Mise en quarantaine du navigateur

Fabrice Desclaux et Frédéric Vannière
fabrice.desclaux@cea.fr
frederic.vanniere@cea.fr

Commissariat à l'énergie atomique et aux énergies alternatives

Résumé. L'article suivant détaille les choix techniques ainsi que l'architecture d'un outil permettant de sortir le navigateur des postes clients d'un réseau d'entreprise et de l'exécuter dans un environnement plus contrôlé. Le client n'a alors plus qu'un déport visuel de ce navigateur. Nous détaillerons dans une première partie l'architecture permettant d'exécuter ces navigateurs. Les choix techniques ainsi que les performances obtenues seront discutés dans un second temps.

1 Introduction

1.1 Les menaces de la navigation sur internet

Les navigateurs font aujourd'hui énormément de travail et ont une taille de code relativement importante. Ils doivent manipuler une multitude d'objets différents, ce qui expose une grande surface d'attaque : du javascript, du HTML, des polices d'affichage, des formats vidéo, audio, pdf ...

Les conséquences sont là : les vulnérabilités sur les navigateurs sont nombreuses, même si Flash et Java ont disparu de cette surface d'attaque. Le problème est d'autant plus sensible que le navigateur s'exécute sur le poste utilisateur, avec ses droits, poste qui lui-même est directement relié au domaine Windows. En prenant le contrôle du navigateur, l'attaquant a donc un tapis rouge déroulé vers les ressources de l'utilisateur ainsi que vers l'Active Directory du domaine. Le navigateur du poste client n'est qu'une barrière de péage dans laquelle l'attaquant doit jeter négligemment quelques pièces pour accéder à l'autoroute du système d'information de l'entreprise.

Un autre cas de figure certes plus rare mais néanmoins existant reste le ver dans la pomme : comment détecter et quantifier les données exfiltrées par un utilisateur indélicat ?

— en surveillant ses mails personnels ?
— au jugé de la quantité de données exfiltrées et en croisant les doigts pour que les volumes deviennent détectables ?

— en observant le marc de café du fond de la tasse durant les insomnies chroniques ?

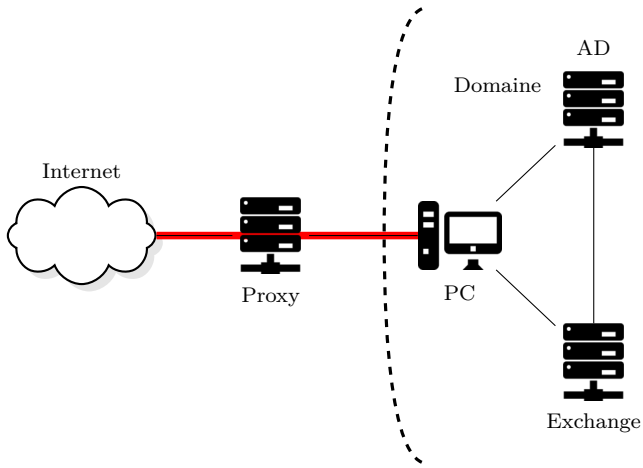

Fig. 1. Schéma réseau d'accès à internet : en rouge, nous sommes aveugles, impuissants

1.2 État de l'art

Pour se protéger des menaces détaillées ci-dessus, différentes protections sont souvent mises en place.

Proxy filtrant et interception TLS La première protection est le proxy HTTP filtrant avec des règles d'accès et des listes de filtres mises à jour régulièrement pour bloquer l'accès aux sites les plus douteux. Le proxy peut être authentifiant et assurer la traçabilité des accès web.

Malheureusement, la réalité est qu'aujourd'hui la durée de vie d'une *landing page*[1] dépasse rarement les 12h. Le temps que le signalement d'un site malveillant circule, l'attaquant a déjà fermé sa trousse, pris son cartable et installé sa tente sur un autre domaine.

Par ailleurs, l'augmentation du trafic HTTPS n'a pas arrangé la situation : autour de 90% du trafic web est chiffré. Une parade est de

[1] Page web installée par un attaquant, attendant le chaland tel une toile tissée par une araignée, attendant le moucheron imprudent

pratiquer pour certaines entreprises le *man-in-the-middle* HTTPS permettant d'analyser le trafic des navigateurs. Mais depuis déjà 2015, l'exploit kit *Angler* fait par exemple son propre Diffie-Hellman en javascript sous le HTTPS [10].

Aussi, l'interception TLS pose un certain nombre de problèmes et oblige notamment à diminuer la sécurité des sites visités en usurpant leur certificat TLS, de plus, elle expose les données personnelles des utilisateurs. Le choix d'accepter ou non le certificat serveur revient à l'équipement qui fait l'interception et non plus à l'utilisateur. Les navigateurs sont de plus de plus en plus vigilant vis à vis du TLS, et notamment avec les mécanique de *cert pinning*. Il faut alors garantir que l'équipement suive ces mécanismes.

Dans ces conditions, la palette des parades disponibles pour les entreprises tire sur le pastel.

Navigateur distant Certaines entreprises utilisent une solution de bureau à distance pour la navigation sur Internet en se basant sur des protocoles comme VNC, RDP ou encore SPICE.

Ce type d'infrastructure complexifie les attaques car une fois la main prise sur le navigateur web, l'attaquant doit remonter vers le poste utilisateur.

Malheureusement, ces protocoles sont complexes et ont leurs propres failles allant jusqu'à l'exécution de code sur le poste utilisateur. En voici quelques exemples :

1. CVE-2020-14355 SPICE exécution de code à distance entre client et serveur
2. CVE-2021-34535 et CVE-2022-21851 pour Remote Desktop Client : vulnérabilité entraînant une exécution de code arbitraire
3. CVE-* En novembre 2019, Kaspersky annonce que leur CERT a trouvé 37 vulnérabilités dans diverses implémentations du protocole VNC [8]

RDP possède de nombreux canaux et peut exposer un périphérique du poste utilisateur vers le serveur. Un *Terminal Server* permet par exemple à un utilisateur distant de brancher une clef USB sur son poste de télétravail, et de retrouver sa clef USB montée directement sur le terminal server.

Un point un peu plus subjectif, concernant certaines de ces solutions comme VNC par exemple, est que les performances ne sont pas forcément au rendez-vous ou sont trop consommatrices de bande passante :

l'utilisateur rechigne alors à perdre son confort et la solution n'est pas adoptée.

L'étude [6] montre que dans une configuration dite "Low Bandwidth" limitée à 10Mbit/s, les protocoles Spice et VNC offrent une qualité très faible.[2] Dans un environnement limité à 100Mbit/s, la qualité atteint les 50% de la vidéo originale. Le problème est que nous souhaitons ici pouvoir supporter plusieurs centaines d'utilisateurs en concurrence sur une liaison 1 Gbit/s. La bande passante pouvant être allouée pour un utilisateur se situerait par conséquence plus autour de 4Mbit/s, ce qui est un débit encore plus bas que le profil bas débit de l'article cité.

Le lecteur averti est sûrement en train de bondir de sa chaise, rétorquant que d'autres solutions existent déjà. Les solutions comme Seamless RDP, VMWare Horizon ou Citrix offrent ce genre de fonctionnalités. Ces solutions propriétaires ne permettent pas forcément un contrôle fin entre le client et l'hyperviseur : il est difficile de connaître avec exactitude les informations qui circulent à travers ce lien. De plus, les protocoles utilisés sont majoritairement propriétaires et peuvent être sujets à des vulnérabilités permettant de remonter vers la machine de l'utilisateur.

Ces solutions propriétaires ont également eu leur lot de vulnérabilités :

— Citrix, à travers un avis de sécurité du CERT-FR en février 2022 [3] ;
— ici une exécution de code privilégiée dans la machine virtuelle permet de planter l'hyperviseur VmWare Fusion au travers d'une vulnérabilité dans le parser de fonte *TueType* [7] ;

Cloudflare Zero Trust Network Access Cloudflare propose une plateforme *zero trust* pour les entreprises. Elle combine des règles de filtrage avancées avec une technologie de rendu HTML déporté dans le navigateur. La manipulation du HTML, du CSS ainsi que du code javascript d'un site distant est faite dans un navigateur web de Cloudflare. Les opérations graphiques vectorielles en résultant sont envoyées au navigateur du client. Ce dernier ne fait alors qu'exécuter ces opérations graphiques pour obtenir le rendu final de la page web.

La surface d'attaque est réduite ici à l'interface des opérations graphiques. Cette solution est performante et sécurisée mais elle n'est disponible que dans un mode SaaS (*Software as a Service*) exécuté sur l'infrastructure de Cloudflare qui a donc accès à toutes les données du client : cookies, mots de passe, historique,...

[2] La qualité vidéo est ici mesurée en utilisant la méthode du Slow–motion benchmarking référencée dans leur article

1.3 Nos objectifs et solution proposée

Les objectifs de notre solution sont multiples :
— limiter l'impact sur le réseau d'entreprise d'une compromission du navigateur ;
— assurer la traçabilité des accès Internet avec notamment la possibilité d'analyser finement ce qui entre et sort du réseau d'entreprise ;
— en cas de compromission, pouvoir analyser une attaque et retrouver le vecteur d'attaque.

Pour une protection efficace, la solution doit être utilisable par tous les utilisateurs du domaine et donc fournir une bonne expérience utilisateur : quasiment aussi naturelle qu'un navigateur classique, fluide et transparente.

L'idée est ici de sortir le navigateur de la zone de sécurité de l'utilisateur (et donc du domaine). Pour cela, on installe un hyperviseur dans une DMZ qui fera tourner des machines virtuelles. Chaque client a sa propre machine virtuelle. Cette machine virtuelle fait tourner le navigateur du client. La solution assurera un accès graphique confortable à ces machines depuis un poste client.

2 Projet Sanzu : déport vidéo pour des machines virtuelles

Comme vu en introduction, la solution d'isoler la navigation web dans une VM avec du déport d'affichage est bonne mais à condition de bien maîtriser la surface d'attaque et d'offrir de bonnes performances aux utilisateurs.

C'est là qu'arrive le projet Sanzu de déport d'affichage vidéo et des entrées/sorties utilisateur (clavier, souris, copier/coller).

Chaque navigateur s'exécute dans une machine virtuelle elle même s'exécutant sur un hyperviseur dans une DMZ.

2.1 Architecture

Les flux venant d'Internet arrivent sur ces navigateurs. Les utilisateurs lancent un client qui affichera le rendu déporté du navigateur et le tour est joué.

L'architecture est pensée pour minimiser la friction entre l'utilisateur et le système remplaçant son navigateur sur Internet. Elle se découpe en plusieurs parties :

Un service s'exécute sur l'hyperviseur : il reçoit les connexions des clients, protégées par TLS et authentifiées en Kerberos avec les *credentials*

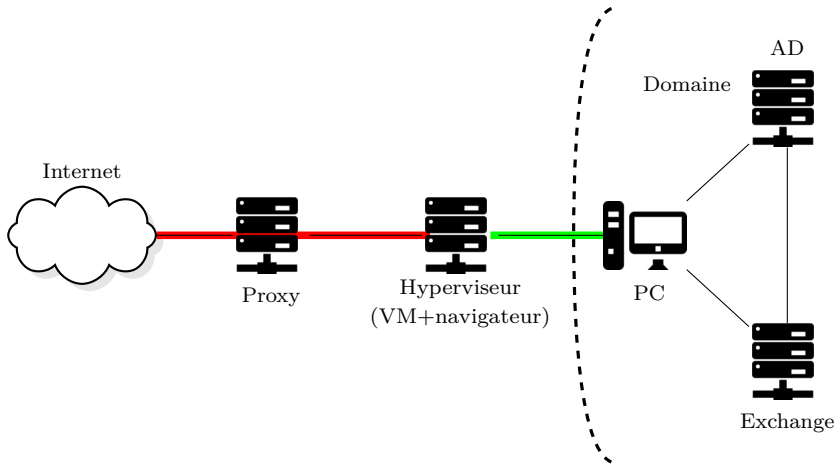

Fig. 2. Schéma réseau avec hyperviseur

de l'utilisateur. L'utilisateur n'a ici pas besoin de retaper un mot de passe. Pour cela, une *keytab* du domaine est générée et installée sur l'hyperviseur. Ainsi, l'hyperviseur peut vivre hors du domaine.

On peut donc souligner que de nouvelles surfaces d'attaque sont exposées par le client lourd :

— le protocole client/serveur ;
— le décodeur vidéo ;
— le décodeur audio.

Pour limiter ce problème, l'encodeur vidéo/son est sorti de la machine virtuelle.

Les machines virtuelles, une par client, embarquent un agent qui s'occupe de récupérer l'écran du navigateur et de simuler la frappe des touches et les mouvements de souris de l'utilisateur. Il ne compresse pas la vidéo, mais envoie les images brutes hors de la machine virtuelle.

Un *proxy vidéo* est également lancé et s'occupe de faire le lien entre le client et l'agent tout en encodant la vidéo sur l'hyperviseur. Ainsi, les cartes graphiques utilisées pour l'encodage vidéo sont liées à l'hôte et non à la machine virtuelle.

Le point intéressant est que si l'attaquant arrive à prendre le contrôle du navigateur ainsi que du système situé dans la machine virtuelle (y compris l'agent) la surface d'attaque à laquelle il a accès est réduite à la communication entre l'agent et le proxy qui utilisent Protobuf. Les données embarquées par Protobuf à ce stade sont limitées à une image de

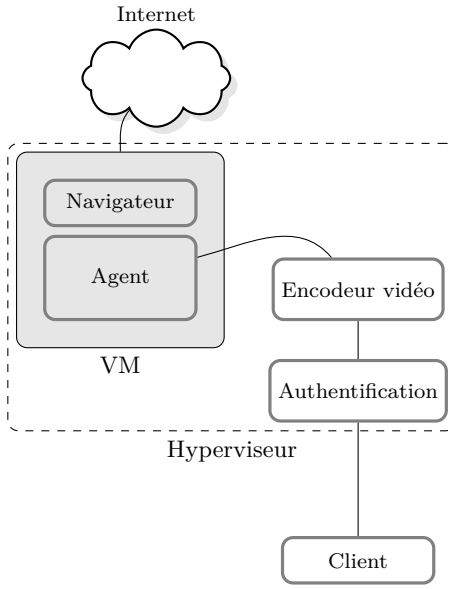

Fig. 3. Vue interne de l'hyperviseur

pixels et de sons *bruts* non parsés par le proxy. Le décodeur vidéo/son du client est ici sorti de la surface d'attaque.

Pour être réactif lors de la connexion des clients, des machines virtuelles sont préparées et lancées *sans* profil spécifique. Un service s'exécutant sur l'hyperviseur s'occupe de maintenir un nombre constant de machines prêtes à être utilisées. Si une machine virtuelle est consommée, il en démarre une nouvelle. Le but est de pouvoir encaisser un grand nombre de clients qui se connectent dans une fenêtre de temps réduite. Les étapes suivantes sont alors déroulées :

— l'utilisateur lance le client lourd sur son poste ;
— la connexion en TLS + authentification Kerberos est faite ;
— le broker récupère le *username*.

Le serveur exécute ensuite les étapes suivantes :

— Une machine virtuelle est choisie dans le pool de VMs non affectées. Ces machines sont déjà démarrées. Un utilisateur générique est utilisé.
— Chaque utilisateur possède une image disque sur l'hyperviseur contenant :
  — son profil du navigateur ;
  — son cache ;

— ...
— Cette image est alors montée dans la machine virtuelle sélectionnée et le profil du navigateur est prêt.
— le navigateur est lancé en utilisant l'utilisateur générique, en profitant du profil de l'utilisateur situé sur l'image disque.
— La machine virtuelle est prête.

On peut décider de conserver le proxy authentifiant les clients ou d'autres services situés dans la DMZ. On installe alors un serveur Kerberos sur l'hyperviseur. Ce dernier permettra de faire le lien entre les clients authentifiés provenant du domaine à protéger (par exemple le domaine Windows), et l'accès à des services dans la DMZ.

Les étapes suivantes sont effectuées :

— un domaine Kerberos à part entière est installé sur l'hyperviseur ;
— le domaine est vide à l'origine et ne contient aucun utilisateur ;
— l'utilisateur qui s'authentifie auprès du service est rajouté à la volée dans le domaine Kerberos ;
— une *keytab* est créée pour l'utilisateur sur le domaine personalisé ;
— les tickets associés aux services qui seront utilisés dans le futur sont créés et insérés dans la machine virtuelle ;
— la machine virtuelle est prête !

L'avantage de ce système est d'éviter d'avoir une synchronisation entre le vrai domaine et le domaine hébergé dans la DMZ. Si un utilisateur est ajouté sur le domaine à protéger et que son authentification est acceptée, il sera automatiquement rajouté et reconnu dans le domaine de la DMZ. Ce mécanisme a quand même un inconvénient : le service qui authentifie les utilisateurs provenant du domaine à protéger a besoin de droits élevés sur le domaine de la DMZ pour pouvoir rajouter des utilisateurs à la volée. Sa compromission permettrait de créer n'importe quel compte utilisateur sur ce domaine (mais pas sur le domaine à protéger, bien entendu).

2.2 Autres services : téléchargement, envoi, impression

Des questions sont pour le moment en suspens : comment télécharger un fichier ou imprimer une page web ? Effectivement, quand l'utilisateur enregistre un fichier depuis son navigateur, celui-ci est stocké dans la machine virtuelle et non sur son poste.

Pour pallier ce problème, nous installons également un service *WebDAV* (donc HTTP), écrit en Rust [1], permettant le transfert bi-directionel des fichiers. Ce service WebDAV authentifie également d'un côté les clients venant du vrai domaine en utilisant une keytab de ce domaine, et d'un

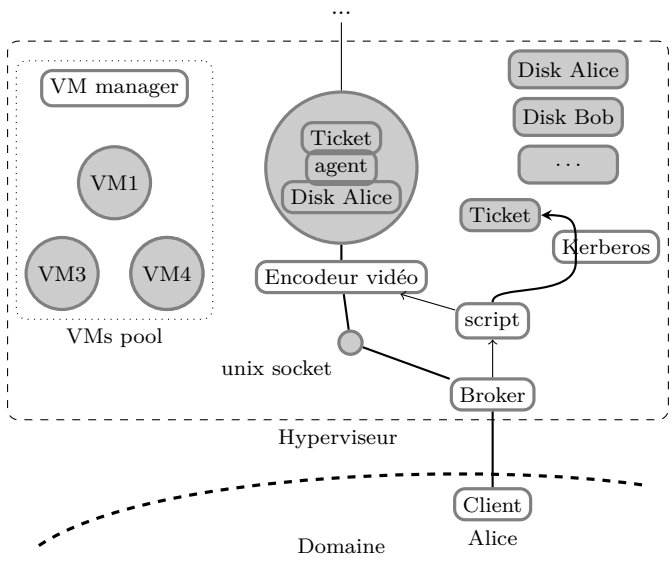

Fig. 4. Authentification d'un utilisateur du domaine à protéger sur le service de navigation déportée

autre côté les utilisateurs venant des machines virtuelles en utilisant les tickets préalablement insérés dans ces dernières.

Le client WebDAV est natif et activé sur les postes Windows par défaut. Le WebDAV est monté dans la machine virtuelle sur le dossier de téléchargement côté machine virtuelle et sur un lecteur réseau côté poste utilisateur. Chaque utilisateur est cloisonné dans son propre répertoire.

Pour envoyer un fichier sur une page web, l'utilisateur doit tout d'abord pousser le fichier sur l'espace de stockage WebDAV puis, dans le navigateur, séléctionner le fichier depuis le dossier "Téléchargements".

2.3 Résumé

Pour résumer, la nouvelle surface d'attaque côté hyperviseur est :
— QEMU/KVM : L'attaquant est dans une machine virtuelle ;
— Protobuf : connexion agent/*host*
  — vidéo brute : l'encodage est fait sur le *host* ;
  — son brut : l'encodage est fait sur le *host*.

Côté client :
— Le décodeur vidéo n'est pas exposé à l'attaquant ;
— Le décodeur son n'est pas exposé à l'attaquant.

— L'interface WebDAV / HTTP

Il est également à noter que le service qui reçoit les connexions des clients est écrit en Rust. L'agent, l'encodeur vidéo et le client sont aussi écrits en Rust.

2.4 Durcissement en profondeur

Maintenant que le principe du système a été exposé, intéressons-nous aux points pouvant être travaillés pour améliorer le niveau de sécurité de l'ensemble du système.

Génération des images de VM Il est indispensable d'avoir toujours un navigateur web et un système d'exploitation à jour. Pour cela, les images des VMs sont reconstruites de zéro chaque matin et toutes les VMs sont détruites afin de forcer l'utilisation de la nouvelle image.

Une fois l'image construite, celle-ci est testée automatiquement pour vérifier qu'elle est fonctionnelle et aussi pour s'assurer que le navigateur web et le noyau Linux sont bien à la dernière version disponible.

Système de fichiers overlay L'image disque de la VM est montée en lecture seule sur les VMs de surf, de cette façon un attaquant ne peut pas modifier une image et contaminer d'autres VMs. Aussi, un disque est créé spécifiquement pour la session de surf et monté par-dessus en overlay afin d'autoriser les écritures. Ce disque contient les journaux du système et toutes les modifications effectuées, il peut être utilisé pour auditer une VM de surf en cas de compromission ou comportement douteux.

Linux durci L'hyperviseur, comme chaque VM, dispose d'un pare-feu local bloquant toutes les connexions sauf les flux autorisés qui sont bien connus et maitrisés.

Les VMs peuvent utiliser SecureBoot, le principal intérêt est d'activer le *kernel_lockdown* du noyau Linux qui va bloquer tous les accès directs en mémoire et le chargement des modules noyau non signés. La signature du noyau se fait lors de la génération de la VMs avec une clé externe.

Pour aller plus loin, il est possible d'utiliser SELinux ainsi qu'un noyau Linux durci avec *Grsecurity* et *PAX* pour compliquer grandement l'exploitation d'une faille de sécurité noyau. Effectivement, un avantage de cette solution est que nous avons les mains libres dans la configuration du système d'exploitation qui supporte le navigateur.

Navigateur durci Au niveau du navigateur web, il est possible de désactiver certaines fonctionnalités au détriment des performances :

— le support Wasm
— le JiT de Javascript (responsable d'environ 30% des failles de sécurité)
— l'installation de modules non validés
— DNS over HTTPS afin de garder la traçabilité des requêtes DNS

Il est aussi envisageable de recompiler le navigateur pour qu'il enregistre toutes les URLs chargées.

Contrôle des entrées/sorties Chaque fichier téléchargé ou envoyé passe obligatoirement par le serveur WebDAV (servant de passe-plat). Il est possible d'ajouter sur ce dernier une passerelle d'analyse antivirale ainsi que de séquestrer les *hashes*, voire les fichiers qui sont échangés avec Internet.

Le flux entre le client et l'hyperviseur utilise un protocole maîtrisé (sérialisation et désérialisation par Protobuf) et peut donc être surveillé en cas de suspicion (exfiltration de données par copier/coller, ...). Le copier/coller peut également être configuré en sens unique : dans ce cas, l'utilisateur pourra copier depuis Internet et coller sur son bureau, mais copier depuis son bureau pour coller vers Internet lui sera impossible.

2.5 Confort d'utilisation et performances

Voici quelques points intéressants qui permettent à la solution de gagner en confort :

— la compression vidéo est faite par plusieurs cartes graphiques, interchangeables à chaud ;
— le son passe par le même flux et est compressé sur l'hyperviseur (sur CPU) ;
— l'authentification est faite sans mot de passe avec Kerberos/*credentials* utilisateur ;
— le copier/coller est supporté : il peut être modifié pour être unidirectionnel ;
— la résolution de l'écran est adaptée : lorsque la taille de la fenêtre du client est changée, le serveur adapte (sous xvfb par exemple) sa résolution à celle du client ;
— la latence réduite, en limitant les copies au niveau du code ;
— la qualité de l'image en utilisant par exemple le format YUV444 ;
— les images peuvent être exfiltrées de la VM par

  - TCP,
  - mémoire partagée (brutes, sans parsing),
  - Vsock ;
- l'encodeur vidéo est coupé quand l'image ne bouge plus. Les ressources sont alors libérées et peuvent être utilisées par un autre utilisateur ;
- *seamless* : seules les fenêtres liées au navigateur sont affichées sur le poste utilisateur. Le but est d'avoir un ressenti utilisateur d'intégration du navigateur dans son environnement.

L'exfiltration des images de la machine virtuelle est consommatrice de bande passante. Cela reste relativement faible vis-à-vis des vitesses des bus PCI express,[3] mais certaines mesures peuvent être prises. Un calcul rapide permet d'estimer la bande passante prise par l'exfiltration des images : $1920 \times 1080 \times 4 \times 30 \sim 248 Mo/s/client$. Ici on peut soit extraire les images en TCP, soit en utilisant une mémoire partagée entre l'hyperviseur et les machines virtuelles. L'exfiltration se fait alors avec un simple *memcpy*. Chaque machine virtuelle a un */dev/shm/video_x* associé sur l'hôte (40Mo) . Dans la machine virtuelle, cette mémoire partagée est vue comme une RAM PCI. L'échange de pixels *bruts* (non parsés) se fait alors entre l'hôte et la VM.

Pour tester la latence "porte à porte" on lance le serveur et le client sur la même machine et sur le même écran. On exécute le script suivant, avec une configuration 60fps :

```
while true; do echo $(($(date +%s%N)/1000000)); done
```

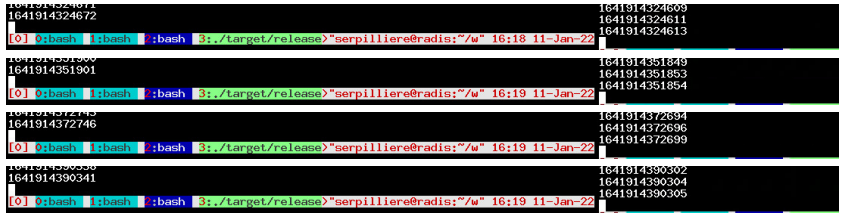

Fig. 5. Latence à l'affichage d'un déport graphique local. À gauche, l'écran original, à droite, l'affichage déporté

[3] Une carte Nvidia Tesla T4 possède une mémoire graphique GDDR6 avec une bande passante de 300Go/s. Elle est connectée en PCI Express 3.0x16 avec une bande passante de 32Go/s [9].

Ce test permet (à la louche) de calculer la latence de l'encodage / décodage. Les 4 tests montrent une différence à l'affichage de 59ms, 47ms, 47ms, 36ms, donc 48ms en moyenne. Il faudra rajouter à cela la latence réseau.

Pour rappel, certains tests en ligne montrent que Stadia [12] a une latence pouvant aller de 60ms à 120ms (de l'appui d'une touche au résultat affiché en passant par internet).

Une vue d'ensemble de l'architecture présentée dans cette partie est représentée sur le schéma 6.

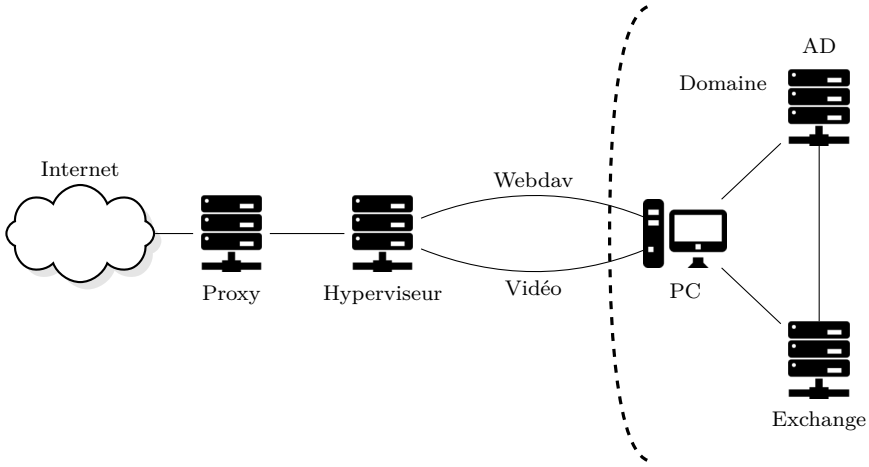

Fig. 6. Schéma réseau final

3 Solution graphique

Nous l'avons vu, il existe d'autres solutions d'affichage graphique déporté. Le nerf de la guerre pour que le système soit acceptable est que l'utilisateur ne se rende pas compte que le navigateur n'est pas exécuté sur son poste. Il faut donc qu'il ait :

— une faible latence (pour une bonne réactivité) ;
— un nombre d'images par seconde confortable (pour la fluidité) ;
— une qualité graphique élevée (pour une lecture confortable).

On peut alors se demander : quels utilisateurs possèdent aujourd'hui des solutions avec ce genre de caractéristiques ? Les *pro gamers*.

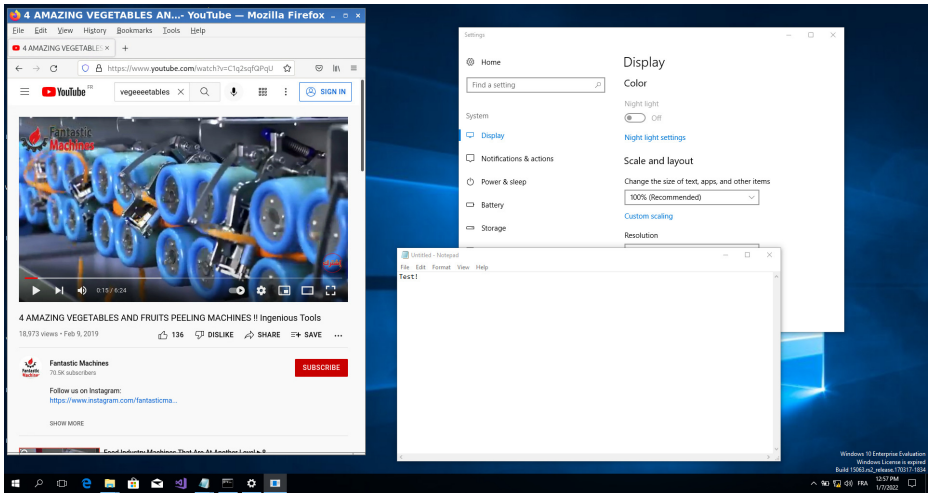

Fig. 7. Navigateur déporté affiché sur un poste Windows

3.1 Doom Guy

La solution adopte la même idée que les systèmes modernes utilisés dans le domaine du jeu vidéo sur PC/téléphone portable. L'idée est de déporter la complexité du rendu des jeux vidéo (Stadia, Shadow, ...) sur une machine spécialement dédiée à cette tâche et de *streamer* le résultat sur le PC/téléphone client.

Aujourd'hui, les cartes graphiques intègrent des compresseurs vidéo très performants : ils peuvent encoder plusieurs flux vidéos haute définition en parallèle [11]. Ici, on *streame* directement la sortie vidéo de la machine et on l'affiche sur la machine de confiance. La solution de base est donc simplement un client/serveur qui permet de *streamer* un bureau à distance. Dans ce mode, elle peut être utilisée comme n'importe quel autre système de déport de bureau graphique.

3.2 Performances graphiques

Cette partie décrit le principe utilisé pour la compression vidéo. Bien que ces informations peuvent paraître anodines, elles sont pourtant essentielles. Si l'outil a une qualité vidéo qui n'est pas au rendez-vous ou que sa latence est trop importante, il nuit à l'expérience utilisateur et ne sera pas adopté. Un effort a donc été apporté à cette facette du problème.

Nous nous plaçons dans le cas où l'outil a des restrictions de bande passante : si le serveur doit gérer un grand nombre d'utilisateurs en

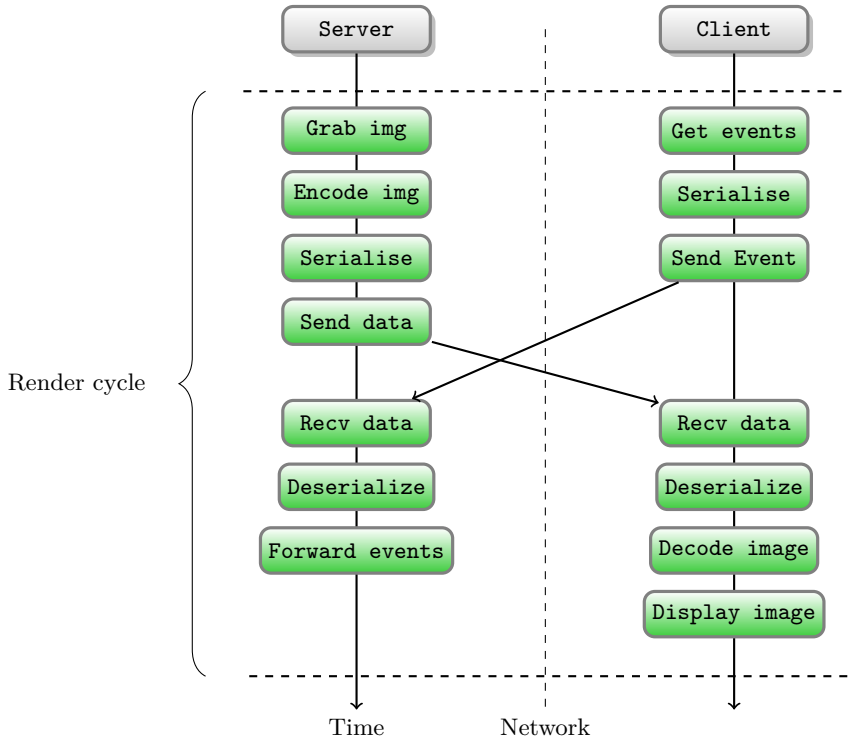

Fig. 8. Opérations effectuées par le client/serveur

parallèle, le réseau doit également supporter la transmission de tous ces flux vidéo. Aujourd'hui, une vidéo 1080p à 30fps consomme environ 300ko/s. Un hyperviseur connecté en gigabit pourra donc gérer autour de 400 clients. Pour obtenir ce résultat, nous utilisons la puissance des compresseurs vidéo modernes intégrés dans les cartes graphiques ou les CPU. La bibliothèque ffmpeg offre justement une API permettant d'envoyer un flux vidéo brut vers un encodeur matériel : NVENC [13] pour Nvidia, QSV (Quick Sync Video [5]) pour Intel. Notons que la compression vidéo peut également se faire sur CPU en utilisant par exemple le codec libx264. Elle consomme alors pour le flux précédent environ 3 cœurs à temps plein.

La seule différence est que ces derniers utilisent des formats de pixels différents. Le plus classique est le YUV420 (un plan pour la composante Y, un plan pour le U et un pour le V). L'encodeur QSV utilise par exemple le format NV12 (un plan pour le Y et un avec les composantes U et V entrelacées).

De façon identique ffmpeg permet de décoder une vidéo en utilisant un décodeur matériel. Cette facette est un peu moins critique, car le décodage d'un flux précédemment décrit consomme moins de 50% d'un CPU (ceci est un peu différent sur téléphone ou sur *raspberry pi*).

Nous arrivons au point des performances. Ces compresseurs vidéo sont relativement performants et prennent en règle générale moins de 10ms pour compresser une *frame* d'un flux vidéo. Pour rappel, si l'utilisateur final désire travailler en 1080p à 60fps, le budget pour une *frame* est en dessous de 17ms. Pour minimiser la latence lors de la compression de l'image, les encodeurs doivent également être configurés pour utiliser un profil "temps réel". L'astuce est ici de désactiver totalement la partie de compression vidéo temporelle qui se base sur l'image $N + 1$ pour encoder l'image N. En clair, l'encodeur n'a pas le droit de regarder dans le futur pour compresser la trame présente. Ainsi, dès qu'on donne une image brute à l'encodeur vidéo, ce dernier nous renvoie une image compressée sans attendre la suite des images du flux brut. Cela se fait au prix d'un ratio d'encodage un peu moins bon.

L'étape avant la compression est le changement d'espace colorimétrique. Traduire une *frame* du format RGB à YUV est également consommateur de ressources. Ici, nous avons fait le choix d'utiliser les capacités du CPU. Les extensions SSSE3 permettent de réaliser cette opération en moins de 3ms pour une *frame* (aux alentours de 10ms en CPU pur, sans SSSE3). On pourra dans le futur utiliser CUDA qui est totalement adapté à ce genre de tâches (et sera encore plus efficace).

La sérialisation/désérialisation se fait en utilisant Protobuf. Son temps d'exécution est négligeable vis-à-vis des autres opérations. Ce choix est motivé par la sécurité du code des parseurs générés par Protobuf.

Le choix du format de pixel peut avoir un impact sur la qualité visuelle. Le YUV420 par exemple profite du fait que l'œil humain est moins sensible aux différences de couleurs qu'aux différences de luminance. Ainsi pour un groupe de 4 pixels, chacun possède son information de luminance mais les 4 partagent la même information de couleur.

Cela n'est en général pas perçu lors du visionnage de films, mais peut devenir pathologique dans certains cas, notamment sur l'affichage d'un texte coloré sur un fond coloré [4], on peut le constater sur l'image 10.

Avec ces principes, nous obtenons une architecture client/serveur relativement performante et peu consommatrice de bande passante (autour de 300ko/s)

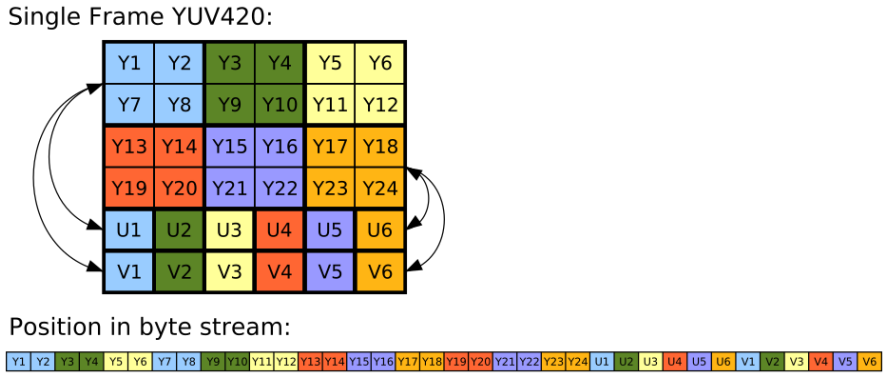

Fig. 9. Représentation des pixels encodés en YUV420 *Wikipedia*

Fig. 10. Artefact graphique lié à l'utilisation du YUV420

3.3 Webrtc

Des tests ont également été réalisés pour intégrer la partie client dans un navigateur web, de manière semblable à Guacamole [2]. Les premiers résultats sont encourageants. Cette modification se traduit par l'implémentation d'un proxy intermédiaire qui est *client* de l'encodeur vidéo. Il extrait les *frames* encodées en h264 des messages Protobuf et les encapsule dans le protocole Webrtc. Il n'y a pas ici de réencodage à opérer sur ces *frames*. La partie récupération de la souris/clavier du navigateur n'a pas encore été traitée à l'heure actuelle.

Ce mode est un peu moins bien intégré côté client, mais permettrait de se passer d'un client lourd à déployer sur les postes utilisateurs.

4 Retour d'expérience

Au sein de notre équipe, Sanzu a tout d'abord été utilisé pour remplacer notre navigation web et a ainsi offert un bien meilleur confort d'utilisation tout en étant plus sécurisée.

```
...
[client]
21.0ms  evts: 16.0µs  send: 63.7µs  recv: 9.8ms   dec msg: 417ns (dec 2.7ms   yuv 1.6ms   parse 11.9µs)
12.8ms  evts: 8.5µs   send: 66.8µs  recv: 4.4ms   dec msg: 1.1µs   (dec 4.6ms   yuv 1.7ms   parse 34.3µs)
17.3ms  evts: 10.0µs  send: 65.7µs  recv: 10.2ms  dec msg: 395ns (dec 2.6ms  yuv 1.8ms   parse 18.5µs)
14.2ms  evts: 28.2µs  send: 87.0µs  recv: 5.3ms   dec msg: 1.6µs   (dec 4.3ms  yuv 2.0ms   parse 27.7µs)
16.5ms  evts: 27.7µs  send: 81.6µs  recv: 9.8ms   dec msg: 386ns (dec 2.3ms   yuv 1.7ms   parse 12.9µs)
16.4ms  evts: 20.1µs  send: 65.1µs  recv: 10.1ms  dec msg: 445ns (dec 2.7ms   yuv 1.7ms   parse 12.9µs)
16.4ms  evts: 17.4µs  send: 50.8µs  recv: 10.3ms  dec msg: 372ns (dec 2.5ms   yuv 1.7ms   parse 17.9µs)
20.1ms  evts: 9.4µs   send: 62.0µs  recv: 11.7ms  dec msg: 387ns (dec 3.6ms   yuv 2.6ms   parse 13.0µs)
18.6ms  evts: 19.0µs  send: 55.4µs  recv: 9.6ms   dec msg: 405ns (dec 2.6ms   yuv 3.2ms   parse 13.2µs)
13.3ms  evts: 12.0µs  send: 69.3µs  recv: 3.1ms   dec msg: 905ns (dec 5.7ms   yuv 2.4ms   parse 44.1µs)

...
[server]
16.1ms grab: 3.2ms evt: 62.6µs encode: 12.8ms (yuv 4.0ms enc 8.8ms ) snd: 327ns send: 45.6µs  recv: 10.1µs
11.9ms grab: 1.4ms evt: 62.2µs encode: 10.3ms (yuv 3.1ms enc 7.2ms ) snd: 504ns send: 89.7µs  recv: 17.0µs
14.0ms grab: 1.6ms evt: 49.2µs encode: 12.3ms (yuv 2.9ms enc 9.5ms ) snd: 161ns send: 39.1µs  recv: 9.5µs
14.1ms grab: 1.5ms evt: 47.0µs encode: 12.5ms (yuv 3.2ms enc 9.4ms ) snd: 173ns send: 40.2µs  recv: 10.9µs
14.1ms grab: 1.5ms evt: 53.3µs encode: 12.4ms (yuv 3.1ms enc 9.4ms ) snd: 173ns send: 28.6µs  recv: 9.6µs
15.3ms grab: 1.5ms evt: 47.2µs encode: 13.7ms (yuv 3.1ms enc 10.6ms) snd: 224ns send: 39.0µs  recv: 10.1µs
16.6ms grab: 1.9ms evt: 1.6ms  encode: 13.1ms (yuv 4.3ms enc 8.8ms ) snd: 170ns send: 39.3µs  recv: 11.3µs
12.1ms grab: 1.4ms evt: 85.1µs encode: 10.6ms (yuv 4.0ms enc 6.6ms ) snd: 142ns send: 121.0µs recv: 29.9µs
15.5ms grab: 2.4ms evt: 64.6µs encode: 13.0ms (yuv 4.3ms enc 8.7ms ) snd: 114ns send: 33.2µs  recv: 11.7µs
11.7ms grab: 1.5ms evt: 60.5µs encode: 10.0ms (yuv 4.5ms enc 5.5ms ) snd: 139ns send: 56.3µs  recv: 12.4µs
```

Fig. 11. Détail des temps mis par les diverses opérations d'encodage / décodage

Aussi, avec le développement du télétravail, nous avons eu besoin d'une solution de déport de bureau performante passant par des VPNs et sessions SSH. Dans ce cadre, Sanzu est utilisé sans VM et expose directement le bureau. Là aussi, le gain en performance et qualité d'affichage par rapport aux solutions traditionnelles a été grandement appréciable.

5 Futur

5.1 Autre utilisation : administration graphique déportée

Comme nous l'avons vu, le système proposé permet de sortir le navigateur du domaine réseau à protéger. Une deuxième utilisation est simplement applicable à l'administration graphique déportée. Le choix peut être fait de conserver le même mécanisme de compression vidéo pour éviter d'exposer le décodeur vidéo de l'administrateur, ou de le supprimer et de se reposer sur la protection des processus du système d'exploitation du serveur administré.

On peut noter également que la partie client pourrait être séparée en deux dans le futur :

— la partie décodage vidéo/son pourrait être mise dans une sandbox seccomp : le décodage vidéo étant beaucoup moins gourmand que l'encodage, il peut se faire totalement sur CPU, donc sans accès à la carte graphique) ;

— la partie client et désérialisation Protobuf dans le processus courant.

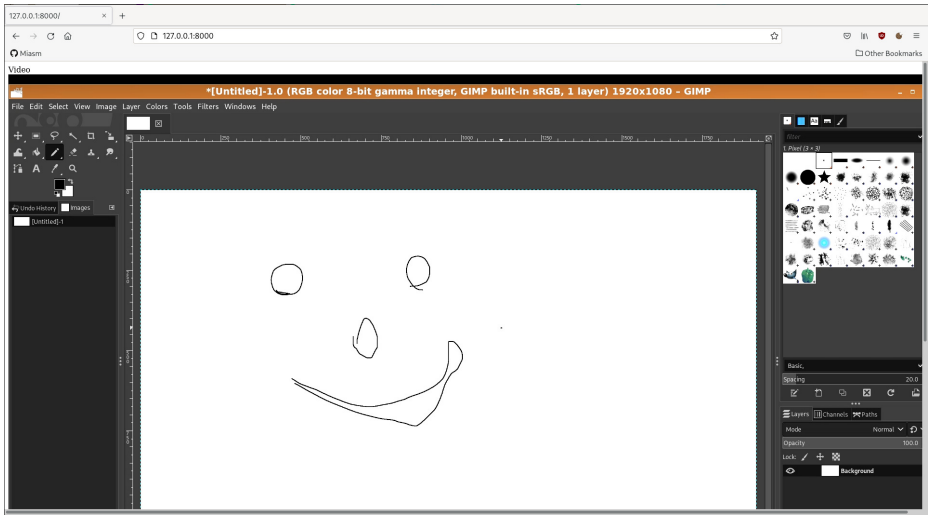

Fig. 12. Bureau distant intégré au navigateur

Cela permet d'alléger le système tout en limitant l'impact d'une compromission côté serveur.

5.2 Téléphones

Les *smartphones* pourraient aujourd'hui rejoindre la partie : ils intègrent des navigateurs et offrent de facto la même surface d'attaque. Un effet de bord intéressant est qu'on y trouve de plus en plus souvent les mêmes outils nécessaires à un PC classique : des extensions pour bloquer les publicités, les sites malveillants et même des antivirus.

De par leurs capacités de capture et de lecture vidéo, ils intègrent presque tous un encodeur/décodeur vidéo matériel efficace. Il serait alors intéressant de porter la partie cliente sur Android par exemple et d'associer l'ouverture des pages web à ce dernier. Tous les moyens de protection modernes seraient exécutés sur le serveur avec des ressources quasi-illimitées comparé au téléphone. La navigation web du téléphone deviendrait une simple lecture de vidéo *youtube*. Évidement, les ressources du téléphone lisant une vidéo seraient sûrement un peu plus sollicitées comparativement à un navigateur embarqué avec ses extensions et son antivirus.

Bien sûr il reste les applications. Si le téléphone utilise un VPN et/ou un proxy, il est raisonnable de penser que les accès à leur site pourraient se *whitelister* (comparativement à la navigation web) et ainsi limiter la surface d'attaque du téléphone.

Notons que la solution utilisant le WebRTC a également été testée avec succès sur téléphone portable. Avec cette dernière, nul besoin d'un client lourd installé sur téléphone : le navigateur natif du téléphone diffuse ici directement la vidéo de la solution de déport graphique.

Références

1. https://github.com/miquels/webdav-server-rs.
2. Apache Guacamole. https://guacamole.apache.org/.
3. CERT-FR : Multiples vulnérabilités dans Citrix Hypervisor. https://www.cert.ssi.gouv.fr/avis/CERTFR-2022-AVI-133/.
4. Comparison between H.264 YUV420 and YUV444. http://sschaber.de/2018/12/03/2-of-6-comparison-between-h-264-yuv420-and-yuv444/.
5. Intel Quick Sync Video. https://en.wikipedia.org/wiki/Intel_Quick_Sync_Video.
6. Remote desktop protocols, A comparison of Spice, NX and VNC. https://www.diva-portal.org/smash/get/diva2:530960/FULLTEXT01.pdf.
7. VMware Workstation and Horizon Client for Windows updates address a denial-of-service vulnerability. https://www.vmware.com/security/advisories/VMSA-2022-0002.html.
8. Vulnérabilité VNC. https://www.kaspersky.com/blog/vnc-vulnerabilities/31462/.
9. Nvidia T4 Tensor core GPU. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf, 2008.
10. Attacking Diffie-Hellman protocol implementation in the Angler Exploit Kit. https://securelist.com/attacking-diffie-hellman-protocol-implementation-in-the-angler-exploit-kit/72097/, 2015.
11. Roman Arzumanyan. Turing h.264 video encoding speed and quality. https://developer.nvidia.com/blog/turing-h264-video-encoding-speed-and-quality/.
12. Gamers Nexus. Google Stadia Lag Review. https://www.youtube.com/watch?v=m0gILReDQsY.
13. Nvidia. Video Encode and Decode GPU Support Matrix. https://developer.nvidia.com/video-encode-and-decode-gpu-support-matrix-new.