LINFO1341 - Analyse de Shadow Drive

Antonutti Adrien
31202100

Bokungu Nathan
31232100

I. Introduction

Le but de ce projet est d'analyser le fonctionnement réseau d'une application de transfert et de partage de fichiers. Notre analyse se portera sur l'application **Shadow Drive**. Elle commencera par un tour d'horizon de l'application, ensuite une analyse du protocole DNS de l'application sera faite. Enfin elle abordera plusieurs scénarios en analysant les protocoles utilisés et leurs différentes utilités.

Le lien du github du projet contenant les traces et les scripts utilisés pour ce rapport est disponible ici.

II. Scénarios

Voici les différents scénarios de l'application que nous allons analyser :

- Transfert d'un fichier et création d'un dossier
- Modification d'un fichier déjà existant
- Déplacement d'un fichier
- Suppression d'un fichier
- Modification du type de connexion : Wi-Fi, Ethernet ou partage 4G

III. Analyse DNS

Lors du lancement de l'application (native ou web), l'application résoud les noms de serveurs pour connaître les adresses IPs du/des serveur(s) avec qui échanger des données. Un nom de domaine est résolu : *drive.shadow.tech*. Deux requêtes de type *query* sont envoyés : un record A pour l'IPv4 et un record AAAA pour l'IPv6. Chaque requête obtient une réponse.

La réponse au record A renvoie 2 adresses IP : *46.105.132.157* et *46.105.132.156*, toutes les requêtes de l'application se feront par la suite sur une de ces 2 adresses, qui est choisie au démarrage de l'application. Nous supposons que ce choix est fait de manière à équilibrer la charge sur les deux serveurs.

La réponse au record AAAA ne contient pas de champ *Answers*, on conclut donc qu'aucune adresse IPv6 n'est liée à ce nom de domaine. Le nom du serveur autoritatif est donné : *candy.ns.cloudflare.com*, qui s'occupe de gérer le domaine shadow.tech. On peut confirmer cela en faisant cette commande : *dig shadow.tech +nssearch*, qui va tenter de récupérer le nom des serveurs autoritatifs liés à ce nom de domaine. Le résultat de cette commande est une liste de record DNS SOA (Start Of Autority). On a 8 records de ce type, dont chacun vient d'un serveur différent avec une adresse IP différente. Ce record contient plusieurs champs :

- **Mname** : *candy.ns.cloudflare.com* - Nom du serveur de nom principal pour cette zone.
- **Rname** : *dns.cloudflare.com* - Mail du responsable du domaine.
- **Serial** : *2337391626* - Représentation numérique de la version du record, ce numéro modifié à chaque nouvelle version.
- **Refresh** : *10000* - Durée en secondes que les serveurs secondaires doivent attendre avant de demander au serveur primaire un nouvel enregistrement SOA.
- **Retry** : *2400* - Délai qu'un serveur doit respecter avant de renouveler une demande d'actualisation après un échec.
- **Expire** : *604800* - Délai maximal avant qu'une zone ne soit considérée comme n'ayant pas d'autorité.
- **TTL** : *1800* - La durée maximale qu'un serveur peut garder ce record en cache.

TABLE I: Réponses DNS

Nom	Type	TTL	IP/Resultat
drive.shadow.tech	A	61	46.105.132.156
drive.shadow.tech	A	61	46.105.132.157
drive.shadow.tech	AAAA	/	/

Afin de sécuriser les requêtes DNS, les solutions utilisées par Shadow Drive sont le fait de lier un différent port source UDP pour chaque requête et de répondre en envoyant les queries et les adresses IP correspondantes. Il existe d'autres méthodes proposées pour sécuriser les records DNS, comme par exemple avoir plusieurs serveurs autoritatifs, répondre à la fois via IPv4 et IPv6 ou utiliser DNSSEC. L'application n'emploie aucune d'entre-elles.

Lors d'une nouvelle connexion à Shadow Drive, on observe des requêtes vers d'autres serveurs. Les domaines *auth.eu.shadow.tech.static.cloudflareinsights.com* et *challenges.cloudflare.com* sont résolus et sont utilisés pour le processus de connexion de l'utilisateur.

A. TTL

TTL vient de l'acronyme *time to live* et indique une durée pendant laquelle le client peut garder l'adresse IP associée à ce nom de domaine en cache. Une fois cette durée passée, il faut résoudre à nouveau le nom de domaine afin de vérifier que l'IP associée n'ai pas été modifiée. Dans notre cas, les adresses IPv4 ont un TTL de 61 secondes.

IV. Différence entre application web et native

Nous n'avons pas remarqué de grande différence entre le fonctionnement web et natif de l'application. Les serveurs

utilisés sont les mêmes et les requêtes sont similaires. Dans la suite de notre analyse nous nous sommes plus basés sur la version web, car en stockant les secrets échangés avec *SSLKEYLOGFILE* via le navigateur, le decryptage des paquets cryptés via TLS est possible, ce qui permet une analyse détaillée.

V. Protocoles utilisés

Les principaux protocoles que l'on a pu observés en dehors de DNS sont : TCP pour le transfert des données, qui est securisé à l'aide de TLS. Lorsqu'on décrype les paquets TCP, on remarque que des requêtes HTTPS sont égalements faites vers le serveur. On observe pas de trafic QUIC ou UDP directement. On observe un tout petit peu de UDP, causé par les requêtes DNS qui sont basées sur UDP.

A. TCP

TCP est utilisé pour le transport des informations de l'application, principalement à cause du fait que HTTP est basé sur TCP et que l'application utilise HTTP pour le transport des données. On observe que le client ne communique qu'avec un seul serveur pour échanger des données, ce qui a facilité notre analyse. Il choisit entre les 2 serveur que le DNS a pu trouvé, on imagine que le choix est fait pour équilibrer au maximum la charge entre les 2 serveurs. A chaque établissement de connexion on observe le 3-Way hanshake typique de TCP (SYN-SYN/ACK-ACK), comme le montre la Figure 1

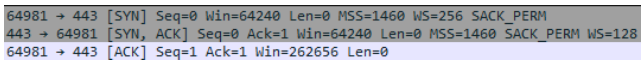


Fig. 1: TCP 3-Way hanshake

1) Fermeture de connexion: Habituellement, un *graceful release* est observé pour la fermeture de connexion. Cependant, lorsque l'application native est fermée manuellement, l'envoi d'un paquet *RST* (reset) est observé, ce qui indique un *abrupt release* (voir Figure 2).

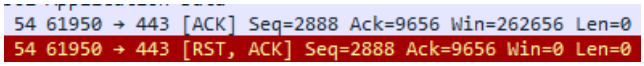


Fig. 2: Abrupt release avec flag RST

B. TLS

La version TLS utilisée est TLS 1.3, cette information se situe dans le champ *supported_version* du *Server Hello*. TLS 1.2 apparaît également dans Wireshark, mais uniquement dans un soucis de rétro-compatibilité des différentes versions TLS. Les certificats n'apparaissent pas dans Wireshark, mais nous avons utilisé SSLLabs pour avoir des informations. Le certificat est fourni par *Sectigo RSA*, qui est une entreprise connue pour fournir ce genre de certificat. Le certificat est valide depuis le 30 octobre 2023 et a une durée de validité de 1 an. Lors de l'établissement du chiffrement, le *Client Hello* spécifie les algorithmes de chiffrement disponible dans le champ *Cipher Suites*, dans notre cas il y en a 16. Ensuite, dans le *Server Hello*, l'algorithme utilisé se situe dans le champ *Cipher Suite*. Pour Shadow Drive, c'est *TLS_AES_128_GCM_SHA256* qui est utilisé.

C. HTTPS et WebDAV

En faisant nos recherches, nous avons remarqué à plusieurs endroits le nom *webdav* (entre autre dans l'url des requêtes HTTP). Ce nom se retrouve également en faisant des recherches sur les méthodes HTTP non standard comme **PROPFIND**, **MOVE** ou encore **MKCOL**. Ces méthodes non standard sont liées à *WebDAV*, qui est une extension de HTTP, dont le but est d'écrire des applications qui ont pour but de gérer des ressources plus simplement [1].

VI. Analyse des différents scénarios

A. Démarrage

En plus des requêts DNS, plusieurs requêtes HTTP **GET** sont faites pour récupérer le contenu associé à des images, du javascript, des feuilles de style css...

B. Fonctionnement en idle

En laissant l'application tourner dans le vide, on remarque que des requêtes sont faites toutes les 30 secondes vers le serveur.

1) Analyse des requêtes pour la synchronisation:

- Etablissement de la connexion TCP avec un 3-Way Handshake et du chiffrement TLS.
- Le client envoie une première requête HTTP2.0. Celle-ci contient une préface de connection (nommé "magic" dans Wireshark) et qui confirme au serveur qu'on va utiliser HTTP2.0 pour la suite de la connection. Le client envoie 4 paramètres **SETTINGS** : *header table size*; *enable push : 0*; *initial windows size*; *max header list size*. Enfin, la taille de la fenêtre actuelle de transmission **WINDOW_UPDATE** passe de 65535 à 15728640 octets.
- Une requête de type *GET* est faite au serveur sur l'url */ocs/v2.php/apps/notifications/api/v2/notifications*. Cette requête est faite sur l'API de Shadow et permet de vérifier si une notification est disponible pour l'utilisateur ou pas.
- Le serveur renvoie une réponse avec le statut 304 si une notification est disponible.
- Ensuite, après 5 secondes le serveur envoie un **GO_AWAY**, qui annonce la fermeture de la connexion. Enfin, un warning TLS annonce la fermeture de la connexion et la connexion TCP est fermée.

Nous pensions que le but de ces requêtes était de vérifier que le contenu du drive n'a pas été modifié depuis la dernière synchronisation. Nous avons voulu vérifier cela en mettant un fichier depuis un autre ordinateur, mais cela n'a pas modifié le résultat de la réponse (ni ce qu'affichait le site, on ne voyait pas le fichier apparaître sans actualiser la page). Probablement que ces requêtes ponctuelles sont liées au système de notification utilisateur du site (au vu du nom de l'url contactée), mais nous n'avons pas vérifié cette hypotèse.

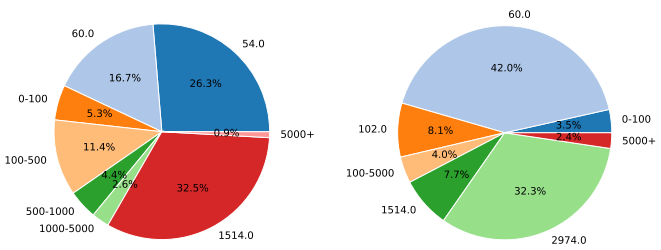
2) NextCloud: Shadow Drive est basé sur une solution de l'entreprise next-cloud. On peut se rendre compte de cela car l'url de la requête vers l'API correspond à celle d'une d'une API de NextCloud. Aussi, dans le header de la réponse, on trouve un en-tête non-standard *x-nextcloud-user-status* qui indique que le logiciel de nextcloud est derrière l'infrastructure de Shadow Drive. Ce header permet, comme son nom l'indique, d'identifier le statut de l'utilisateur.

C. Transfert d'un fichier

1) Analyse des requêtes:

- Etablissement connexion TCP et TLS
- Requête HTTP **PUT** : cette méthode dit au serveur de remplacer le contenu correspondant à cette url par le contenu de la requête. Dans le header on a l'url de la requête : */remote.php/webdav/**nom_du_fichier***. On retrouve également *content-length*, qui contient la taille en bytes du nouveau fichier à uploader. Le champ *content-type* indique le type de fichier qui va être transferé : il vaut *application/octet-stream* pour un fichier binaire (.exe) et *text/plain* pour un simple fichier texte. On retrouve d'autres headers, contenant d'autres informations comme le navigateur utilisé, le nom du fichier transferé, l'OS de l'utilisateur... Mais le but n'est pas de faire une liste exhaustive.
- Réponse du serveur avec le numéro de statut 201, qui signifie *Created*, qui confirme que le transfert du nouveau fichier s'est bien passé.
- Requête du client HTTP/WebDAV du type **PROPFIND** vers l'url */remote.php/dav/files/901d3889-bacc-43f8-acb8-844fa59cb9be/**nom_du_fichier***, la chaîne de caractère avant le nom du fichier correspond est lié à l'utilisateur, c'est une espèce d'identifiant. Le but de cette méthode est de récupérer les propriétés d'une ressource, ici, celle du fichier qui vient d'être uploadé.

Dans le cas de la création d'un dossier, une requête HTTP de méthode **MKCOL** est faite, qui signifie *make collection* permet de créer un dossier. Cette méthode n'est pas standard et provient aussi de l'extension HTTP *WebDAV*.



(a) Petit fichier (14kb) (b) Gros fichier (49mb)

Fig. 3: Répartition de la longueur des paquets en bytes lors d'un transfert

2) Taille des paquets: On observe que énormément de paquet (environs 40%) ont une taille de 54 ou 60 bytes, cela correspond à tous les ACK du protocole TCP renvoyés par le client ou le serveur pour confirmer la réception d'un paquet. Le reste des paquets sont plus volumineux (plus de 1000 bytes), on observe certaines tailles caractéristiques comme 1514 ou 2974 bytes. Ces paquets correspondent au contenu des fichiers et des requêtes. Lorsque la taille des fichiers transferés augmentent, la proportion de paquets plus volumineux augmentent également. Ainsi, aux figures 3a et 3b, pour le petit fichier, on remarque 32.5% des paquets ont une taille de 1514 bytes, tandis que pour le fichier plus gros, 32.3% des paquets ont une taille de 2974 bytes.

3) Compression: Lorsqu'on transfert un fichier sur le serveur, la Table II montre que le trafic est légerement supérieur à la taille du fichier transferé (et ce peu importe la taille du fichier). Aucun algorithme de compression n'est donc utilisé. Cela pourrait être une piste d'amélioration pour diminuer le trafic de donnée, mais cela a le désavantage de prendre du temps et de la puissance de calcul côté client pour la compression et décompression des données.

TABLE II: Quantité de données échangées lors du transfert

Taille du fichier	49MB	8.97MB	14KB
Quantité de données échangées	53MB	10MB	83KB

D. Modification d'un fichier existant

Le but de ce scénario est de modifier un fichier plutôt grand et de voir si l'application reupload le fichier en entier ou envoie uniquement la partie modifiée. Pour vérifier cela, nous avons créé un fichier texte de 44Mb. Nous le modifions et enregistrons les paquets au moment où l'application synchronise les modifications. La version web ne permet pas de modifier les fichiers en ligne directement, nous sommes donc passés par la version native. Par conséquent nous n'avons pas eu accès aux contenus décryptés des paquets. On remarque rapidement que l'application upload le fichier modifié comme un nouveau fichier. En modifiant seulement quelques lettres de notre gros fichier texte, l'application reupload la totalité du fichier. Comme le montre la trace des paquets lors de cette opération, 47Mb de données sont envoyés vers le serveur et que le serveur envoie 1Mb vers le client. L'application elle-même montre qu'elle reupload le fichier en entier, comme le montre la Figure 4. Une piste d'amélioration serait que le client détecte si le fichier est completement nouveau ou si uniquement une petite partie a été modifiée. Dans ce cas, le client pourrait envoyer uniquement les données utiles, et le serveur effectuerait les modifications adéquates.

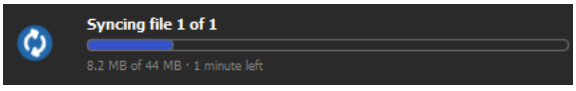


Fig. 4: Reupload d'un fichier modifié

E. Suppression d'un fichier

Lors de la suppression d'un fichier, on observe de façon classique l'établissement de la connexion TCP, le handshake

pour la connexion TLS, la préface de connexion *magic* et les paramètres HTTP. On observe également quelques requêtes HTTP dont la méthode est GET pour récupérer des images au format *svg*, mais pas toujours, cela dépend de si elles sont présentes ou pas dans le cache du navigateur. Voici les éléments plus intéressants que nous avons relevé :

- Requête **GET** du client vers l'url */ocs/v2.php/apps/files_reminders/api/v1/2414193206*
- Réponse du serveur pour cette requête : elle contient un statut 200 (OK) indiquant que tout s'est bien passé, ainsi que du contenu *JSON*, qui contient des métadonnées confirmant que ça s'est bien passé, et une donnée : *dueDate*, mais dont la valeur est nulle, ce qui est un peu étrange.
- Requête **DELETE**, c'est cette requête qui indique au serveur de supprimer le fichier. Cette requête pointe vers */remote.php/dav/files/901d3889-bacc-43f8-acb8-844fa59cb9be/**nom_du_fichier***. La chaîne de caractère avant le nom du fichier est liée à l'utilisateur qui fait la requête (voir points suivants)
- Réponse HTTP du serveur avec le code 204, qui signifie qu'il n'y pas de contenu à renvoyer pour cette requête (ce qui est logique car on a voulu supprimer du contenu). Dans le cas où le fichier a déjà été supprimé (par un autre ordinateur par exemple), le serveur renvoie l'erreur *404 Not Found*, car naturellement il ne retrouve pas le fichier.
- Requête **GET** du client vers */apps/files/api/v1/stats* : le but est de récupérer les stats d'utilisation de l'espace de stockage du client.
- Réponse HTTP du serveur : elle contient le statut OK de la requête ainsi qu'une structure JSON qui contient les stats dans ses différents champs. Un champ intéressant est le champ *owner* qui permet de confirmer que la chaîne de caractère *901d3889-bacc-43f8-acb8-844fa59cb9be* correspond à l'utilisateur qui possède l'espace de stockage. Les champs *quota*, *free*, *total* et *used* indiquent combien d'espace de stockage l'utilisateur a droit et combien il en utilise actuellement.
- **GOAWAY** du serveur et fermeture de la connexion TCP.

Le volume de données échangé est très petit (10kB) puisqu'aucun fichier n'est échangé, il suffit d'envoyer l'ordre de supprimer le fichier et le tour est joué.

F. Déplacement d'un fichier

La procédure est assez similaire à la suppression d'un fichier, mais on a tout de même une différence notable : la requête avec la méthode **DELETE** est remplacée par une requête dont la méthode est **MOVE**. Cette méthode n'est pas standard et vient de *WebDAV*. Elle permet de déplacer une ressource. Dans le header on a 2 informations importantes : *path*, qui va contenir le chemin vers le fichier à déplacer et *destination* qui contient, comme son nom l'indique, le nouvel emplacement du fichier. *destination* a ce format : *.../remote.php/dav/files/**id_client/nouvel_destination/nom_du_fichier***. Le serveur envoie ensuite une réponse avec le statut qui correspond : 201/Created en cas de succès. D'autres codes sont prévus si la tentive de déplacement est un échec [1]. Tout comme pour la suppression de fichier, le volume de données échangé est petit puisque aucun fichier n'est transferé sur le réseau, uniquement quelques paquets pour demander le déplacement du fichier. Aussi, le renommage d'un fichier comporte exactement les mêmes requêtes que le déplacement d'un fichier.

G. Type de connexion

Nous avons regardé si il y avait une différence au niveau du fonctionnement lorsqu'on est en 4G, nous n'avons pas constatés de différence notable. Les serveurs contactés sont les mêmes et le processus pour uploader un fichier est le même que présenté à la section VI-C.

TABLE III: 4G vs Ethernet vs Wifi

	Ethernet	**4G**	**Wifi**
Retransmission TCP en moyenne [#/sec]	2.73	0.8	1.14
Temps d'upload [sec]	61	260	68
Débit [MB/sec]	0.8	0.19	0.72

1) Analyse des retransmissions TCP Wifi vs 4G: Le tableau III nous montre que le nombre moyen de retransmissions par seconde est beaucoup plus élevé pour la connexion Ethernet que 4G. Ce résultat nous paraît un peu surprenant. Intuitivement, Ethernet semble un moyen de communication plus fiable. En ce qui concerne le débit, pour l'upload d'un même fichier, le temps d'upload est beaucoup plus long pour la 4G que par Ethernet. En ce qui concerne le wifi, il est légerement plus lent en terme de débit qu'Ethernet, mais a, comme la 4G, moins de retransmission TCP qu'Ethernet. Cependant ces résultats sont à nuancer car ils dépendent beaucoup des conditions du réseau (4G ou filaire) et de l'infrastructure dans laquelle les mesures sont réalisées. Le test pour Ethernet et Wifi ont été réalisé avec un relais CPL, qui pourrait peut-être influencé la qualité de la connexion et donc le nombre de retransmission et le débit.

VII. Conclusion

Cette analyse du fonctionnement d'un point de vue réseau de l'application Shadow Drive n'est pas complète : nous n'avons pas abordé le partage de fichier, le processus de connexion, etc. Cependant cela nous a permis de comprendre en profondeur les différents protocoles réseaux et leurs rôles propres à chacun, afin d'avoir une meilleur compréhension du fonctionnement d'internet et de son infrastructure.

References

[1] Archiveddocs. *WebDAV Reference*. Aug. 20, 2015. URL: https://learn.microsoft.com/en-us/previous-versions/office/developer/exchange-server-2003/aa486282(v=exchg.65) (visited on 04/03/2024).

[2] M. Belshe, R. Peon, and M. Thomson. *RFC7540*. IETF HTTP Working Group Specifications. May 2015. URL: https://httpwg.org/specs/rfc7540.html (visited on 04/02/2024).

[3] Carson. *HTTP/2 and How it Works*. Medium. Mar. 23, 2021. URL: https://cabulous.medium.com/http-2-and-how-it-works-9f645458e4b2 (visited on 04/02/2024).

[4] *Certificats encryptés*. URL: https://superuser.com/questions/1648334/in-wireshark-where-can-i-find-the-tls-servers-certificate (visited on 04/04/2024).

[5] *Certificats wireshark*. URL: https://richardatkin.com/post/2022/01/15/Identifying-and-retrieving-certificates-from-a-PCAP-file-using-Wireshark.html (visited on 04/04/2024).

[6] *En-têtes HTTP - HTTP — MDN*. Dec. 21, 2023. URL: https://developer.mozilla.org/fr/docs/Web/HTTP/Headers (visited on 04/02/2024).

[7] *Enregistrement DNS SOA : définition et utilisation*. okta. URL: https://www.okta.com/fr/identity-101/soa-record/ (visited on 04/04/2024).

[8] *HTTP response status codes - HTTP — MDN*. Nov. 3, 2023. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (visited on 04/02/2024).

[9] *Méthodes de requête HTTP - HTTP — MDN*. Aug. 3, 2023. URL: https://developer.mozilla.org/fr/docs/Web/HTTP/Methods (visited on 04/03/2024).

[10] *Network Address Translation Definition — How NAT Works — Computer Networks*. CompTIA. URL: https://www.comptia.org/content/guides/what-is-network-address-translation (visited on 04/02/2024).

[11] *notifications/docs/ocs-endpoint-v2.md at master · nextcloud/notifications*. GitHub. URL: https://github.com/nextcloud/notifications/blob/master/docs/ocs-endpoint-v2.md (visited on 04/02/2024).

[12] *OCS Share API — Nextcloud latest Developer Manual latest documentation*. URL: https://docs.nextcloud.com/server/latest/developer_manual/client_apis/OCS/ocs-share-api.html (visited on 04/03/2024).

[13] *Protection DNS*. URL: https://ebrandservices.fr/attaques-dns-comment-proteger-efficacement-ses-noms-de-domaine/ (visited on 04/04/2024).

[14] Dan Shanahan. *Graphing Packet Retransmission Rates with Wireshark*. The Visible Network. Feb. 4, 2015. URL: https://networkvisibility.wordpress.com/2015/02/04/graphing-packet-retransmission-rates-with-wireshark/ (visited on 04/03/2024).

[15] *TCP/IP Model*. GeeksforGeeks. Section: Computer Networks. Oct. 4, 2017. URL: https://www.geeksforgeeks.org/tcp-ip-model/ (visited on 04/03/2024).

[16] *The SOA record*. NsLookup.io. Mar. 3, 2022. URL: https://www.nslookup.io/learning/dns-record-types/soa/ (visited on 04/05/2024).

[17] *What are DNS records?* URL: https://www.cloudflare.com/learning/dns/dns-records/ (visited on 04/03/2024).

[18] *Wikiwand - SOA Resource Record*. Wikiwand. Sept. 2020. URL: https://www.wikiwand.com/fr/SOA_Resource_Record (visited on 04/05/2024).