Télécom ParisTech

Télécom SudParis

Mémoire de stage

Sécurité d'un contrôleur SDN : ONOS

Julien Schoumacher

Diplôme préparé : Ingénieur

Stage effectué du 20 juillet 2016 au 20 janvier 2017 à Télécom SudParis sous la direction de Grégory Blanc

Remerciements

Avant d'entamer la lecture de ce rapport, je tiens avant tout à remercier toute l'équipe du département RST (Réseaux et Services des Télécommunications) de Télécom SudParis qui m'a si bien accueilli durant ce stage. Je remercie également mon encadrant côté Télécom ParisTech Rida Khatoun, m'ayant mis en relation avec le maître de conférences Gregory Blanc qui m'a encadré avec bienveillance pendant toute la durée du stage. Enfin, toutes les autres personnes que j'ai pu cottoyer plus ou moins longtemps à l'occasion d'évènements ponctuels comme la conférence RAID qui s'est tenue en septembre.

Table des matières

1 Réseau SDN (Software Defined Network)

1.1 Motivation

Bien qu'il soit préférable d'éviter les approches rasoirs lorsqu'on souhaite introduire un sujet quelconque, on ne peut pas dans le cas de cette étude portant en partie sur les réseaux SDN, oublier de mentionner quelques éléments difficilement contestables sur les réseaux actuels [1] :

- ※ La demande ne cesse de croître : on observe un accroissement considérable des enjeux liés au traitement de masse importante de données, de l'utilisation de services cloud, du trafic mobile et peut être bientôt de l'utilisation d'objets connectés. Or tous ces éléments présentent le point commun de communiquer avec de nombreuses entités situées sur des réseaux potentiellement éloignés. Cela mobilise donc un trafic réseau intense.
- ※ Les technologies actuelles pour soutenir cette demande énorme sont capables de fournir un débit titanesque : que l'on considère des technologies sans fil ou non, au coeur des réseaux tant au niveau des terminaux des utilisateurs, on atteint aujourd'hui des débits théorique de l'ordre du Gigabit par seconde pour l'utilisateur. Tout cela sans que l'on ait vraiment conscience des conditions que cela requiert.
- ※ Les méthodes d'accès sont aujourd'hui bien différentes. Précédemment le modèle client/serveur était largement employé, avec dans le cas d'une entreprise, un réseau interne constitué de plusieurs LAN séparés, et connecté à internet de manière quasiment unique. Cela entrainant une configuration possiblement statique et donc aisée, le trafic se déroulant principalement sur un mode requête/réponse. Or la tendance, notamment à cause des deux premiers points, est à l'émergence de nouveaux modes d'accès plus horizontaux. Ce type de communication tient entre autres de la distribution plus éparse des données à travers le réseau : grossissement de la taille des bases de données, duplication de celles-ci (mise en cache sur différents serveur à travers le monde pour permettre un accès plus rapide), augmentation du trafic volumineux (vidéo, voix) et de nouveaux trafics (IoT (Bring Your Own Device), ...) même au sein de l'entreprise. Enfin, l'utilisation de plus en plus répandue de services cloud, avec ses implications en terme de virtualisation (que ce soit des applications, ou bien des bases de données), susceptible de changer en permanence la localisation des serveurs pour garantir une certaine flexibilité.

Or, le réseau principal global tel que nous le connaissons (la partie reposant sur TCP/IP en tout cas) a été conçu d'abord dans un but de résilience : chaque paquet doit être reçu, et peu

1. Aussi résumé dans `https://www.opennetworking.org/sdn-resources/sdn-definition`

importe la route empruntée. L'architecture distribuée actuelle n'est donc pas batie pour assurer spécifiquement extensibilité, ni qualité de service définie. Le routeur (et le réseau d'ailleurs) des années 1980 a donc été progressivement amélioré sur la base de ce paradigme initial, avec le plan de données et le plan de contrôle attachés aux mêmes équipements, configurés en partie manuellement. Tout s'est complexifié également : nouveaux protocoles, ajouts d'équipements capables de répartir la charge réseau, filtrer les paquets, prévenir de certaines tentatives d'attaque, etc ... Certains éléments de réflexion peuvent éventuellement nous mettre sur la voie d'une complexité qui, à défaut d'être exponentielle, l'est d'avantage que simplement linéaire (c'est du moins une conviction personnelle non vérifiée) :

- Plus il y a d'éléments statiques dans un réseau, et plus la modification en profondeur de celui-ci est coûteuse (puisqu'il faut penser à chaque impact sur les parties statiques).
- Si un problème survient, dans le même ordre d'idée, il est difficile d'avoir une vue globale de ce qui se passe puisqu'aucune vue globale du réseau n'est accessible : il faut vérifier (potentiellement) que chaque élément se comporte correctement et est bien configuré.
- Les interfaces entre switchs, routeurs et autres éléments peuvent varier selon le constructeur, et le logiciel sur les équipements est souvent propriétaire et complexe.

De nombreux problèmes se résolvent avec un niveau d'abstraction supplémentaire [2]. Si le développement de systèmes de plus en plus complexes s'est fait de manière très rapide sur PC, c'est d'abord grâce à la première couche d'abstraction qu'ont concstituées les instructions assembleur, puis à la seconde qu'a été le système d'exploitation. Certaines personnes ont eu l'idée, au lieu de considérer le réseau comme un élément périphérique, de le voir comme un processeur capable d'exécuter des instructions basiques, fournissant de fait un service plus facilement adaptable. C'est sur ce principe que repose le Software Defined Network (SDN). Avec une couche d'abstraction supplémentaire que constitue le protocole choisi pour véhiculer le flot d'instructions (Openflow dans notre cas, mais il y en a d'autres que nous évoquerons succintement plus tard), et un système d'exploitation spécifique (Network Operating System, NOS), l'idée est de découpler les différents chemins qu'empruntent les données et le plan de contrôle, à la manière d'un système d'exploitation qui sépare le code d'un programme et les données qu'il utilise.

1.2 Concepts

On peut poursuivre la dernière analogie (avec un ordinateur) de la manière suivante. Sur un PC classique, on crée et utilise des applications qui reposent sur un système d'exploitation res-

2. `https://en.wikipedia.org/wiki/Fundamental_theorem_of_software_engineering`

ponsable des éléments matériels. C'est la même chose dans un modèle SDN : on souhaite créer des applications "réseau" sans se soucier de la manière dont les éléments de ce dernier vont s'organiser pour répondre à la problématique. On sépare ainsi la couche applicative, le système d'exploitation réseau et la communication entre les entités du réseau.

Pour réaliser cela, il est nécessaire, puisqu'un réseau est constitué d'entités physiquement séparées, de disposer d'un protocole de communication standard entre celles-ci. Mais ça n'est pas suffisant : si tous les éléments communiquent entre eux, encore faut-il qu'ils aient des choses à se dire pour faire circuler, outre les données à faire transiter, les instructions qui vont leur indiquer quoi faire avec ces données. Cela n'est possible que si il existe un cerveau central qui coordonne les opérations (il n'existe pas vraiment d'intelligence collective à ce jour). C'est le rôle du contrôleur SDN. On déporte ainsi l'intelligence humaine déployée dans la configuration de tous les éléments du réseau vers un seul (même si il peut être dupliqué).

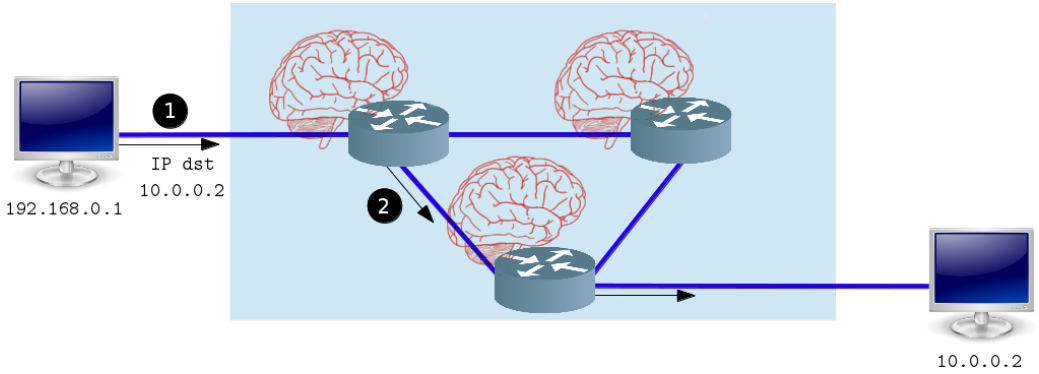


FIGURE 1 – Réseau classique : chaque routeur contient une partie de la logique de contrôle

Les avantages de cette architecture sont multiples :

- D'abord cela réduit grandement la complexité de configuration manuelle et le risque d'erreur (si le système d'exploitation réseau est fiable).
- Cela facilite donc énormément le développement d'applications réseau complexes, puisque la tâche peut être quasiment séparée de sa réalisation physique.
- Les routes optimales sont plus facilement calculables qu'au sein d'un réseau classique : un seul élément gère les différentes distances et métriques qui peuvent changer selon le trafic, et être modifiées à la volée par des applications spécifiques.
- La gestion du réseau devient plus simple, les évènements importants (perte d'un lien, disfonctionnement, ralentissement ...) peuvent être remarqués rapidement, la réaction pouvant être automatique et quasiment instantanée.

3. Schéma extrait du mémoire "Étude d'OpenFlow dans le contexte de la sécurité" de Maxence Tury

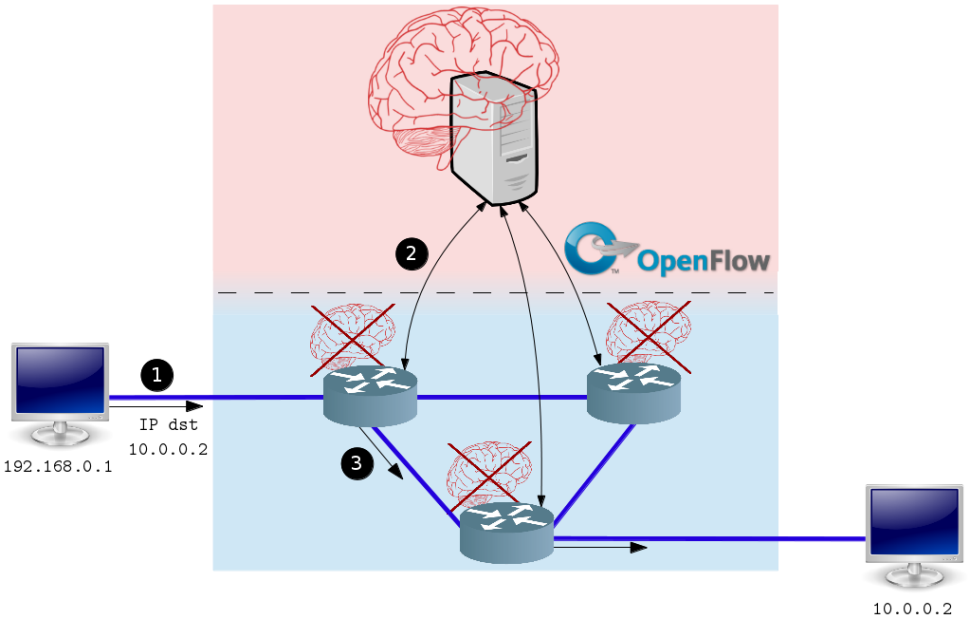


FIGURE 2 – Réseau SDN : "l'intelligence" est déportée vers le contôleur [3]

※ Les coûts matériels sont diminués puisque seuls subsistent les switchs "de base" et le contrôleur qui n'est pas réellement un équipement spécifique. Le reste peut être pris en charge logiciellement au niveau du contrôleur.

Bref, pour résumer, beaucoup plus de flexibilité est permise par cette approche, économisant temps et matériel. Evidemment, l'idée n'est pas nouvelle, mais n'a pas que des avantages. Notamment : la sécurité du contrôleur devient un point brûlant, puisque toute la gestion du réseau provient de lui.

1.3 Historique

La ressource "A Survey of Software-De ned Networking : Past, Present, and Future of Programmable Networks" [4] présente un état de l'art à la date du 19 janvier 2014 (donc assez récemment pour avoir une vue globale de l'évolution de ce type de technologie). En voici un rapide résumé complété :

Assez tôt lors de l'essor d'internet tel que nous le connaissons, des idées pour fournir une sorte d'API réseau ont émergées (milieu des années 1990 environ) : le groupe Open Signaling propose un protocole d'accès universel au matériel (switchs) permettant de distribuer facilement des nouveaux services, pendant qu'Active Networking propose un mécanisme de propagation de code

4. https://hal.inria.fr/hal-00825087/file/hal_final.pdf

que l'équipement réseau exécute lorsqu'il reçoit les paquets encapsulant le code (ce qui pose au passage un énorme problème de sécurité).

Dans le même temps, le groupe DCAN (Devolved Control of ATM Networks) propose une approche qui se rapproche très fortement du paradigme SDN : convaincus que les fonctions de contrôle et de gestion des différents éléments du réseau doivent être séparées du routage des données et déléguées à des équipements spécialisés, ils développent un protocole minimaliste entre contrôleur et autres équipements, à la manière du protocole Openflow aujourd'hui majoritaire dans les réseaux SDN.

Le projet 4D [5] initié en 2004 (et semblant s'être fini un peu avant 2010), ajoute quant à lui des abstractions diverses (découverte des voisins proches et remontée des informations, dissémination d'informations sur l'état général du réseau, puis prise de décision sur la base des informations récoltées). C'est ce genre de projet qui a inspiré l'idée de système d'exploitation réseau, qu'implémentent les contrôleurs SDN.

On peut encore citer NETCONF et Ethane (2006), le premier pouvant être vu comme une extension de SNMP, le second comme un ancêtre immédiat d'openflow (un contrôleur qui décide si et où un paquet devrait être redirigé, et des switchs qui constitués d'une table de flux et d'un canal sécurisé vers le contrôleur).

Openflow a quant à lui précédé l'apparition du terme SDN lors d'expérimentations à Stanford vers 2010 (la première spécification d'Openflow pour la production (1.0.0), a été publiée début 2010).

1.4 Exemples d'applications

En 2011, l'Open Networking Foundation (ONF) est créée. Regroupant des gros acteurs comme Google, Yahoo, Facebook, Verizon, Microsoft ou encore Deutsche Telekom, c'est l'organisme principal qui encourage l'adoption de la technologie SDN, en publiant régulièrement de nouvelles spécifications Openflow.

Google, en 2012, présente, pour la première fois, une architecture SDN pour ses datacenters,

5. `http://www.cs.cmu.edu/~4D/`

utilisant Openflow sur des switchs conçus par eux-mêmes (étant pionniers, leur position étant qu'ils auraient utilisé des switchs existants si ceux-ci implémentaient toutes les fonctionnalités Openflow leur étant nécessaires). Grâce à ce nouveau paradigme, Google affirme (en 2012) obtenir des performances dix fois supérieures en terme de débit, et surtout utiliser 100% [6] de leurs lignes (contrairement aux 30 à 40% en vigueur dans l'industrie, notamment pour garantir un service même en cas de nombreuses pannes, ce qui n'est plus nécessaire avec un réseau SDN qui adapte automatiquement le routage pour pallier aux problèmes).

Microsoft semble également s'être intéressé au SDN pour son service "Azure" depuis quelques années maintenant [7], et de manière générale toutes les figures de proue de l'industrie numérique (celles qui disposent de nombreux serveurs et gèrent des flux énormes et grandissants de données comme Amazon, AT&T, Facebook, ...) se sont plus ou moins annoncées investies dans le processus, possiblement avec leur propre protocole SDN. Si des grosses entreprises ont annoncé l'utilisation de ce type de réseau, les données précises concernant les performances obtenues sont difficilement accessibles, ce qui rend compliquée l'évaluation des avantages réels de SDN.

Par ailleurs, les switchs compatibles Openflow, ou les switchs Openflow seuls, que fournissent (depuis 2011 environ) certaines entreprises majeures du domaine comme Brocade, HP, IBM, Juniper ou encore NEC, Pronto ou Pica8, manquent encore parfois de maturité (RFC parfois interprétée différemment, vulnérabilités spécifiques, ...). L'évolution des versions Openflow est également parfois difficile à suivre (la spécification de la version 1.0.0 (fin 2009) fait 42 pages [8], celle de la dernière version stable (1.5.0, fin 2014) en fait 277 [9]). Or l'industrie dans ce domaine présente une certaine inertie (peu sont les compagnie qui proposent des switchs compatibles avec la dernière version d'Openflow, certaines vendant encore des switchs Openflow 1.0).

Les applications semblent donc d'un premier abord limitées aux datacenters (beaucoup de données à traiter, grande variabilité de la topologie (les machines virtuelles changent souvent d'emplacement/adresse)). En réalité, SDN peut être utilisé de manière effective dans plusieurs cas :

- ※ Réseaux d'entreprises/universités : dans le cas de nombreux équipements/protocoles différents utilisés, SDN permet théoriquement de faciliter le déploiement de politiques réseau com-

6. http://www.networkworld.com/article/2189197/lan-wan/google-s-software-defined-openflow-backbone-drives-wan
html
7. http://www.networkworld.com/article/2937396/cloud-computing/microsoft-needs-sdn-for-azure-cloud.
html
8. http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
9. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/
openflow/openflow-switch-v1.5.0.noipr.pdf

plexes.

- Réseaux optiques : faciliter la transition, la gestion et l'incorporation des réseaux optiques au sein du réseau actuel.
- Infrastructure based WAN (réservé aux entreprises disposant de nombreux serveurs à travers le monde) : à la manière de Google, un moyen de connecter de grosses entités en évitant les goulots d'étranglement. Egalement un moyen envisageable pour un utilisateur de se connecter depuis n'importe quel endroit en disposant des services auxquels il souscrit.
- Infrastructure personnelle, petites entreprises : surveiller le trafic et alerter l'administrateur local ou le fournisseur internet en cas de détection d'activité réseau suspecte (utile notamment dans le cadre de l'IoT).

Au final, une technologie pouvant sembler prometteuse, mais demeurant assez peu utilisée pour plusieurs raisons. Nous allons en étudier deux d'entre elles par la suite : la sécurité générale du réseau, et l'importance capitale du contrôleur qui devient quasiment l'unique clé de voûte du système.

2 Cadre de l'étude

2.1 Problématique et objectifs

Mon stage, dont le sujet n'était pas complètement fixé au départ, a pris la tournure suivante : d'abord une étude des réseaux SDN (ne connaissant pas le domaine), puis début d'expérimentations sur le protocole Openflow, avec Scapy et Wireshark. Ensuite, constitution d'un rapide état de l'art en matière d'attaques (générales) sur les réseaux SDN. Puis, l'orientation s'est faite sur l'étude plus précise d'un contrôleur SDN particulier (ONOS), avec la conception de scénarios d'attaque, suivie de leur réalisation.
Comme on l'a dit au-dessus, le contrôleur SDN est le point névralgique de toute l'infrastructure. Si on le compromet d'une manière où d'une autre, les conséquences peuvent être désastreuses. L'étude a donc eu pour but de déterminer les principaux vecteurs d'attaque envisageables dans ce genre de réseau, principalement concernant le contrôleur ONOS (principalement, parce qu'il est possible d'appliquer une majorité des attaques sur d'autres contrôleurs, même si cela n'a pas été expérimentalement vérifié).
Pour résumer, un audit (se voulant le plus exhaustif possible, même si il est impossible de couvrir l'ensemble des vulnérabilités d'un tel élément logiciel) a été réalisé. Cet audit a été conçu informellement à partir de la méthode STRIDE [10] (utilisée par microsoft à la base, cette manière de modéliser les menaces dans le domaine de la sécurité s'est beaucoup répandue). Certains scénarios ont été expérimentés pour prouver la faisabilité d'attaques précises (donc sous certaines hypothèses qui sont décrites). D'autres faiblesses sont également détaillées dans leur aspect théorique sur la base de sources externes. Bien que n'ayant pas eu accès à une situation réelle de déploiement SDN, j'essaierai de donner une conclusion pas trop biaisée aux tests effectués, et formulerai quelques recommandations et remarques.

2.2 Architecture d'un réseau SDN

Avant toute chose, il est nécessaire de détailler la façon dont un réseau SDN fonctionne, afin que les attaques qui seront évoquées plus tard puissent être bien comprises. Pour cela, on insistera particulièrement sur la description du protocole Openflow, mais aussi sur celle de l'objet de notre étude, ONOS.

10. Spoofing, Tampering, Repudiation, Info disclosure, Denial of service, privilege Escalation

2.2.1 Openflow

Openflow est un des protocoles qui permet de séparer le plan de données et le plan de contrôle (protocole majoritairement utilisé à l'heure actuelle, étant non propriétaire et porté par l'ONF). Pour l'utiliser, il est nécessaire de disposer de switchs compatibles, c'est à dire des switchs capables de gérer des paquets Openflow. Ainsi, les décisions prises au niveau du contrôleur sont transmises aux switchs concernés par le biais de ce protocole. Pour que l'abstraction de la gestion du réseau soit intéressante, le protocole permet aux switchs une gestion assez fine des paquets reçus à leur niveau, et ce jusqu'au niveau 4 (pour TCP comme on va le voir après). C'est le principal élément de ce qu'on appelle l'interface sud d'un réseau SDN (interface entre contrôleur et entités matérielles) :

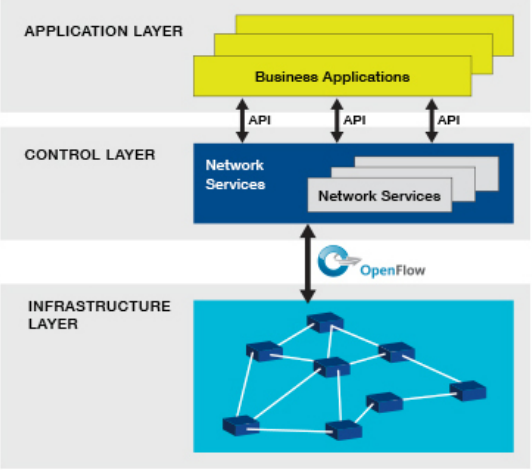


FIGURE 3 – Différentes interfaces, Openflow = interface sud

Chaque switch (qu'il soit compatible Openflow ou virtuel) consiste en plusieurs tables de flux et une table de groupe (non présente en version 1.0), ainsi qu'un (ou plusieurs) canal(aux) vers le(s) contrôleur(s) (sécurisé(s) via TLS obligatoire en version 1.0 mais obligation supprimée dès la version 1.1 pour des raisons de facilité de déploiement). La description du fonctionnement qui suit est celle de la version 1.3 du protocole (ça n'est donc ni la dernière version, ni la première, mais celle que j'ai principalement utilisée durant le stage, Wireshark la disséquant complètement).

La notion de flux est essentielle. Un flux est constitué de 3 parties :

- La première une règle qui filtre les paquets : en fonction des attributs du paquet à l'entrée du switch (port d'entrée physique, adresse ethernet source, adresse ethernet destination, type de paquet, VLAN id, VLAN priority, adresse IP source, adresse IP destination, protocole IP, ToS IP, TCP port source, TCP port destination entre autres). Il est possible de générer des filtrages généraux avec l'utilisation de jokers (wildcard en anglais) sur les

champs souhaités (par exemple, si on veut autoriser toutes les adresses ethernet source ayant pour adresse IP x.x.x.x, on pourra utiliser un joker sur le champ adresse ethernet source).

- ※ La seconde est un compteur qui permet de tenir à jour, si le switch le permet, des statistiques sur l'utilisation du flux.
- ※ La troisième est une action à appliquer en cas de correspondance du paquet : si le paquet remplit les conditions du filtre, plusieurs types de traitements sont possibles. Entre autres : envoyer le paquet au contrôleur, rediriger le paquet vers un port physique spécifique, vers une table de flux, vers tous les ports sauf le port d'entrée, vers les switchs voisins mis à jour par spanning tree, supprimer le paquet, ou encore modifier certains champs avant redirection, rediriger le paquet vers une queue. Bref, toutes les opérations envisageables sur un paquet. Certaines de ces actions doivent être prises en charge par les switchs Openflow pour que ceux-ci puissent être considérés comme tels. D'autres actions sont optionnelles (par exemple la redirection vers les switchs mis à jour par spanning tree).

Chaque nouveau flux ajouté au switch par le contrôleur l'est dans une table de flux spécifiée. Ainsi, plutôt que d'avoir à organiser relativement la priorité de chaque flux, des regroupements peuvent se faire par table pour factoriser certains traitements. Le switch parcourt chaque table de flux (en considérant le premier flux de chaque table) jusqu'à trouver un filtrage correct pour le paquet. A ce moment, le paquet parcourt et subit les traitements de chacun des flux dans la table trouvée (c'est la notion de pipeline Openflow) jusqu'à ce qu'une action de redirection soit trouvée. La redirection peut également être faite vers une table de flux de priorité inférieure (pour éviter qu'un paquet boucle indéfiniment).

Basiquement, on peut résumer le protocole Openflow comme étant le protocole permettant :

- ※ de mettre à jour ces tables de flux : ajouts, ajouts partiels (par exemple ajouts de précisions à un flux avec joker), suppressions, modifications, etc ....
- ※ d'obtenir des statistiques sur certains éléments (en Openflow 1.3, on peut accéder aux statistiques des flux, des tables, des ports, des queues, des groupes, ou encore du débit (pour la qualité de service)).
- ※ d'obtenir des informations sur les entités du réseau (nom de l'équipement, nom du fabricant, débit supporté ...)

C'est un protocole qui s'établit de manière classique sur une session TCP (éventuellement surmonté d'une session TLS), habituellement sur les ports 6633 ou 6653 (ce dernier étant maintenant alloué pour cet usage par l'IANA).
Il est constitué :

- ※ de messages symétriques (Hello, Echo), qui s'utilisent pour ou démarrer une session Openflow ou s'assurer que les deux parties sont encore connectées

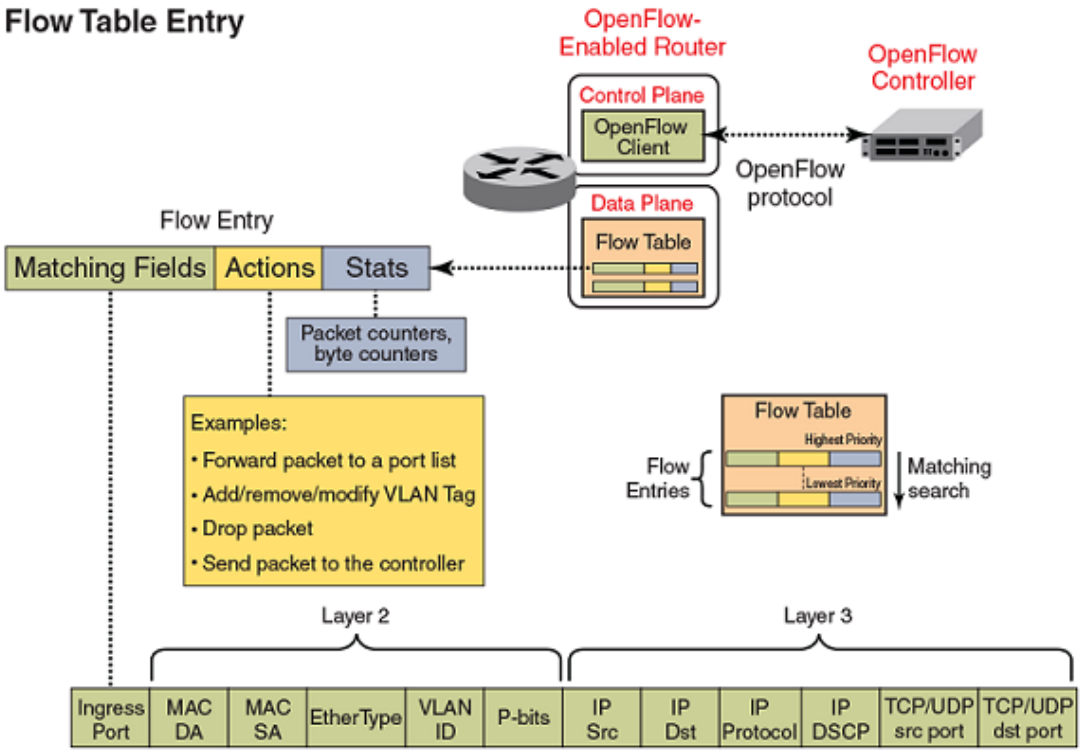
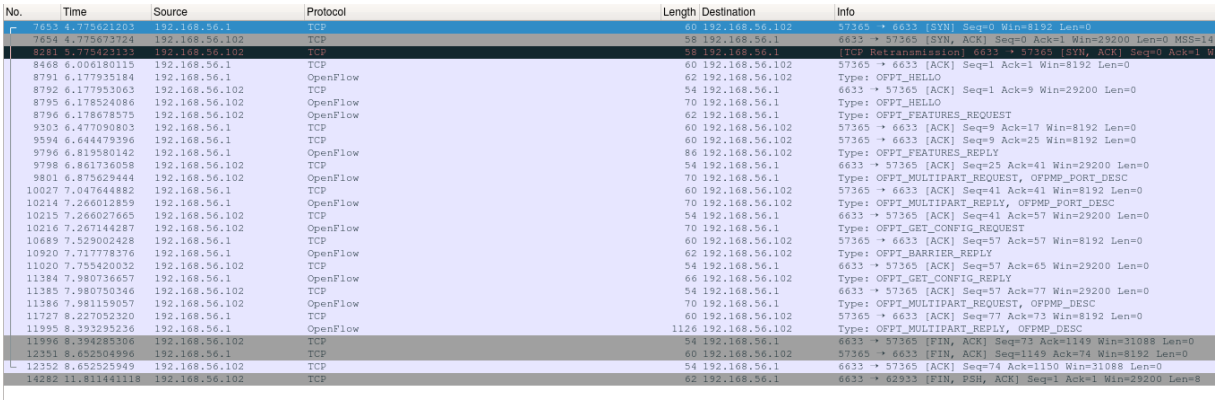


FIGURE 4 – Flux et table de flux

- de messages contrôleur vers switch (ressemblant à du requête/réponse), généralement les messages qui permettent au contrôleur d'obtenir des informations sur le switchs et de lui donner des ordres. Mais aussi, et c'est important, un message qui permet directement d'insérer du trafic dans le réseau. Ainsi, lorsque le switch envoie un paquet qu'il ne sait pas traiter au contrôleur, celui-ci peut le renvoyer au switch (après l'avoir traité et éventuellement modifié), ce qui évite de perdre le paquet [11].
- de messages asynchrones émis par les switchs, qui sont par exemple susceptibles d'informer le contrôleur lorsqu'ils ont modifié, supprimé ou ajouté un flux, mais également lorsqu'un paquet ne correspond à aucun flux, qu'une erreur survient ou bien qu'un port physique change de configuration. Le message asynchrone le plus important est sûrement celui qui survient lorsque le switch doit envoyer le paquet au contrôleur [12] (soit parce qu'une règle le demande explicitement, soit parce que le switch ne peut pas traiter le paquet et que la règle par défaut associée demande un envoi au contrôleur).

La figure précédente provient de mes expérimentations, j'ai en effet été amené durant mon stage à créer une sorte de mini-switch virtuel très basique avec Scapy, ce qui m'a forcé à implémenter à la fois la pile TCP et la session Openflow . Même si par la suite cela m'a été peu utile pour

11. On appelle alors ce paquet OFPT_PACKET_OUT (selon la spécifiation) qu'on abrégera par la suite en PACKET_OUT
12. Paquet appelé OFPT_PACKET_IN (selon la spécifiation) qu'on abrégera par la suite en PACKET_IN



No.	Time	Source	Protocol	Length	Destination	Info
7653	4.775621203	192.168.56.1	TCP	60	192.168.56.102	57365 → 6633 [SYN] Seq=0 Win=8192 Len=0
7654	4.775673724	192.168.56.102	TCP	58	192.168.56.1	6633 → 57365 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=14
8281	5.775423133	192.168.56.102	TCP	58	192.168.56.1	[TCP Retransmission] 6633 → 57365 [SYN, ACK] Seq=0 Ack=1 W
8468	6.006180115	192.168.56.1	TCP	60	192.168.56.102	57365 → 6633 [ACK] Seq=1 Ack=1 Win=8192 Len=0
8791	6.177935184	192.168.56.1	OpenFlow	62	192.168.56.102	Type: OFPT_HELLO
8792	6.177953063	192.168.56.102	TCP	54	192.168.56.1	6633 → 57365 [ACK] Seq=1 Ack=9 Win=29200 Len=0
8795	6.178524086	192.168.56.102	OpenFlow	70	192.168.56.1	Type: OFPT_HELLO
8796	6.178678575	192.168.56.102	OpenFlow	62	192.168.56.1	Type: OFPT_FEATURES_REQUEST
9303	6.477090803	192.168.56.1	TCP	60	192.168.56.102	57365 → 6633 [ACK] Seq=9 Ack=17 Win=8192 Len=0
9594	6.644479396	192.168.56.1	TCP	60	192.168.56.102	57365 → 6633 [ACK] Seq=9 Ack=25 Win=8192 Len=0
9796	6.819580142	192.168.56.1	OpenFlow	86	192.168.56.102	Type: OFPT_FEATURES_REPLY
9798	6.861736058	192.168.56.102	TCP	54	192.168.56.1	6633 → 57365 [ACK] Seq=25 Ack=41 Win=29200 Len=0
9801	6.875629444	192.168.56.102	OpenFlow	70	192.168.56.1	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
10027	7.047644882	192.168.56.1	TCP	60	192.168.56.102	57365 → 6633 [ACK] Seq=41 Ack=41 Win=8192 Len=0
10214	7.266012859	192.168.56.1	OpenFlow	70	192.168.56.102	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
10215	7.266027665	192.168.56.102	TCP	54	192.168.56.1	6633 → 57365 [ACK] Seq=41 Ack=57 Win=29200 Len=0
10216	7.267144287	192.168.56.102	OpenFlow	70	192.168.56.1	Type: OFPT_GET_CONFIG_REQUEST
10689	7.529002428	192.168.56.1	TCP	60	192.168.56.102	57365 → 6633 [ACK] Seq=57 Ack=57 Win=8192 Len=0
10920	7.717778376	192.168.56.1	OpenFlow	62	192.168.56.102	Type: OFPT_BARRIER_REPLY
11020	7.755420032	192.168.56.102	TCP	54	192.168.56.1	6633 → 57365 [ACK] Seq=57 Ack=65 Win=29200 Len=0
11384	7.980736657	192.168.56.1	OpenFlow	66	192.168.56.102	Type: OFPT_GET_CONFIG_REPLY
11385	7.980750346	192.168.56.102	TCP	54	192.168.56.1	6633 → 57365 [ACK] Seq=57 Ack=77 Win=29200 Len=0
11386	7.981159057	192.168.56.102	OpenFlow	70	192.168.56.1	Type: OFPT_MULTIPART_REQUEST, OFPMP_DESC
11727	8.227052320	192.168.56.1	TCP	60	192.168.56.102	57365 → 6633 [ACK] Seq=77 Ack=73 Win=8192 Len=0
11995	8.393295236	192.168.56.1	OpenFlow	1126	192.168.56.102	Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
11996	8.394285306	192.168.56.102	TCP	54	192.168.56.1	6633 → 57365 [FIN, ACK] Seq=73 Ack=1149 Win=31088 Len=0
12351	8.652504996	192.168.56.1	TCP	60	192.168.56.102	57365 → 6633 [FIN, ACK] Seq=1149 Ack=74 Win=8192 Len=0
12352	8.652525949	192.168.56.102	TCP	54	192.168.56.1	6633 → 57365 [ACK] Seq=74 Ack=1150 Win=31088 Len=0
14282	11.811441118	192.168.56.102	TCP	62	192.168.56.1	6633 → 62933 [FIN, PSH, ACK] Seq=1 Ack=1 Win=29200 Len=8

FIGURE 5 – Capture d'une session Openflow réalisée sous Wireshark entre le switch virtuel crée en 192.168.56.1 et le contrôleur en 192.168.56.102

réaliser mes scénarios d'attaque, cela m'a permis de bien comprendre le protocole.

Le lecteur désirant rentrer dans les détails techniques peut se référer à la spécification de la version 1.3 [13].

2.2.2 Contrôleur SDN

Le contrôleur est, comme on l'a déjà dit précédemment, l'élément central du réseau SDN, puisqu'il offre au niveau de son interface nord, une API pour développer des applications réseau, et, au niveau de son interface sud, il contrôle les entités réseaux se chargeant du plan de données, avec le protocole Openflow.

Pour fonctionner correctement, le contrôleur doit avoir la représentation interne la plus exacte possible de la topologie réseau qu'il dirige. Pour cela, Openflow prévoit certains paquets spécialisés. Mais ce n'est pas suffisant, puisque les switchs eux-mêmes ne sont pas capables de renseigner le contrôleur sur la topologie alentours. C'est pourquoi certains mécanismes sont mis en place (qui dépendent généralement du contrôleur, même si, devant utiliser des protocoles classiques compréhensibles par des switchs, les possibilités restent limitées).
Le mécanisme que j'ai été amené à constater est celui de l'utilisation de paquets LLDP fabriqués par le contrôleur et envoyés aux switchs sous forme de PACKET_OUT. En recevant un tel paquet, un switch va le retransmettre en broadcast aux switchs alentours, qui, normalement, sont configurés pour le renvoyer en PACKET_IN au contrôleur (comportement par défaut, si aucun flux gérant ce type de paquet n'est spécifié, ce qui est préférable). Or, un PACKET_IN encapsule toutes les informations nécessaires au contrôleur pour mettre à jour la topologie locale :

13. https ://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (106 pages, dont 49 d'explications)

en vérifiant que c'est bien lui qui est à l'origine de l'émission du paquet LLDP initial (avec un champ spécial par exemple), il sait que le switch émetteur du PACKET_IN est relié au switch auquel il avait précédemment envoyé un PACKET_OUT, ces premiers étant des encapsulations de paquets réels circulant sur le réseau, on peut donc y lire des adresses ethernet, des adresses IP, ... La question de la confiance relative à la réception de tels paquets est cruciale et on va voir par la suite qu'il est relativement aisé d'attaquer le contrôleur par ce biais.

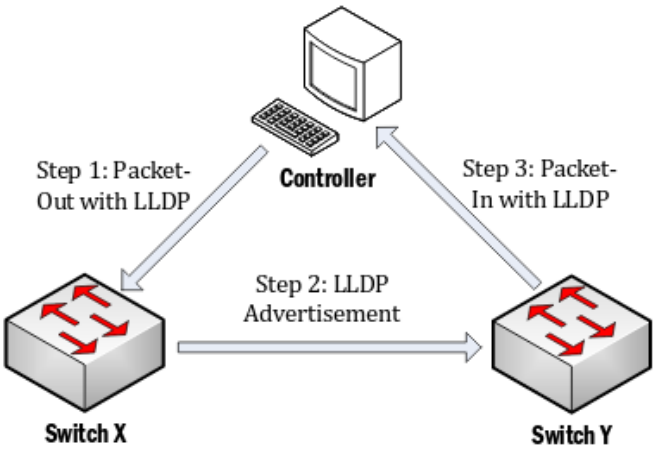


FIGURE 6 – Mécanisme de découverte de topologie par envoi de paquets LLDP

Au sein du contrôleur même, on trouve tout le logiciel nécessaire pour lier les informations reçues depuis les différentes entités du réseau aux intentions de plus haut niveau émises. Comme sur un système d'exploitation classique, on peut trouver une abstraction plus ou moins riche : selon la maturité du contrôleur et le dynamisme de la communauté qui le porte, on trouvera ainsi de nombreuses différences dans la quantité de développement à fournir pour arriver à un même résultat. Par exemple sur ONOS il est possible de spécifier uniquement une intention de haut niveau qui sera automatiquement traduites en règles qui seront si c'est possible envoyées aux switchs.

Le contrôleur offre finalement une API plus ou moins fournie qui permet aux utilisateurs d'écrire des applications en disposant d'abstractions susceptibles de lui éviter l'écriture de code fastidieux.

Parmi les contrôleurs SDN les plus connus, on trouve notamment NOX (premier contrôleur SDN, 2008, rendu open source depuis), POX (juin 2011, en python), OpenDayLight (avril 2013, open source, crée par The Linux Foundation), ONOS (l'objet final de ce stage, décembre 2014, open source, développement repris par The Linux Foundation en 2015), et bien d'autres (Beacon, RoseMary, Ryu ...). Certains projets se ressemblent énormément au niveau des choix effectués

(langages utilisés, paradigmes ...). Par exemple OpenDayLight et ONOS sont deux contrôleurs très proches dans ce qu'ils offrent (java, architecture OSGi, sécurité vue comme un élément crucial, ...).

2.2.3 ONOS

ONOS [14] est un contrôleur SDN open source récent (début en décembre 2014, la fondation Linux arrive en octobre 2015 dans le projet), écrit en java, déployable avec Maven et utilisant apache-karaf comme conteneur OSGi (qui fournit entre autre l'interface utilisateur permettant l'interaction avec le contrôleur). La version actuelle est Hummingbird (octobre 2016) (1.6), la prochaine Ibis (cycles de développement d'environ 6 mois). C'est un projet basé sur la technique [15] :

> goal is to « provide an environment that thrives on technical meritocracy. Merit is based on technical contribution, not on financial contribution. »

Le contrôleur est architecturé de la manière suivante :

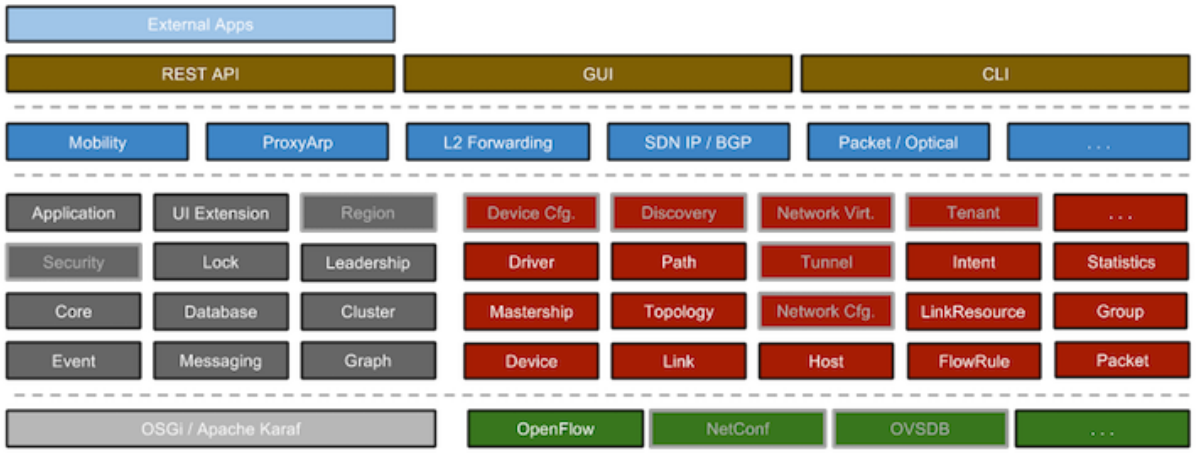


FIGURE 7 – Architecture logicielle d'ONOS (schéma extrait du site officiel)

- Au Sud : une API (southbound) gérant plusieurs protocoles (dont Openflow (toutes les versions jusqu'à la version 1.5, la version 1.6 étant en cours de prise en charge)). C'est la partie d'ONOS qui se charge de la communication avec les switchs. Il est ainsi possible de prendre en charge de nouveaux protocoles ou de nouveaux drivers.
- Sur le côté : un protocole d'échange entre contrôleurs. Cela permet un contrôle partagé du réseau. Pour cela, des informations sur la topologie de celui-ci doivent être échangées

14. Open Network Operating System
15. `http://onosproject.org/governance/`

entre contrôleurs. Le protocole en question n'est pas standardisé.

- Au Nord : une API (northbound) permettant d'écrire des applications utilisant les ressources offertes par le contrôleur. Si le secure mode est activé (nous reviendrons plus en détail sur cela ultérieurement), l'ensemble des méthodes utilisables est restreint.
- Au Nord encore : une interface utilisateur fournie par Karaf et composée de 3 parties : une API REST accessible sur le port 8181 (configurable), une interface web (permettant de visualiser l'état du réseau, les applications lancées ...) accessible sur ce même port, et une CLI accessible en SSH sur le port 8101 (ou bien directement sur le contrôleur, là encore tout est configurable).

2.3 Surface d'attaque

Pour tester le contrôleur, l'installation suivante est actuellement réalisée : - une machine virtuelle utilisant l'émulateur réseau mininet, qui crée des switchs et hôtes virtuels, et qui permet de générer du trafic réseau SDN. - une machine virtuelle (ubuntu server) sur laquelle le contrôleur ONOS est installé et est accessible. - une machine virtuelle (ubuntu desktop) permettant d'intéragir avec le contrôleur en SSH (on peut aussi utiliser la machine non virtuelle).

On va donc appliquer la méthode STRIDE aux divers éléments et interfaces qui composent ONOS, à la manière de ce qui est présenté dans un article de l'institut Fraunhofer [16].

2.3.1 Menaces au niveau de l'interaction avec les switchs

Le contrôleur reçoit et interprête des données d'éléments externes. Cela signifie que, si l'une des entités avec qui il communique est malveillante, celle-ci a la possibilité d'agir négativement sur le contrôleur. Concernant la méthode STRIDE appliquée à l'interface sud, on trouve majoritairement 4 menaces :

- Spoofing (S) : possibilité pour un élement de se faire passer pour ce qu'il n'est pas (un switch se faisant passer pour un autre switch par exemple, ...).
- Tampering (T) : possibilité de modifier le flux de données des switchs en se plaçant sur le chemin du contrôleur (man in the middle). Cette partie est rapide à étudier puisque TLS, si il est correctement utilisé, permet d'éviter toute modification du flux.
- Information disclosure (I) : possibilité d'obtenir les flux d'informations entre les éléments du réseau et le contrôleur (là encore si TLS est activé cela réduit la menace à son minimum).

16. `http://publica.fraunhofer.de/eprints/urn_nbn_de_0011-n-4046948.pdf`

- Denial of service (D) : possibilité de surcharge des interfaces réseau, par exemple un switch non désiré sur le réseau qui surcharge le contrôleur de messages, de manière intelligente (en sachant ce qui ralentira le plus le contrôleur) ou non.

Dans la suite, on testera S,T,D. Mais toujours relativement au contrôleur, c'est à dire qu'on regardera si le contrôleur agit comme il est supposé réagir, permettant ou non l'attaque. Et on constatera ou non la généricité des attaques.

2.3.2 Menaces au niveau de l'interaction utilisateur

Le contrôleur exécute potentiellement des applications fournies par des tiers. Si un utilisateur importe une application malveillante sur le contrôleur, cela peut avoir des répercussions sur tout le réseau. Concernant la méthode STRIDE appliquée à l'interface nord, on trouve majoritairement 5 menaces :

- Spoofing (S) : (abus de langage ici, mais c'est la catégorie qui se rapproche le plus de la réalité) possibilité de modifier le comportement de certaines applications avec des droits non adaptés.
- Tampering (T) : possibilité de modifier le flux de données des applications en se plaçant sur le chemin du contrôleur (man in the middle). Cette partie est rapide à étudier puisque là encore, TLS, si il est correctement utilisé, permet d'éviter toute modification du flux d'information.
- Repudiation (R) : possibilité pour une application de nier certaines actions dont elle est l'origine.
- Information disclosure (I) : possibilité d'obtenir des informations sur d'autres applications, sur l'état général du contrôleur, ...
- Denial of service (D) : possibilité d'action néfaste sur le contrôleur (modification de la topologie, dégradation du débit offert par le contrôleur, ...).

Dans la suite, on testera S,T,D et I. Encore une fois cela sera fait par rapport à ONOS, ce qui ici se justifie d'avantage (aucun standard n'existant au niveau de l'interface nord, celle-ci peut varier beaucoup selon le contrôleur). De plus, ONOS propose un mécanisme de sécurité intéressant à étudier qui est le Security Mode, mis en place depuis la version Drake (1.3) du contrôleur.

Ce module, qui continue d'être amélioré, rajoute la possibilité de définir des permissions fines (ce qui se traduit par le droit d'utiliser ou non certaines fonctions de l'API) par rôle aux applications, et sera légèrement détaillé plus tard. Les attaques qui constitueront cette partie seront donc moins génériques que les attaques menées au niveau de l'interface sud.

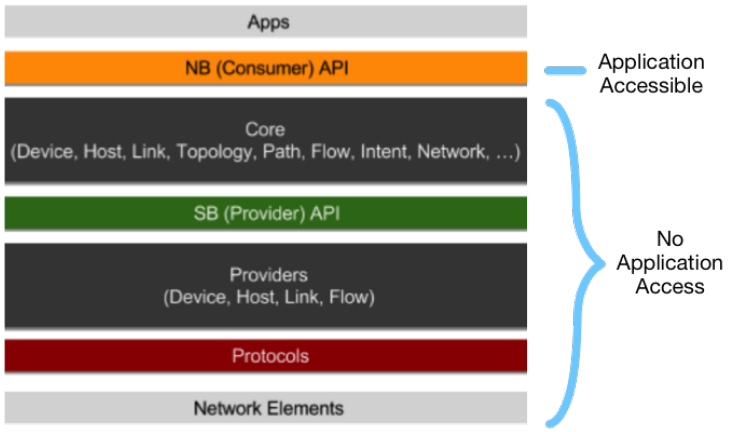


FIGURE 8 – Secure mode activé : accès aux fonctions critiques restreint lors de l'utilisation de l'API par certaines applications

2.3.3 Autres menaces

Nous avons évoqué les menaces qui pesaient sur les interfaces nord et sud, mais il existe encore d'autres menaces :

- Sur le contrôleur en lui même : bien que cela soit laborieux et que je n'aie pas réussi à le faire durant mon stage, il n'est pas impossible qu'il existe des vulnérabilités dans le code source du contrôleur. Sans aller jusqu'à l'exécution de code arbitraire (sachant que le code est en java, donc cela nécessite normalement une faille de la machine virtuelle, puisqu'à aucun endroit du code la possibilité est offerte d'exécuter du code externe), il est envisageable de trouver des enchaînements (mise à jour de variables bien choisies, modification de l'état interne du contrôleur) qui réalisent des actions non prévues. Cela demande cependant une connaissance excellente du code, ce qu'il est très compliqué d'obtenir vu le peu de documentation qui est offerte lorsqu'on souhaite se plonger dans le coeur du contrôleur et la complexité générale de l'ensemble.

  Sur le contrôleur on peut aussi trouver un problème de répudiation : bien que les logs soient sauvegardés et soient assez complets, rien n'empêche à l'heure actuelle de les supprimer (le but étant plus de fournir du debug au développeur qu'un outil d'analyse forensique voire une preuve certaine des évènements passés).

  Enfin toujours sur le contrôleur, on peut trouver des problèmes de DoS, comme par exemple en 2015 avec la CVE-2015-7516 [17]. Cela revient là encore à utiliser les faiblesses du code pour avoir une action non prévue néfaste sur les performances globales.

17. https ://wiki.onosproject.org/display/ONOS/Security+advisories

- Sur les échanges inter-contrôleurs : le concept SDN prévoit la possibilité de gestion à plusieurs contrôleurs du réseau SDN. Pour cela, il est nécessaire que les contrôleurs partagent entre eux la topologie à laquelle ils ont accès. Cela introduit une vulnérabilité supplémentaire puisque sans authentification mutuelle, il existe un risque de parler à un contrôleur malveillant envoyant de fausses informations.
- Sur les stations d'administration et de déploiement : comme sur un réseau classique, si on compromet les machines utilisées pour gérer le réseau ou distribuer les mises à jour logicielles (social engineering, compromission de l'environnement (DNS ou ARP spoofing, ...)), on a théoriquement un accès privilégié au contrôleur qui permet donc sans y être autorisé d'y apporter des modifications importantes.
- Sur les éléments du réseau : bien que cela demeure peu probable, les mêmes attaques que sur un réseau classique sont toujours possibles. Elles permettent ensuite d'utiliser les attaques sur l'API southbound.

Le très récent site regroupant entre autres les projets Security-mode ONOS, Delta et Barista [18] (nous en reparlerons dans la conclusion) nous donne un bon récapitulatif d'une partie des attaques que nous allons détailler et qui s'inscrivent dans les catégories précédentes :

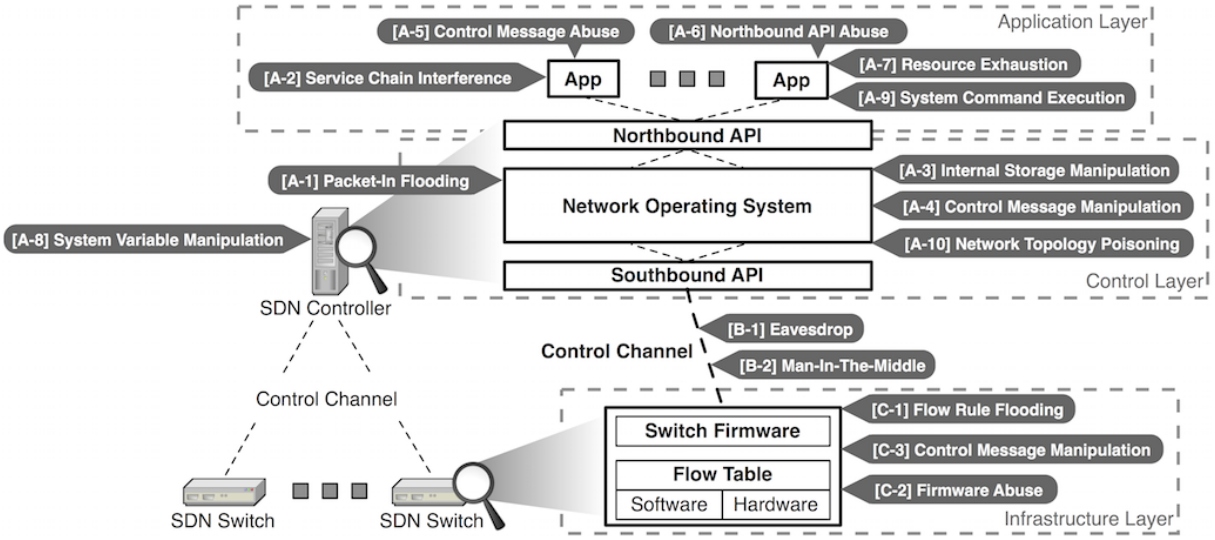


FIGURE 9 – Principales menaces sur un réseau SDN

2.4 Scénarios envisagés

Pour éclairer de manière expérimentale le large spectre de menaces auquel peut être soumis un réseau SDN, j'ai mis en place au fur et à mesure 6 preuves de concept, certaines n'étant pas très

18. http ://sdnsecurity.org

complexes mais démontrent toutefois certaines faiblesses. Même si parmi les scénarios envisagés il y en a certains qui sont spécifiques à ONOS (notamment ceux qui concernent l'interface nord), on verra que les attaques sont globalement les mêmes que dans un réseau classique, avec toutefois des impacts plus lourds.

Pour rester dans la nomenclature précédente, voici les attaques envisagées :

- Scénario 1 : Tampering et Information disclosure (interface sud)
  But : Intercepter et modifier les communications sur le plan de données ou de contrôle si TLS n'est pas activé.
- Scénario 2 : Spoofing et DoS (interface sud)
  But : Altérer la topologie estimée par le contrôleur en usurpant l'identité d'un switch ou en inventant un faux switch et en créant des faux messages LLDP.
- Scénario 3 : DoS (interface sud)
  But : Réduire fortement le débit au niveau de certains noeuds par envoi d'un très grand nombre de paquets dont on espère qu'ils vont chacun aboutir à la création d'une règle au niveau du contrôleur. Ceci afin de surcharger les tables de flux des switchs visés.
- Scénario 4 : DoS (interface nord)
  But : Altérer les performances du contrôleur, tester certaines permissions critiques avec le Secure-Mode.
- Scénario 5 : Information disclosure (interface nord)
  But : A partir d'une application banale qui n'a pas le droit de regarder quelles sont les autres applications présentes sur le contrôleur, observer quels éléments peuvent quand même être rendus accessibles sans que cela soit explicitement prévu.
- Scénario 6 : Spoofing (interface nord)
  But : Tester la frontière entre permission liée à une application et permission liée à l'API REST.

Chacun de ces 6 scénarios est détaillé séparément dans la partie suivante.

3 Audit

3.1 Man in the middle au niveau de l'interface sud

Prérequis/Hypothèses :
- Un switch malveillant connecté à un autre switch dans le réseau local
- Une machine A connectée directement à un switch s1 dans un sous réseau accessible depuis le switch malveillant
- Une machine B connectée indirectement à un switch s2
- A cherche à joindre B
- Le switch s1 a ses tables de flux par rapport au paquets ARP susceptibles d'être crées à partir de A vers B vides (c'est à dire aucune règle n'est susceptible d'appliquer une action prédéfinie aux paquets ARP provenant de A vers B (et donc génération d'un PACKET_IN en conséquence)).
- Une topologie en partie connue (IP des hôtes à usurper) est un plus

Buts :
- Tester la capacité du contrôleur à détecter des Man in the Middle, ce qui devrait s'avérer finalement plus simple que dans un réseau normal (en effet le contrôleur dispose normalement d'une vision globale de la topologie réseau à partir des switchs qu'il contrôle.

Déroulement :
- La machine A cherche à contacter la machine B (ping par exemple)
- Une requête ARP est donc envoyée depuis A et s1 la retransmet au contrôleur n'ayant pas encore d'action associée au flux
- Le contrôleur renvoie le paquet en PACKET_OUT sur le switch s1 qui le rediffuse
- Notre switch malveillant répond plus rapidement que l'équipement concerné à la requête ARP et cherche à intercepter la communication entre A et B voire à la supprimer

Dans la pratique, l'attaque se passe exactement comme dans un réseau classique : le switch malveillant envoie de très nombreuses fausses requêtes ARP en broadcast pour altérer les tables ARP des équipements cibles. On utilisera donc ettercap pour mener l'attaque.

Détails techniques :
Se référer au scénario 1 décrit dans l'annexe (page).

Résultat :
Comme sur un réseau classique, l'attaque fonctionne (on est à la fois en mesure d'intercepter et de modifier le trafic entre A et B). Cela s'explique facilement puisque les mécanismes utilisés pour le routage et le transport sont les mêmes. Si TLS n'est pas utilisé au sud on peut envisager l'attaque en plus sur le plan de contrôle, et donc pousser ses propres règles sur l'entité visée.

Parades proposées :

Dans un réseau SDN, il me semble faisable de détecter et contrer ce genre d'attaque beaucoup plus facilement que dans un réseau classique. Si on enlève la solution TLS avec authentification mutuelle (qui permet de supprimer ce problème au niveau du plan de contrôle mais pas au niveau du plan de données, le switch malveillant ne pouvant plus communiquer avec le contrôleur mais le peut toujours avec les autres switchs), on peut proposer 2 parades.

La première tient à la "signature" de l'attaque. Dans le cas d'une topologie inconnue par le switch malveillant, celui-ci va envoyer de nombreux paquets ARP pour découvrir les éléments présents sur le réseau. Or si aucune règle spéciale n'est présente sur les switchs, les paquets sont envoyés au contrôleur. Ce dernier peut donc détecter, si les envois sont trop rapprochés par exemple, une activité anormale en provenance d'une même adresse mac.

La seconde est encore liée à la capacité du contrôleur à disséquer les paquets qu'il reçoit. En effet, lors de l'attaque, il va recevoir des paquets ARP portant une adresse IP connue (la cible) associée à une adresse MAC ne correspondant pas à la description de l'hôte qu'il détient. Si il analyse les paquets ARP reçus, il peut donc détecter l'attaque. L'inconvénient de cette méthode est l'obligation de traiter chaque paquet ARP reçu, ce qui peut être éventuellement utilisé dans un but de DoS.

Limitations/Impact/probabilité :

Comme sur un réseau traditionnel, l'attaque ne fonctionne qu'au sein d'un réseau local donc l'attaquant doit avoir accès au réseau. De plus, même si les impacts de ce genre d'attaque peuvent être importants (la communication passant par une troisième entité, tout peut être modifié entre les équipements concernés) :

-d'une part l'utilisation d'un chiffrement entre contrôleur et switch permet théoriquement d'éviter à un élément non authentifié de pouvoir modifier de manière conséquente le réseau (même si la gestion d'une PKI fiable au sein d'un réseau SDN est complexe à mettre en œuvre).

-d'autre part comme on l'a dit au dessus, le contrôleur SDN est mieux armé pour répondre à ce genre d'attaque qu'un réseau classique, au prix d'un surcoût éventuel en ressources (gestion de tous les paquets ARP depuis le contrôleur).

3.2 Altération de la topologie depuis l'interface sud

Prérequis/Hypothèses :
- Une entité malveillante connectée au réseau local
- Une topologie en partie connue (adresse mac de deux switchs)
- 2 switchs s1 et s2 non reliés entre eux

Buts :
L'hôte malveillant (en se faisant passer pour un switch) fait croire au contrôleur qu'il existe un lien entre lui et s1, ainsi qu'entre lui et s2. Si la manière dont ONOS calcule les plus courts chemins peut être exploitée et que l'hôte arrive à faire croire que ces liens sont rapides, il est possible que le contrôleur crée un nouveau lien logique entre s1 et s2. Cela engendre un déni de service puisque les paquets qui correspondront aux règles ajoutées sur les switchs s1 et s2 empruntant le faux lien seront envoyés à notre entité malveillante, pouvant les modifier ou tout simplement les détruire : on peut alors aboutir à un "trou noir" dans le réseau.

Déroulement :
Notre hôte malveillant envoie de faux paquets LLDP en multicast pour simuler un lien entre lui, le switch s1 et le switch s2 (en utilisant le mécanisme de découverte de topologie d'ONOS évoqué à la page 15).

Détails techniques :
Se référer au scénario 2 décrit dans l'annexe (page).

Résultat :
L'attaque, en partie reproduite depuis l'article "Poisoning Network Visibility in Software-Defined Networks : New Attacks and Countermeasures" [19], fonctionne dans certains cas, permettant de stopper le contact entre s1 et s2. Je n'ai pas trouvé comment faire fonctionner l'attaque à chaque essai. Il est également envisageable d'intercepter/modifier du trafic réseau avec cette méthode même si je ne l'ai pas fait.

Parades proposées :
Une solution intéressante proposée dans le document associé à l'attaque est de rajouter un champ dans les paquets LLDP envoyés par le contrôleur qui contienne une partie authentification : par exemple n'accepter des paquets LLDP que lorsqu'un champ supplémentaire crée par le contrôleur et basé sur certaines caractéristiques du switch auquel il est envoyé est vérifié (mécanisme de signature). Cela résiste à la fabrication de paquets sur un hôte, mais ne résiste pas si l'attaquant dispose d'un switch connecté au réseau qui est capable de recevoir des paquets LLDP. Toutefois,

19. `http://www.internetsociety.org/sites/default/files/10_4_2.pdf`

cela octroie une sécurité supplémentaire non négligeable puisqu'il devient impossible de mettre en œuvre cette attaque si on n'est pas physiquement connecté.

Limitations/Impact/probabilité :

Là encore l'attaquant doit faire partie du réseau local. De plus, la détermination de la topologie (même supposée optimale) trouvée par le contrôleur n'est pas forcément simple à prévoir, et ce n'est pas dit que le faux lien qu'on indique sera effectivement utilisé par le contrôleur.
Donc encore une fois on a une attaque avec une probabilité (très) faible et un impact fort. On peut noter que l'avantage SDN précédent (obtenir beaucoup d'information locale pour construire une topologie globale) se retourne dans cette situation contre lui : vu que tout est centralisé, si on arrive à modifier la vision du réseau du contrôleur les conséquences sont plus graves contrairement à un réseau classique où il faudrait potentiellement modifier un grand nombre de routeurs avant d'arriver à un point de déni de service équivalent.

3.3 Deni de service au niveau de l'interface sud

Prérequis/Hypothèses :
- Un switch malveillant connecté à un switch du réseau ou directement au contrôleur
- Une topologie en partie connue (adresse mac et IP des switchs à attaquer)

Buts :
Surcharger les tables de flux d'un ou de plusieurs switch(s) pour provoquer un deni de service (moins de bande passante).

Déroulement :
Notre switch malveillant peut envoyer à ses switchs voisins des paquets avec une adresse IP source, une adresse MAC source, un VLAN id, un type de service, un port TCP/UDP, aléatoires. Ainsi, les chances que le switch cible envoie un PACKET_IN au contrôleur sont élevées. D'une part on consomme ainsi des ressources en envoyant beaucoup de PACKET_IN, et d'autre part si les décisions prises par le contrôleur sont trop spécifiques (peu de jokers utilisés par exemple), le switch sur lequel seront appliquées les règles va progressivement se retrouver surchargé de règles inutiles.

Détails techniques :
Se référer au scénario 3 décrit dans l'annexe (page).

Résultat :
L'attaque est réussie : dans les conditions théoriques testées, on passe d'une bande passante de 6,7 Gb/s entre 2 hôtes, à une bande passante de quelques Mb/s. Parfois le déni de service est moins élevé et fournit des variations de débits importantes. Cela s'explique notamment par le fait que certaines règles souvent utilisées restent en haute priorité sur le switch malgré les tentatives de surcharge des tables de flux (et donc sont utilisées sur le plan de données avec des performances acceptables malgré les règles poubelles ajoutées).

Parades proposées :
Les parades pour cette attaque sont assez nombreuses et relativement faciles à mettre en œuvre. Tout d'abord disposer au niveau du contrôleur d'algorithmes de création de règles capables de rassembler plusieurs règles en une seule (c'est à dire capacité de factoriser des règles avec des jokers). D'autre part une politique de filtrage générale (par exemple DROP des paquets sur certaines IP/pour certains ports ou autre) s'avère très efficace. En résumé, une politique réseau stricte conservant la flexibilité initiale avec des jokers dans les règles ajoutées aux tables de flux.

Limitations/Impact/probabilité :
Pour cette troisième attaque, l'attaquant doit encore avoir un accès proche du réseau (il doit

être connecté à un switch du réseau). De plus, si les switchs sont correctement configurés à la base (admettons que la politique du contrôleur soit une politique "opt-in" et non "opt-out", c'est à dire que par défaut les paquets ne sont pas transmis en PACKET_IN au contrôleur mais jetés, sauf cas choisis par le contrôleur), alors l'attaque ne fonctionne plus. Openflow à partir de sa version 1.3 permet d'ailleurs à l'administrateur de définir les actions à appliquer à des paquets inconnus (auparavant ils étaient envoyés au contrôleur dans tous les cas). Les switchs sont normalement sensés pouvoir gérer un nombre suffisant de flux et de règles (cela est spécifié dans le premier OFPT_FEATURES_REPLY, par exemple avec les switchs virtuels mininet, ce paquet indique le support de 256 tables de flux). Le risque majeur de l'attaque est donc finalement l'écrasement de règles utiles par des règles qui ne le sont pas. Le risque est faible, l'impact moyen.

3.4 Deni de service au niveau de l'interface nord

Prérequis/Hypothèses :
- Un éditeur d'application malveillant

Buts :
Tester le secure mode d'ONOS. Regarder ce qu'il est possible d'effectuer comme action néfaste sur le contrôleur, sur les performances du réseau en général. Modifier la topologie du réseau, effacer les tables de flux.

Déroulement :
Un utilisateur mal intentionné charge une application sur le contrôleur. Cette application contient des instructions de tous les types pour consommer les ressources du contrôleur et modifier son fonctionnement. Par exemple on testera si il est possible de provoquer l'arrêt du contrôleur. On testera également l'import et l'utilisation des fonctions de l'API d'ONOS bas niveau, c'est à dire celles qui sont susceptibles d'être utilisées par le coeur d'ONOS pour avoir des informations sur le réseau environnant. Enfin, on regardera si il est possible de monopoliser en partie certaines ressources du contrôleur (par exemple accès au disque, mais aussi processeur avec des calculs couteux répétés en boucle).

Détails techniques :
Se référer au scénario 4 décrit dans l'annexe (page).

Résultat :
Si le contrôleur n'est pas correctement configuré ou est volontairement permissif, il faut avoir une confiance absolue dans les applications qui tournent sans droits restreints. En effet, sinon il est possible d'effectuer toutes les actions envisageables sur le contrôleur et donc sur le réseau.

Parades proposées :
Le secure mode a été mis en place pour parer ce genre de vulnérabilité, et il est efficace pour cela. C'est une protection cruciale qu'on est en droit d'attendre pour un tel contrôleur. Le secure mode est assez puissant car il offre un niveau de granularité très fin [20]. Si l'administrateur général configure correctement le contrôleur et octroie à chaque fois le minimum de privilèges requis pour les applications dont il ne maitrise pas forcément l'origine, cela minimise le risque.

Limitations/Impact/probabilité :
Cette fois la probabilité d'une telle attaque n'est pas à prendre à la légère. Compte tenu de l'offre des contrôleurs SDN concernant la possibilité d'ajouter facilement des applications au réseau, le risque de rencontrer un utilisateur malveillant désirant nuire au réseau ou seulement disposer de

20. `https://wiki.onosproject.org/display/ONOS/ONOS+Application+Permissions`

plus de ressources qu'allouées est élevé. L'impact d'une telle menace est élevé. Les vulnérabilités au sein du contrôleur même sont les plus dangereuses au sein d'un réseau SDN. C'est donc un point qu'il ne faut à aucun prix négliger lorsqu'on souhaite mettre en place un tel réseau. Encore une fois, si le contrôleur est compromis, tout l'est dans le domaine contrôlé.

3.5 Fuites d'information au niveau de l'interface nord

Prérequis/Hypothèses :
- Un éditeur d'application malveillant

Buts :
Voir quelles informations sensibles il est possible de collecter sur les autres applications tournant sur le contrôleur à partir d'une application malveillante. Avec le secure mode activé ou sans.

Déroulement :
Un utilisateur écrit une application d'apparence quelconque mais cherche à utiliser ce qui est à sa disposition dans l'API d'ONOS pour d'une part obtenir des renseignements sur les applications tournant à côté de notre application malveillante, et d'autre part à modifier son fonctionnement, en altérant ce qu'elle est susceptible de recevoir. Lorsque le secure mode est activé, on vérifie que l'application n'a pas accès à des fonctions critiques, et on regarde quelles informations peuvent toutefois fuiter.

Détails techniques :
Se référer au scénario 5 décrit dans l'annexe (page).

Résultat :
Lorsque le secure mode n'est pas activé, ayant accès au service gérant les applications et aux services internes du contrôleur, on peut donc tout faire sur celles-ci (désactivation, envoi de données falsifiées, ...). Sinon, selon les permissions, on peut effectuer certaines actions qui ont plus ou moins d'impacts. Par exemple avec les droits d'accès en lecture au système de fichier, on peut lire certains bouts de mémoire du contrôleur et accéder à des informations pas forcément dénuées d'intérêt. Avec les droits d'accès aux informations des applications en lecture, on peut lister les applications présentes et obtenir d'autres informations.

Parades proposées :
Si il est bien utilisé, le secure mode est efficace pour empêcher des fuites d'information non désirées. Là encore, la responsabilité de donner des droits corrects incombe à l'administrateur et ne doit pas être négligée. Si les permissions d'une application sont réduites au minimum, celle-ci n'a plus beaucoup de possibilités. Une autre protection envisageable permettant d'isoler chaque application des applications voisines est celle qui a été implémentée dans le contrôleur RoseMary (propriétaire), à savoir une séparation des droits d'accès à la mémoire du contrôleur en fonction de l'application (ainsi, contrairement à ONOS au sein duquel il est possible d'accéder à toute la mémoire utilisée par le contrôleur, RoseMary interdit à une application d'accéder à des pages mémoires dont elle n'est pas à l'origine).

Limitations/Impact/probabilité :

Comme précédemment, le risque est élevé. Même si l'impact est plus faible que dans la situation précédente, si il est possible d'extraire de l'information de "vraies" applications, il est envisageable que cela puisse servir en vue d'une attaque ultérieure cette fois sur les vraies applications à un niveau plus haut (en ciblant par exemple des applications avec un niveau de privilège élevé). Cela demeure toutefois assez complexe à mettre en œuvre.

3.6 Mauvaise configuration au niveau de l'interface nord

Prérequis/Hypothèses :
- Un contrôleur ONOS mal configuré (mot de passe faible pour l'API REST)
- Un utilisateur malveillant qui obtient en conséquence des droits d'utilisation de l'API Rest

Buts :
Tirer parti de la politique de gestion d'accès en mode role-based pour qu'un utilisateur avec des droits suffisants puisse altérer de manière non prévue le fonctionnement du contrôleur ou de certaines applications.

Déroulement :
Un utilisateur mal intentionné utilise des fonctionnalités de l'API Rest d'ONOS pour modifier le plus possible le bon fonctionnement du contrôleur.

Détails techniques :
Se référer au scénario 6 décrit dans l'annexe (page).

Résultat :
Depuis l'API REST il est possible d'avoir un impact conséquent sur toutes les parties du contrôleur (applications, mais aussi éléments du réseau et configuration interne). On peut par exemple choisir le comportement par défaut associé à la réception d'un PACKET_IN depuis l'API (et donc éventuellement court-circuiter la réception du paquet par des applications quelconques si on choisit de tout renvoyer automatiquement en tant que PACKET_OUT [21]), désactiver ou activer une application, supprimer un élément réseau connecté ...

Parades proposées :
Le fait de mélanger des droits utilisateurs en rôle et des droits pour chaque application est relativement embêtant dans la mesure où un utilisateur avec des droits suffisants peut théoriquement agir sur toutes les applications existantes sans distinction. Il faudrait je pense ajouter la possibilité de pouvoir agir uniquement sur certaines applications (créer des groupes d'applications qu'on associe à un droit particulier, de cette manière un utilisateur peut avoir les droits de modifications sur certaines applications et pas sur d'autres). Mais là encore, si l'administrateur configure les permissions de manière correcte et si peu de gens ont un accès à l'API REST d'administration, cela constitue une bonne première défense.

Limitations/Impact/probabilité :
L'impact de cette "attaque" est fort (la modification de certaines options peut entraîner de nombreux DoS potentiels). La probabilité elle, reste faible, car l'utilisateur malveillant doit tout

21. http ://nss.kaist.ac.kr/wp-content/uploads/2016/05/p23-lee.compressed.pdf

de même disposer des droits liés à l'utilisation de l'API ainsi que d'un accès à l'API. Il faut donc veiller à changer les identifiants par défaut sur l'interface nord pour éviter qu'un utilisateur quelconque puisse utiliser cette API et prévoir une politique de gestion de mot de passe robuste à ce niveau.

4 Validation et évaluation

4.1 Résultats de l'étude

6 attaques ont donc été réalisées et s'avèrent fonctionnelles (même si les impacts et risques pour chacune sont assez différents). Dans les différentes conclusions tirées, on trouve principalement deux éléments communs :

- La possibilité de contrer certaines attaques lorsqu'on rajoute de l'intelligence humaine dans le contrôleur (algorithmes factorisant la création de règles, gestion des paquets ARP par le contrôleur ...). Cela nécessite cependant du temps (de développement) et peut s'avérer coûteux au niveau du temps de traitement sur le contrôleur.
- La nécessité de configurer correctement ONOS au niveau de l'interface nord, et d'être conscient de l'implication éventuelle de chaque permission octroyée en terme de potentiel d'action sur le contrôleur. Cela étant primordial pour éviter la prise de contrôle du contrôleur par une entité externe.

Les trois premières attaques permettent de montrer qu'on retrouve les vulnérabilités de réseaux classiques sur un réseau SDN. Les impacts y sont globalement plus élevés mais les contre-mesures plus simple à prendre (avoir une vue globale du réseau permet rapidement de bien estimer l'impact d'une action quelconque, ce qui n'est pas forcément réalisé sur un réseau décentralisé). Les trois dernières sont propres à ONOS mais on retrouve les mêmes problématiques sur tous les contrôleurs SDN. ONOS et OpenDayLight restent à ma connaissance les contrôleurs les plus avancés en matière de sécurité, grâce aux modes additionnels qu'ils proposent (secure mode pour ONOS et AAA (Authentification-Authorization-Accounting) pour OpenDayLight). OpenDayLight implémente même un module anti DoS. En revanche, on trouve un grand nombre de documents qui prouvent la dangerosité liée à l'utilisation de nombreux contrôleurs ne proposant pas au minimum une restriction des possibilités offertes à l'utilisateur externe qui a le droit de rajouter une application.

Les tests effectués sont cependant loin de couvrir l'intégralité des menaces qui existent sur le contrôleur, c'est pourquoi la partie suivante tente de compléter celles-ci.

4.2 Autres considérations

Je ne l'ai découvert que trop tard, mais durant la blackhat 2016, s'est déroulée une présentation sur le sujet du stage [22]. Cette présentation résume les différents points névralgiques d'ONOS

22. https://www.blackhat.com/docs/us-16/materials/us-16-Yoon-Attacking-SDN-Infrastructure-Are-We-Ready-For-The
pdf

et d'OpenDayLight, et les schémas y sont limpides (pour les lecteurs désirant bien se figurer certaines attaques).

Parmi les principales attaques qui n'ont pas encore été évoquées, on trouve :

- les attaques sur les switchs : si ceux-ci n'implémentent pas correctement le protocole Openflow ou sont faiblement configurés et qu'il est possible d'en prendre le contrôle, on se retrouve dans le cas où on peut plus facilement exécuter les attaques précédentes sur l'interface sud d'homme au milieu et de deni de service.
- les attaques liées à la gestion multi-contrôleurs éventuelle : il existe une possibilité de contrôler un switch depuis plusieurs contrôleurs à la fois (par exemple pour assurer le service si un contrôleur tombe en panne) et également une possibilité d'échange d'informations entre contrôleurs. Or il n'existe pas encore ni de mécanisme standardisé ni de sécurité très élevée pour de tels échanges, et la capacité des switchs à être contrôlés par plusieurs contrôleurs repose sur la notion de contrôleurs maître/esclaves qui peut aboutir à un deni de service si un contrôleur malveillant monopolise le rôle de maître sur un switch (sans parler des vulnérabilités existantes sur les switchs qui permettent même en étant un contrôleur esclave de modifier les tables de flux [23]).
- les attaques sur le plan de communication : comme on l'a déjà dit, sans TLS, pas de confidentialité ni de confiance dans les données qui transitent via Openflow et donc possibilité pour un attaquant situé dans le réseau de prendre le contrôle d'une partie de celui-ci et de créer des messages malicieux dirigés contre le contrôleur. Mais l'activation de TLS avec authentification mutuelle n'est pas évidente à mettre en place (PKI fiable, qui puisse assurer la révocation, ...).
- les attaques sur les environnements de développement et de déploiement : lors de la construction du contrôleur avec maven le code mais aussi d'autres éléments nécessaires peuvent être (et le sont même dans tous les cas réels) obtenus de manière distante sur des dépôts externes. Si la machine utilisée est corrompue (fichiers de configurations modifiés, DNS cache poisoning, ARP spoofing, malware, ...) alors toute l'installation qui en découle sur les contrôleurs peut fournir une opportunité énorme à l'attaquant de contrôler l'ensemble du réseau. C'est une menace très importante en terme d'impact et dont la probabilité n'est pas si faible qu'on pourrait le penser (social engineering, concentration des efforts sur une seule cible).
- les attaques sur les stations de contrôle et d'administration : vu que certains éléments du conteneur OSGi d'ONOS permettent d'obtenir des droits importants sur le réseau, il est, comme sur un réseau classique, crucial de bien protéger les machines utilisées pour l'administration (là encore le social engineering peut être utilisé).

23. Un collègue à Telecom Sudparis a travaillé sur ce point et montré la vulnérabilité sur certains switchs

Pour bien mettre en valeur les spécificités éventuelles d'ONOS, le tableau suivant récapitule impacts et parades des différentes catégories d'attaque qu'on est susceptible de retrouver.

Catégorie d'attaque et spécificité	Impact	Parade
Flux réseaux forgés -non spécifique SDN -non spécifique ONOS	Injection de trafic pouvant conduire à du DoS ou de l'homme au milieu avec des conséquences plus grandes que sur un réseau classique	Programmation intelligente du contrôleur

4.3 Conclusion

conclu

5 Perspectives à l'issue du stage

6 Ressources

7 Annexe

Cette partie montre comment mettre en place tous les éléments pour reproduire les attaques évoquées précédemment. On commencera donc par installer ONOS, mininet, puis on téléchargera certains outils pour reproduire les attaques.

7.1 Installation d'ONOS

Choisir une machine (virtuelle ou non) sur laquelle installer le contrôleur (durant le stage j'ai utilisé une debian serveur avec accès ssh pour y mettre ONOS). Sur la machine, installer java8 si il ne l'est pas encore. Se rendre à l'adresse `https://wiki.onosproject.org/display/ONOS/Downloads`. Télécharger la dernière release (à l'heure actuelle, Hummingbird).

```
$ wget http://downloads.onosproject.org/release/onos-1.7.1.tar.gz
$ tar -xvf onos-1.7.1.tar.gz
$ cd onos-1.7.1
$ cd
```

Créer un

7.2 Installation de Mininet

Il est possible de suivre les instructions à l'adresse `http://mininet.org/download/`. Télécharger l'image qui convient la plus récente sur github (actuellement 2.2.1) : `https://github.com/mininet/mininet/wiki/Mininet-VM-Images`. La machine virtuelle offre un accès ssh avec les identifiants *mininet / mininet*.

Il est également possible d'installer mininet sur une machine virtuelle déjà existante en clonant `git://github.com/mininet/mininet` et en exécutant *mininet/util/install.sh* dans le dossier mininet.

C'est déjà fini.

7.3 Configuration

Pour des raisons de simplicité, il n'est pas nécessaire de modifier trop d'éléments. Par contre, pour que le contrôleur puisse au moins communiquer aux switchs virtuels, il est nécessaire de créer une interface commune entre ONOS et Mininet.