58 :07.941317
py	0 days 23 :59 :23.966294
lt	1 days 00 :00 :06.458687500
cl	1 days 00 :00 :45.645737500
ua	1 days 00 :01 :09.598401500
xn–mgbayh7gpa	2 days 03 :04 :27.313133
xn–mgberp4a5d4ar	4 days 23 :59 :35.851292
sa	5 days 00 :02 :51.921838
sg	5 days 23 :59 :10.148251
xn–clchc0ea0b2g2a9gcd	6 days 00 :07 :01.170658
xn–yfro4i67o	6 days 00 :07 :11.568331
tm	7 days 00 :03 :06.363283
si	9 days 00 :14 :09.630126
xn–mgbt3dhd	9 days 23 :54 :20.291070
shia	9 days 23 :56 :58.171910500

Tableau 2. Temps médian de changement de salt (top 25)

Malifar se différencie de *nsec3map* principalement par sa couche réseau asynchrone, le système de caching, le cracking GPU et l'empreinte mémoire. Porter ce logiciel à *asyncio* et rajouter les autres fonctionnalités aurait probablement pris plus de temps que l'implémentation de zéro.

Malifar est disponible sur Github [4] sous license MIT. Il nécessite *swig* pour la compilation des modules, et *pyopencl* pour le cracker GPU.

3.1 Caching

Afin de pouvoir lister régulièrement la même zone pour y découvrir des différences, il était nécessaire de pouvoir stocker tous les hashs intermédiaires générés par le GPU. Pour cela, j'ai recyclé une ancienne librairie C (*libhap*) écrite lors d'un projet différent. Cette librairie implémente une base de donnée binaire très simple et rapide à base de fichiers mappés en mémoire. Cette même librairie est aussi utilisée pour stocker les résultats du listage. Cette librairie permet de chercher des clefs entre deux

Opt-out	Nombre de TLDs
Activé	1118
Désactivé	184

Tableau 3. Nombre de TLDs NSEC3 utilisant le Opt Out

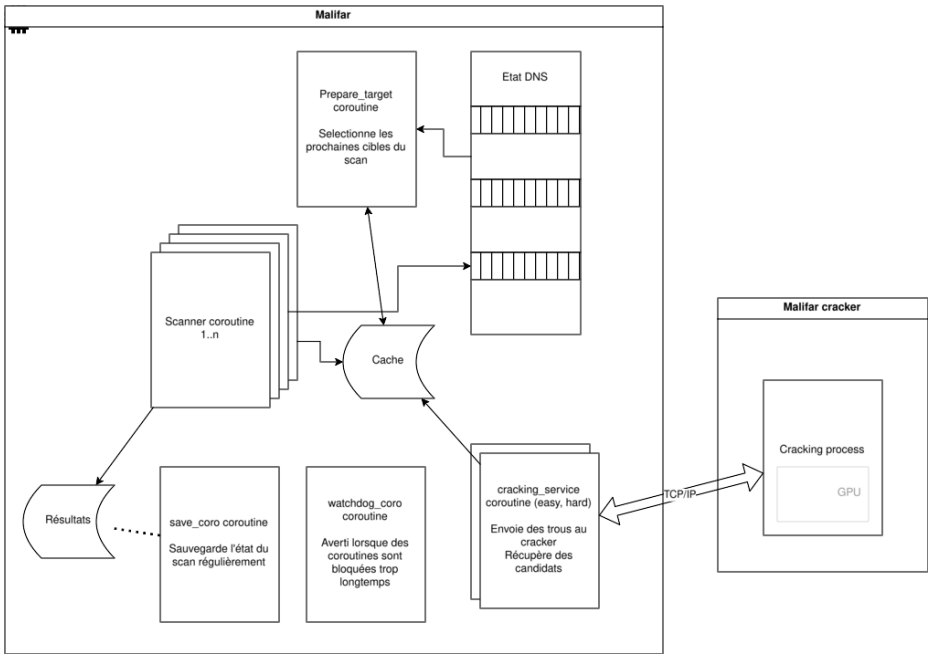

Fig. 1. Architecture de Malifar

bornes (ce qui est plutôt coûteux avec *sqlite*) et évite une dépendance à une instance *Redis*. Son implémentation par fichiers mappés en mémoire permet de rendre l'ouverture et l'ajout de données très rapides tout en consommant un minimum de RAM en structures de données.

Il y a donc deux fichiers produits par Malifar : un cache de valeurs intermédiaires et un fichier de résultats. Le cache contient une liste valeur :donnée, classée en ordre croissant pour permettre une recherche dichotomique. Chaque cache est unique à un seul TLD à cause des paramètres de hashage uniques, mais le cache peut être réutilisé lors des scans ultérieurs, diminuant la nécessité du cracking GPU.

Le fichier de résultat contient quand à lui uniquement une liste de hashes nsec3 récoltés par le listage. Ce fichier peut être converti en format hashcat avec le script *haptool.py*.

3.2 Cracking GPU

Lorsque l'on liste une zone NSEC3, on obtient des paires de hashs de noms qui existent dans la zone. Par exemple, on obtient (a, b) et (f, g). L'algorithme de recherche établit qu'il n'y a rien entre a et b et entre f et g, mais il y a un espace inconnu entre b et f. Les hash b et f sont d'abord convertis en nombres binaires de 64 bits (car 64 bits suffisent largement et permettent de gagner 12 bytes sur un hash SHA1). Ces limites sont envoyées par paire (b, f) au service réseau de cracking.

La coroutine *prepare_target* établit, à partir de l'état interne DNS, une liste de "trous" qui n'ont pas encore de résolution, c'est à dire qu'il n'y a pas de valeur connue dont le hash se situe dans ce trou.

La coroutine *cracking_service* récolte un nombre de ces trous (sous la forme de paire de nombres 64 bits) et les envoie au service de cracking au moyen d'une connexion TCP. Le service de cracking répète l'algorithme de hashage avec les paramètres NSEC3 de la zone DNS, jusqu'à trouver des candidats de noms de domaines qui rentrent dans les trous. Ces noms de domaines sont ensuite réutilisés par l'algorithme de recherche pour découvrir de nouveaux trous ou au contraire clôturer tout un espace de recherche. L'algorithme se termine lorsqu'il n'y a plus de tous à résoudre.

Pour rendre la recherche plus rapide, deux processus de crackings sont démarrés. Un pour les "grands" trous (moins de 30 bits) et un pour les "petits" trous (plus de 30 bits). À 2GH/s sur une RTX4090, certains trous de 40 à 45 bits de la zone .com prennent plus d'une heure à résoudre.

L'intérêt de déporter le cracking GPU sur un service TCP est de pouvoir faire le listage depuis une machine légère dans le cloud et le cracking sur une machine physique locale, car le prix des instances GPU est très élevé.

Le cracking GPU lui-même est implémenté en partie en *OpenCL* via *pyopencl* et en python. Une implémentation de SHA-1 sous license MIT a été empruntée et instrumentée pour réaliser des milliers de SHA-1 en parallèle. En comparaison des performances de Hashcat qui réalise environ 15GH/s sur du bruteforce SHA-1 classique, il y a une marge de progression de x7 encore réalisable. Cela reste néanmoins des milliers de fois plus rapide que l'approche CPU.

4 Cracking des zones

Une fois une zone extraite, il est nécessaire de cracker les noms de domaine comme des mots de passe. Cela se fait avec un outil comme Hashcat.

À ma grande surprise, le cracking de noms de domaines n'est pas si facile qu'il n'y parait. Il n'y a pas de dictionnaires tout faits sur internet, et même si l'algorithme est extrêmement rapide - 15GH/s sur mon matériel - et que l'espace de recherche se limite aux caractères "a-z 0-9 -", la recherche exhaustive jusqu'à 11 caractères ne libère qu'environ un quart des .com.

Le dictionnaire *crackstation* couplé aux rules intégrées à Hashcat a permis d'arriver en moins d'une journée à environ 50%. Cependant *crackstation* contient beaucoup de majuscules et caractères spéciaux qui ne sont pas dans l'espace de recherche, et les rules par défaut ne tirent pas profit de ces limitations.

Quelques stratégies permettent d'augmenter la quantité de noms récupérés :

— Enrichir le dictionnaire à l'aide de noms de domaines existant réellement. Cela inclut des données obtenues via d'autres méthodes : le bon ou la brute, les extractions NSEC (donc non protégées), la collecte de noms via d'autres services en ligne et registres de transparence TLS.
— Modifier des dictionnaires existants pour les adapter à la syntaxe DNS : "a-z 0-9 -".
— Créer des "rules" hashcat pour combiner des mots entre eux, par exemple pour créer des phrases telles que "ceciestunexemple.com" ou "ceci-est-un-exemple.com"
— Prendre en compte les caractères spéciaux au format ACE, c'est à dire composés du préfixe "xn–" et d'un punycode, conversion en alphabet latin de caractères spéciaux. Cela peut se faire à l'établissement des dictionnaires ou avec des "rules" hashcat. Ces noms sont minoritaires mais particulièrement importants pour les TLD de pays dont l'écriture n'est pas latine et pour découvrir des typosquatting de noms latins.
— Chercher des noms de domaine composés, tels que exemple.co.uk, qui sont dans la zone .uk et non .co.uk.

À l'heure de la finalisation de cet article, l'état de cette étape et malheureusement encore très partiel.

5 Mitigation

5.1 Mitigations découvertes

Certains TLD listés ont présenté quelques mitigations, dont voici les plus importantes :

Rotation de salt Le paramètre salt est modifié régulièrement. Par exemple, le TLD *by* modifie le salt toutes les 20 minutes. Cela est très gênant lorsque cela arrive en plein milieu d'un scan. La solution contre cette mitigation est de commencer le scan immédiatement après le changement et de faire en sorte que le scan dure moins de 20 minutes.

Limitation du nombre de requêtes Un nombre important de serveurs DNS limitent le nombre requêtes permises par connexion TCP, généralement aux alentours de 20. Ces serveurs ont également une durée limite par connexion, généralement 5 secondes. La solution est de scanner à l'avance (script *probe_nameservers.py*) et de modifier la configuration de Malifar pour ce TLD, afin de couper la connexion au serveur après que les limites soient atteintes et éviter d'arriver en situations d'erreurs.

Limitation du nombre de flux Une protection réseau contre les déni de service consiste à limiter le nombre de flux entrants pour chaque serveur. Cette situation ne s'est jamais produite lors des scans, car un maximum de deux connexions TCP par serveur est utilisé. Pour les serveurs accessibles par IPv6,[4] une adresse IPv6 double le nombre de connexions possibles par serveur. L'utilisation de TCP à la place de DNS permet aussi de diminuer le nombre de requêtes vues par des outils tels que netflow.

Nombre d'itérations Le paramètre "Iterations" indique combien de répétitions de SHA-1 sont nécessaires pour hasher un nom de domaine. La plupart des TLD utilisent la valeur 0, mais des valeurs couramment utilisées sont 1, 5, 10 et 100. Cela rend la partie hors-ligne du brute-force des noms de domaine plus lente, mais ne ralentit que très peu la partie en ligne car ces zones sont relativement petites. Cette solution est peu prisée par les grosses zones, car le temps nécessaire pour compiler le fichier zone et le signer doit idéalement être le plus court possible.

Erreurs de configuration et d'accessibilité Certains TLD ont des configuration défectueuses ou souffrent de problèmes de réseau récurrents. Par exemple, certains serveurs autoritatifs indiqués dans le SOA ne répondent pas correctement aux requêtes ou ne sont pas accessibles. Ces serveurs sont retirés de la liste à l'aide du probing.

[4] Contrairement à ce qu'on pourrait attendre, tous les TLD ne sont pas encore accessibles en IPv6

5.2 Mitigations standardisées

La RFC9276 [12] "Guidance for NSEC3 Parameter Settings" donne quelques conseils pour la protection des zones DNS. Nous y retrouvons ces mots :

> NSEC3 records are created by first hashing the input domain and then repeating that hashing using the same algorithm a number of times based on the iteration parameter in the NSEC3PARAM and NSEC3 records. The first hash with NSEC3 is typically sufficient to discourage zone enumeration performed by "zone walking" an unhashed NSEC chain.

La RFC avoue à demi-mot que la protection contre le zone walking n'est que peu efficace, et recommande d'utiliser les paramètres suivants :

> In short, for all zones, the recommended NSEC3 parameters are as shown below :
> ; SHA-1, no extra iterations, empty salt :
> bcp.example. IN NSEC3PARAM 1 0 0 -"

La RFC4470 [11] de 2006 propose une technique basée sur NSEC pour forger des réponses à la volée qui "encerclent" la requête de l'attaquant. Cette technique, aussi souvent appelée "White Lies", nécessite que le serveur DNS possède la clef privée de la zone, ce qui est souvent contraire aux bonnes pratiques de gestions de serveurs DNS.

Le draft "Compact Denial of Existence in DNSSEC" [10] est une formalisation de techniques déjà appliquées par Cloudflare sous le nom de "Black Lies" [8]. Il s'agit d'une optimisation de la technique "White Lies" qui se concentre en particulier sur la gestion des wildcards dans les zones complexes.

5.3 Découpage statique de hashs

Aucune de ces mitigations ci-dessus n'a été trouvée sur les serveurs scannés jusqu'à présent. Ce mitigations ont le défaut d'être peu compatibles avec des zones critiques nécessitant de multiples copies sur des caches non gérés par la même organisation. Il y a cependant la possibilité d'implémenter une version statique (signée hors ligne) de White Lies sur NSEC3 en faisant quelques compromis. Prenons par exemple le cas de la requête NXDOMAIN pour .be :

```
1 nexistepas.be: n83v1t44d13qre431gkpm07n1r92soa4
2
3 n83q05hv1lpj19e1m5s6kph55sc5lh66.be. 600 IN NSEC3 1 1 0 -
  ↪  N840INOT5TF2DS37ILFA0FGOT0R535HO NS DS RRSIG
```

Les deux valeurs n83... et n84... correspondent aux hashs de deux noms de domaine existant dans le DNS. En rajoutant soigneusement des valeurs fictives autour de ces deux hashs, on peut forcer l'algorithme de zone walking à générer des hashs intermédiaires de plus en plus compliqués pour tenter de s'approcher d'un hash existant :

```
n83q05hv20000000000000000000000.be.
n83q05hv1lpj20000000000000000000.be.
n83q05hv1lpj19e1n000000000000000.be.
n83q05hv1lpj19e1m5s6l00000000000.be.
n83q05hv1lpj19e1m5s6kph55sc5lh66.be.
n83q05hv1lpj19e1m5s6k00000000000.be.
n83q05hv1lpj19e1m000000000000000.be.
n83q05hv1lpj10000000000000000000.be.
n83q05hv10000000000000000000000.be.
```

Ces valeurs pourraient être choisies de façon arbitrairement proches du hash d'origine (par exemple 48 bits, 64 bits) pour empêcher l'algorithme de zone walking de trouver des trous, mais aussi rendre le cracking des noms de domaines plus compliqué, en forçant une approche mixte offline/online, car un hash de 48 bits n'est pas suffisant pour cracker un nom de domaine de façon sûre. Elle implique une multiplication au minimum par 3 de la taille de la zone DNS, ce qui est un point de contention au niveau du déploiement des larges zones.

Cette approche nécessite des recherches plus approfondies pour valider sa pertinence et validité. Elle est aussi semblable dans l'idée à celle de Arnob Paul et al [5] qui se concentre sur une approche similaire sur NSEC.

5.4 L'approche pragmatique

Ma recommandation serait plutôt de suivre l'exemple de SWITCH qui a passé ses zones DNS en open data. Tenter de protéger indéfiniment le contenu des zones est malheureusement voué à l'échec, et va à contre-courant de ce qui se fait dans le domaine des certificats TLS, qui vise l'ouverture pour pouvoir détecter au plus vite les actions des acteurs malveillants de l'Internet.

6 Travaux futurs

Quelques objectifs restent encore à atteindre. Quelques points qui méritent encore de l'attention :

— Scripter la collecte quotidienne des DNS.

— Scripter le cracking permanent des hashes.
— Créer un tableau de bord web permettant de visualiser des données sur la collecte des noms de domaine, par exemple pour voir les différences (additions/suppressions) et taux de cracking.
— Éventuellement une interface permettant d'interroger la base de données, par exemple pour rechercher du typosquatting.

Références

1. nsec3map - DNSSEC Zone Enumerator. `https://github.com/anonion0/nsec3map`, 2024.
2. TLDR 2 - A Continuously Updated Historical TLD Records Archive. `https://github.com/flotwig/TLDR-2`, 2024.
3. zone-walks. `https://github.com/flotwig/zone-walks`, 2024.
4. Aris Adamantiadis. Malifar NSEC3 scanner. `https://github.com/arisada/malifar`, 2024.
5. Shohrab Hossain Husnu S. Narman Arnob Paul, Hasanul Islam. A Novel Zone-Walking Protection for Secure DNS Server. *IGI Global*, 2022.
6. BenBE. DNSSEC NSEC3 hash interval search #3599. `https://github.com/hashcat/hashcat/issues/3599`, 2023.
7. D.J. Bernstein. Breaking DNSSEC. `https://cr.yp.to/talks/2009.08.10/slides.pdf`, 2009.
8. Cloudflare. Black Lies. `https://blog.cloudflare.com/black-lies`.
9. B. Laurie et al. RFC 5155 DNS Security (DNSSEC) Hashed Authenticated Denial of Existence . `https://datatracker.ietf.org/doc/html/rfc5155`, 2008.
10. S. Huque et al. Compact Denial of Existence in DNSSEC. `https://datatracker.ietf.org/doc/draft-ietf-dnsop-compact-denial-of-existence/`, 2023.
11. S. Weiler et al. RFC 4470 Minimally Covering NSEC Records and DNSSEC On-line Signing. `https://datatracker.ietf.org/doc/html/rfc4470`, 2006.
12. W. Hardaker et al. RFC 9276 Guidance for NSEC3 Parameter Settings. `https://datatracker.ietf.org/doc/html/rfc9276`, 2022.
13. Floyd. DNS zone transfer. `https://www.floyd.ch/?p=344`, 2011.
14. ICANN. Centralized Zone Data Service. `https://czds.icann.org/help`, 2024.
15. Ed. J. Schlyter. RFC 3845 DNS Security (DNSSEC) NextSECure (NSEC) RDATA Format. `https://datatracker.ietf.org/doc/html/rfc3845`, 2004.
16. SWITCH. SWITCH Open Data. `https://portal.switch.ch/pub/open-data/#tab-fccd70a3-b98e-11ed-9a74-5254009dc73c-3`, 2024.