Sécurité des applications graphiques sous X et Wayland

Matthieu Herrb
Laboratoire d'Analyse et d'Archirecture des Systèmes, Université de Toulouse, CNRS
7, avenue du Colonel Roche, BP 123456, 31031 Toulouse Cedex 4

Résumé

Lorsqu'un utilisateur saisit une information confidentielle (identifiant, numéro de compte bancaire,...) dans un formulaire sur une application, il fait confiance à cette application pour ne pas détourner les informations saisies. Les attaques de type phishing reposent sur l'abus de cette confiance en présentant à l'utilisateur un formulaire qui usurpe l'identité d'un site auquel l'utilisateur est enclin à faire confiance.

L'authentification des sites web repose généralement sur la validation d'un certificat, matérialisée par une petite icône verte en forme de cadenas fermé. Mais un attaquant peut être capable de manipuler ce qui est présent à l'écran pour faire croire à tort à l'utilisateur qu'il est bien dans un cas de confiance. Cela va de la simple possibilité de faire passer une fenêtre en mode plein-écran pour recréer un environnement factice, à des attaques capables de contrôler le framebuffer à un niveau plus bas.

Cet article commence par rappeler les mécanismes de protection présents dans le système graphique X. Il montre ensuite quelles solutions peuvent être utilisés pour augmenter cette sécurité, en présentant les mécanismes de compartimentage implémentés dans Qubes OS. Il montre également comment le système Wayland, qui remplace petit à petit X, dispose de mécanismes permettant d'augmenter la sécurité des applications graphiques.

1 Problématique

L'évolution du paysage de la sécurité des systèmes montre depuis plusieurs années une évolution des menaces qui visent de plus en plus les applications du poste de travail et moins le code des serveurs, qui a fait l'objet de plus d'attention coté sécurité.

Le poste de travail et ses applications peuvent être attaqués de plusieurs manières :

- par des vulnérabilités dans le code de l'application (ou des bibliothèques utilisées), par exemple de nombreuses failles ont été publiées dans les décodeurs de formats de fichiers (images, vidéo,...) manipulés. L'exploitation d'une telle vulnérabilité consiste à faire ouvrir par la victime un document spécialement préparé pour provoquer une erreur dans l'application qui conduira à l'exécution du code malveillant injecté dans le document,
- par des vulnérabilités liées à la conception du logiciel qui vont permettre de leurrer la victime en lui faisant croire qu'elle se trouve dans une situation sûre (par exemple que son navigateur web se trouve sur une connexion *https* chiffrée et authentifiée vers un site connu alors que cela n'est pas le cas),
- en l'absence de vulnérabilité connue, de nombreuses attaques vont se baser sur l'ingénierie sociale pour faire exécuter à la victime le code malveillant à l'insu de son plein gré.

Pour protéger les systèmes de ces menaces, plusieurs techniques sont mises en œuvre. On trouve traditionnellement des méthodes préventives telles que :

- l'élimination des failles dans le code (au fur et à mesure de leur découverte, soit fortuite, soit via des programmes d'audit ou de « bug bounty » pour récompenser des chercheurs externes),
- l'utilisation de méthodes formelles permettant d'établir la preuve a priori de l'absence de faille pour l'implémentation des parties sensibles du code, qui manipulent des données potentiellement malicieuses.

Plus récemment sont également apparues des méthodes « curatives » qui prennent en compte l'existence de failles dans le code des applications. Leur but est de limiter la portée d'une attaque exploitant cette faille, soit en la détectant au plus tôt, soit en limitant ses effets potentiels. Parmi ces techniques on trouve entre autres :

- les mécanismes de protection contre les débordements de pile ou de buffer par des « canaris » (valeur particulière qui va être écrasée en cas de débordement de la zone protégée, ce qui permet de le détecter,
- les mécanismes d'introduction d'aléa dans l'espace mémoire des processus qui vont rendre plus difficile l'exploitation des failles en ne fournissant pas les services de base des bibliothèques système à des adresses fixes,
- les mécanismes de réduction des privilèges, qui vont isoler le code qui manipule des données potentiellement malveillantes dans un environnement ne disposant pas des mêmes privilèges que le reste de l'application (par exemple, n'ayant plus accès aux systèmes de fichiers de la machine).

Dans le monde Unix-like (y compris GNU/Linux) cette dernière approche n'est pas très développée et seules les navigateurs web principaux (Firefox [1], Chrome [2]) implémentent une telle protection par bac à sable.

Mais ces différentes approches ne sont vraiment efficaces que contre le premier type de vulnérabilité évoqué plus haut. Il existe d'autres vulnérabilités, qui ne sont pas liée à des erreurs de codage, mais d'avantage à la nature même des fonctionnalités qu'on attend d'un environnement de bureau. Une des caractéristiques d'un environnement de bureau informatique est d'offrir des fonctions de communication entre les applications pour échanger ou partager des données [3]. Ces fonctions sont une aubaine pour une application malveillante qui souhaite agir sur les autres applications ou leurs données, en particulier lorsqu'on mélange des applications manipulant des données de niveaux de sensibilité différents.

Certains systèmes (Apple iOS, Android) ont généralisé la dernière approche sous la forme de « bacs à sable » dédiés à l'exécution des applications de manière isolée des autres et avec un ensemble d'autorisations d'accès aux autres ressources du système fixées par le programmeur de l'application.

Cette notion de bac à sable est présente aussi dans le système Microsoft Windows [4] depuis la version 8, mais peu d'applications existantes en tirent parti.

1.1 Accès aux périphériques d'entrée (clavier, souris,...)

La plupart des environnements de bureau disposent de la possibilité de créer des raccourcis claviers qui fonctionnent partout sur le bureau (par exemple les touches de fonction dédiée à la gestion du volume sonore ou au contrôle de la luminosité de l'écran, mais aussi les touches qui permettent de changer de bureau virtuel ou de basculer vers une autre application).

Cette fonctionnalité implique la possibilité pour un ou plusieurs processus qui vont réagir à ces touches de recevoir l'ensemble des événements d'entrée. Cela permet donc aussi de recevoir toutes les données sensibles saisies au clavier, indépendamment de l'application censée les recevoir normalement.

1.2 Copie d'écran, enregistrement vidéo de session

Parmi les outils de base d'un environnement de bureau, on trouve en général un outil qui permet de créer des copies d'écran, voire de plus en plus souvent de créer des séquences vidéo à partir de tout ou partie du contenu de l'écran.

Entre les mains d'une application malicieuse, cette possibilité permet l'espionnage des autres applications et la capture des données potentiellement sensibles qu'elles affichent.

1.3 Mode plein écran

Pouvoir passer en mode « plein écran » offre la possibilité à une application de contrôler complètement l'affichage et donc de tromper l'utilisateur qui se trouve devant le poste de travail. Un cas simple consiste à réaliser un faux logiciel de blocage du poste de travail, (qui reproduit autant que possible l'apparence du vrai) dans le but de récupérer le mot de passe de l'utilisateur.

Couplé à la fonction de lecture du contenu de l'écran citée plus haut, il est même possible de créer une application qui va, en mode plein écran, recréer le bureau complet de la victime, en modifiant simplement quelques éléments critiques (par exemple l'icône indiquant la validité du certificat d'un site web sur le navigateur).

C'est pourquoi les navigateurs web modernes ont implémenté une alerte qui prévient l'utilisateur de la volonté d'une page web ou de l'utilisateur de passer en mode plein écran et lui demande de confirmer ce choix. Néanmoins c'est une action volontariste des développeurs de ces applications, et cela n'empêche pas d'autres applications potentiellement malicieuses d'utiliser cette technique.

1.4 Alertes et notifications

La possibilité d'afficher des notifications ou des alertes et de réagir à celles-ci peut être exploitée par une application malveillante pour pousser l'utilisateur à réaliser une action qui peut avoir des conséquences. Par exemple certains cadriciels de sécurité utilisent des alertes pour demander de confirmer les opérations nécessitant des privilèges via une fenêtre qui demande à l'utilisateur de saisir son mot de passe (ou l'identifiant et le mot de passe d'un compte autorisé à réaliser l'action). Si l'alerte ne provient pas du cadriciel en question mais d'une application malicieuse, le mot de passe sera volé.

1.5 Copier/coller et glisser/déposer

Ces deux mécanismes de communication inter-application vont également à une application malicieuse capable de lire le tampon d'échange d'accéder à des données qu'elle ne devrait pas forcément voir. Ainsi si un logiciel malveillant est présent pendant le transfert d'une donnée sensible (par ex. la copie d'un mot de passe depuis un gestionnaire de mot de passe),

2 Le cas de X

Dans le monde du système multi-fenêtre X, c'est l'*Inter-Client Communication Convention Manual (ICCCM)* et les *Enhanced Window Manager Hints (EWMH)* qui définissent les mécanismes de communication utilisés par les environnements de bureau (Gnome, XFCE, KDE, etc.).

Par défaut les mécanismes sont très libéraux et autorisent à peu près toutes les opérations listées ci-dessus à n'importe quel processus de l'utilisateur courant.

La granularité du contrôle d'accès est très grossière par défaut [5, 6] : toutes les applications autorisées à se connecter au serveur X ont accès à toutes les informations globales : espionnage du clavier, lecture du contenu de n'importe quelle fenêtre ou de l'écran entier, passage en mode plein-écran, création de nouvelles fenêtres, gestion des buffers de copier/coller.

Le mécanisme le plus répandu de contrôle d'accès au serveur X est le mécanisme des gâteaux magiques (magic cookies) : un secret est déposé par le serveur X lors de l'initialisation dans le répertoire de login de l'utilisateur autorisé à utiliser ce serveur. Toute application qui dispose du droit de lecture de ce fichier (ou bien à qui quelqu'un communique le secret par un canal quelconque) peut donc présenter le secret au serveur X et être autorisée.

X propose même, historiquement, une méthode d'accès par adresse IP, qui peut être configurée pour autoriser n'importe quelle adresse (`xhost +`) à se connecter.

Enfin l'implémentation de l'accélération graphique via le mécanisme de *Direct Rendering* (DRI) a longtemps permis à une application d'accéder directement via l'interface DRI à des zone de mémoire graphique d'autres applications autorisées à se connecter au même serveur [7]. La version 3 de l'infrastructure DRI corrige ce problème, mais est encore en cours de déploiement.

Cette situation rend les applications X extrêmement vulnérables aux problèmes évoqués plus haut. Que ce soit au niveau de l'entrée (une application X peut intercepter tous les événements clavier ou souris destinés aux autres applications pour réaliser un keylogger), de l'affichage (une application peut lire le contenu des fenêtres de n'importe quelle autre application), ou des mécanismes de copier/coller (n'importe quelle application peut lire le contenu de la sélection courante). X permet également à n'importe quelle application de créer des fenêtres sans décoration qui peuvent recouvrir toute ou partie d'une autre application pour modifier l'apparence de ce qui est affiché et tromper l'utilisateur.

L'extension XSecurity permet d'implémenter des politique de contrôle d'accès plus stricte. L'environnement de sécurité SELinux utilise cette extension pour augmenter un peu le niveau de sécurité des applications. Mais cette extension n'a pas été réellement utilisée pour développer des politiques de sécurité adaptées pour des applications réelles.

3 Qubes OS

Qubes OS est un système d'exploitation dont le but est de fournir une sécurité supérieure à la moyenne à ses utilisateurs [8, 9]. Il est basé sur un principe de cloisonnement des applications de niveaux de sécurité différents via un hyperviseur (Xen) qui exécute plusieurs machines virtuelles correspondant aux différents niveaux de sécurité [10].

Les aspects d'interface utilisateur et d'expérience utilisateur ont été étudiés et une solution originale est proposée pour traiter les différents problèmes vus plus haut sans trop restreindre les fonctionnalités [11].

En particulier Qubes OS introduit un pilote virtuel pour le serveur X, *Qubes GUI* qui permet à chaque machine virtuelle d'exécuter son propre serveur X, et dans le Dom0 de Xen, le serveur X principal (qui accède au matériel physique – clavier/souris et écran) affiche l'ensemble des fenêtres avec un gestionnaire modifié qui décore chaque fenêtre d'une couleur qui identifie sa machine virtuelle d'origine (figure 1).

Qubes GUI gère les événements d'entrée en n'envoyant à une VM que les événements générés lorsqu'une de ses applications a le focus. Il n'est ainsi plus possible d'espionner les entrées d'une application qui n'est pas dans la même VM.

Le copier/coller est également géré explicitement, de manière à contrôler quelle VM va pouvoir avoir accès au contenu du presse-papier. Ainsi pendant que le presse-papier contient un mot de passe critique, seule la VM désignée comme destination pourra lire son contenu, et celui-ci sera effacé immédiatement après.

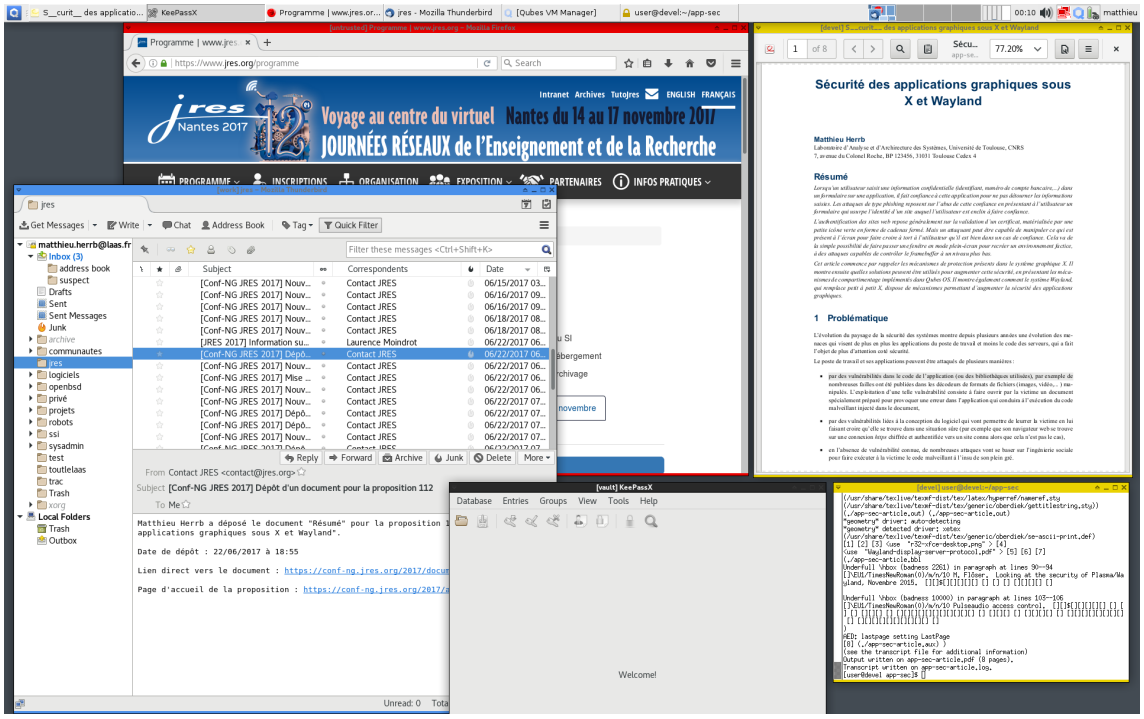

Figure 1 - Exemple de bureau de Qubes OS

Enfin Qubes GUI ne permet pas de créer des fenêtres sans décoration. Même un lecteur multimédia en mode plein-écran sera toujours encadré par un filet indiquant la VM d'origine de l'affichage.

La version actuelle (3.x) de Qubes OS utilise donc toujours X pour réaliser l'affichage, mais grâce au travail sur Qubes GUI, la plupart des problèmes évoqués précédemment ont été résolus pour garantir l'isolation des applications de niveaux de sécurité différents.

4 Wayland

Wayland[1] est un nouveau système graphique pour les systèmes Unix/Linux, basé sur le principe de composition du contenu des fenêtres applicative par un *compositeur*. Il a été créé par des développeurs de X.Org pour corriger un certain nombre des défauts de X. L'augmentation de la sécurité des applications n'était pas la motivation principale du développement de Wayland. La spécification du protocole [12] ne traite quasiment pas de ces aspects.

Néanmoins, en pratique, l'implémentation de référence fournit une sécurité par défaut qui est bien meilleure. Une application Wayland ne peut en effet pas communiquer avec les autres applications (figure 2). En particulier elle ne peut ni accéder aux événements des périphériques d'entrée destinés aux autres applications, ni accéder au contenu des fenêtres des autres applications. Le protocole Wayland ne permet pas non plus aux applications de modifier la taille ou la position de leurs fenêtres [13].

Ces restrictions ont cependant un impact sur des fonctionnalités que les applications peuvent fournir. Ainsi il n'est pas possible de faire une copie d'écran ou de passer une application en mode plein écran sans passer par le compositeur. Plusieurs solutions ont été proposées : pour les bureaux Gnome et KDE, des compositeurs spécifiques ont été développés (dans Mutter et Plasma respectivement) qui intègrent des plugins capables

[1] https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)

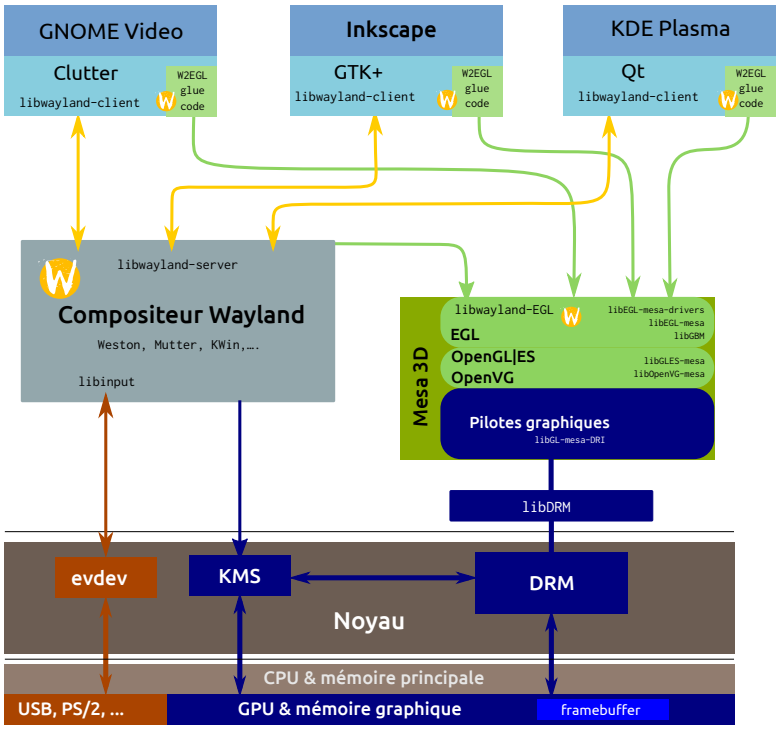

Figure 2 - L'architecture du système Wayland

d'exécuter ces fonctions privilégiées ou de les déléguer aux applications qui en ont besoin. Cette approche a néanmoins un impact sur la sécurité des applications [14].

Une nouvelle interface de gestion des droits d'accès et des politiques associées à été proposée par M. Perez [15] : *Wayland Security Modules* ou `libwsm`. Une version de cette bibliothèque a été intégrée à Tizen, le système d'exploitation développé par Samsung et Intel, basé sur le noyau Linux et Wayland et utilisé notamment pour les interfaces utilisateurs de téléviseurs et de systèmes de navigation pour les véhicules. Cependant cette bibliothèque n'a pas encore été adoptée par les environnements Gnome ou KDE.

5 Le son

Un type d'entrée/sortie assez courant a été négligé dans les paragraphes précédents : le son. Or les applications qui enregistrent ou jouent des sons sont également concernées par les notions de confidentialité : il n'est pas souhaitable que n'importe quelle application puisse enregistrer le son parvenant au micro du poste de travail, ni qu'une application puisse « écouter » le son joué par d'autres.

Dans ce domaine, les concepteurs des navigateurs Web ont aussi été les premiers à prendre conscience du problème et à intégrer dans l'interface utilisateur une demande explicite d'autorisation pour la prise de contrôle du micro (ou de la caméra) et un affichage explicite lorsque l'application enregistre le son. Mais une fois de plus cette solution n'est pas satisfaisante, puisqu'elle ne concerne qu'une petite partie des applications exécutées par l'utilisateur.

Le projet PulseAudio[2] a tiré les leçons de ce qui est fait dans Wayland et implémente désormais des mécanismes similaires de séparation entre les applications, empêchant l'accès aux flux audio entre applications non explicitement autorisées [16].

[2] https://www.freedesktop.org/wiki/Software/PulseAudio/

6 Directions futures

En parallèle au développement de Qubes OS ou de Wayland, dans le monde des serveurs la technologie des conteneurs a pris un essor remarquable. Il est apparu assez rapidement que cette technologie, qui permet d'isoler des services hébergés sur un serveur, peut également s'appliquer à l'isolation des applications sur un poste de travail. Les différentes technologies de conteneurs résolvent à la fois un problème de sécurité et un problème de distribution des paquets binaires des applications, en particulier lorsque plusieurs applications ont des dépendances similaires, mais incompatibles en elles.

L'idée a rapidement fait son chemin parmi les développeurs des environnements graphiques. Plusieurs projets utilisant des conteneurs pour l'exécution d'applications sont apparus [17, 18] : snap[3] ou flatpak[4]. En particulier, flatpak utilise tous les mécanismes d'isolation entre processus du noyau Linux (*namespaces*, *seccomp*, *cgroups*,...) pour exécuter chaque application dans un environnement isolé, sans pour cela avoir besoin des privilèges du super-utilisateur pour son lancement [19]. Un mécanisme de *portals* permet ensuite la communication entre les applications (pour l'accès au réseau ou au système de fichiers) tout en contrôlant celle-ci via une politique de sécurité.

Subgraph OS[5] est pour sa part un système complet, comparable dans son approche à Qubes OS, à ceci près qu'il utilise des conteneurs plutôt que des machines virtuelles pour séparer les applications de niveaux de sécurité différents. Son modèle de sécurité a été mis en cause [20], parce qu'il n'isole que les applications identifiées comme critiques, en laissant de nombreuses applications partager l'environnement par défaut qui continue à avoir un accès très large aux données. Cette remarque peut également s'appliquer aux systèmes assemblés à l'aide d'une combinaison de paquets « classiques » (`deb` ou `rpm`) et de paquets flatpak ou snap.

Si ces technologies semblent prometteuses et bien que des implémentations existent déjà, elles ne sont encore que très peu connues et utilisées.

7 Conclusion

La protection des données manipulées par les applications sur un poste de travail reste un objectif difficile à réaliser. Lorsque des données de niveaux de sécurité différents sont manipulées dans une même session de travail l'isolation a longtemps semblé impossible et la seule solution était d'avoir recours à plusieurs postes isolés physiquement.

Les techniques de virtualisation et de conteneurs apportent cependant une solution à ce problème et plusieurs implémentations (Qubes OS, Wayland, flatpak) proposent des prototypes utilisables.

Il reste cependant une difficulté qui n'a pas encore été abordée de manière réellement satisfaisante : toutes ces approches sont basées sur des politiques de sécurité qui deviennent assez rapidement complexes à analyser. Il est donc difficile de décider qu'il n'existe pas de faille dans la politique utilisée, ou qu'une modification de cette politique, demandée par l'utilisateur pour lui permettre de réaliser une tâche de manière simple, n'introduit pas de vulnérabilité critique.

Par contre si les environnements de bureau graphiques sur les systèmes GNU/Linux ou similaires arrivent à proposer une (ou plusieurs) solutions satisfaisantes en matière de sécurisation des applications de bureau, la fameuse « année de Linux sur le bureau » pourrait finir par arriver.

[3] https://snapcraft.io/
[4] http://flatpak.org/
[5] https://subgraph.com/

Bibliographie

[1] G. Destuynder. La sandbox firefox sous Linux. *MISC*, (78), Mars-Avril 2015.

[2] N. Ruff. Sécurité du navigateur chrome. *MISC*, (78), Mars-Avril 2015.

[3] S. Dodier-Lazaro et M. Peres. Security in Wayland-based desktop environments : Privileged clients, authorization, authentication and sandboxing ! Dans *XDC2014*, Bordeaux, Septembre 2014. https://www.x.org/wiki/Events/XDC2014/XDC2014DodierPeresSecurity/.

[4] S. Renaud et K. Szkudłapski. Protections des systèmes Windows. *MISC Hors-série*, (9), 2014.

[5] M. Herrb. Enhancing XFree86 security. Dans *RMLL 2003*, Metz, 2003. http://homepages.laas.fr/matthieu/talks/ftp/xf86-sec.pdf.

[6] M. Kerrisk. XDC2012 : Graphics stack security. *Linux Weekly News*, Septembre 2012. https://lwn.net/Articles/517375/.

[7] David Herrmann. DRM Security. Dans *X Developper's Conference 2013*, Portland, Oregon, Juillet 2013. https://www.x.org/wiki/Events/XDC2013/XDC2013DavidHerrmannDRMSecurity/.

[8] J. Rutkowska. Partitioning my digital life into security domains, Mars 2011. https://blog.invisiblethings.org/2011/03/13/partitioning-my-digital-life-into.html.

[9] B. Sonntag. Qubes - le nouvel OS sécurisé pour hacktivistes. Dans *PSES 2016*, Paris, Juillet 2016. https://frama.link/vhzs0_7a.

[10] M. Lee. With Virtual Machines, Getting Hacked Doesn't Have to Be That Bad. *The Intercept*, Septembre 2015. https://theintercept.com/2015/09/16/getting-hacked-doesnt-bad/.

[11] M. Marczykowski-Górecki. Improving client systems security with Qubes OS. Dans *RMLL2016*, Paris, Juillet 2016. https://sec2016.rmll.info/files/20160704-04-MarczykowskiQubesOS.pdf.

[12] Kristian Høgsberg. *The Wayland Protocol*, 2012. https://wayland.freedesktop.org/docs/html/index.html.

[13] J. Edge. The status of Wayland security. *Linux Weekly News*, Mars 2014. https://lwn.net/Articles/589147/.

[14] M. Flöser. Looking at the security of Plasma/Wayland, Novembre 2015. https://blog.martin-graesslin.com/blog/2015/11/looking-at-the-security-of-plasmawayland/.

[15] M. Perez. Wayland Compositors - Why and How to Handle Privileged Clients !, Février 2014. http://mupuf.org/blog/2014/02/19/wayland-compositors-why-and-how-to-handle/.

[16] Pulseaudio access control. https://www.freedesktop.org/wiki/Software/PulseAudio/Documentation/Developer/AccessControl/.

[17] J. Corbet. Package managers all the way down. *Linux Weekly News*, Janvier 2017. https://lwn.net/Articles/712318/.

[18] J. Edge. Flatpaks for Fedora 27. *Linux Weekly News*, 07 2017. https://lwn.net/Articles/728699/.

[19] A. Larsson. The flatpack security model, Janvier 2017. https://blogs.gnome.org/alexl/2017/01/18/the-flatpak-security-model-part-1-the-basics/.

[20] M. Lee. Breaking the security model of subgraph os. *Micah Lee's Blog*, Avril 2017. https://micahflee.com/2017/04/breaking-the-security-model-of-subgraph-os/.