Orchestration et Vérification de Fonctions de Sécurité pour Environnements Intelligents

Nicolas Schnepf, Remi Badonnel, Abdelkader Lahmadi, Stephan Merz
Equipes de recherche Madynes et Veridis
Inria Nancy Grand Est, Université de Lorraine, LORIA, CNRS
Campus scientifique, 54600 Villers-lès-Nancy, France

***Résumé*—La protection des équipements intelligents tels que smartphones devient un point sensible du fait de la popularité de ces plateformes et des contraintes qu'elles induisent. Pour assurer cette protection, nous proposons une stratégie fondée sur la génération et la vérification automatique de chaînes de sécurité déployées en environnements cloud. En particulier, nous proposons d'exploiter les possibilités offertes par la programmabilité des réseaux et leur vérification formelle pour assurer le déploiement automatique de ces chaînes. Nous décrivons également un ensemble de résultats préliminaires obtenus dans ce cadre.**

I. Contexte

L'intérêt croissant pour les équipements intelligents tels que les smartphones et les tablettes a amené leur déploiement à grande échelle et le développement de leurs propres systèmes d'exploitation. Android est actuellement le système d'exploitation le plus utilisé pour ces équipements avec 87.6% des parts du marché des smartphones en 2016 [2]. Le nombre d'applications disponibles augmente également de façon exponentielle, ce qui révèle la popularité de ces plateformes mais introduit aussi de nouveaux problèmes de sécurité du fait des applications malveillantes utilisées comme vecteur d'attaques [12]. Pour protéger les équipements intelligents contre ces menaces, les solutions de sécurité dites pro-actives consistent à analyser le comportement des applications en utilisant des méthodes de sandboxing avant leur déploiement sur le marché. Bien qu'elles soient essentielles, ces méthodes ont également montré leurs limites en termes de détection d'applications malveillantes. Elles doivent en particulier être complétées par des mécanismes de sécurité à l'exécution, tels que des pare-feux, des antivirus et des systèmes de prévention de fuite de données [7]. Cependant, ces mécanismes posent des problèmes de déploiement sur les environnements intelligents parce qu'ils compromettent sérieusement leurs ressources en termes de batterie et de temps de calcul. De plus, les utilisateurs n'ont souvent pas les compétences requises pour installer et configurer de tels logiciels et sont régulièrement tentés de les désactiver pour améliorer l'autonomie de leur appareil.

De tels mécanismes de protection peuvent être virtualisés et exposés en tant que fonctions de sécurité, de sorte à automatiser et externaliser leur déploiement dans des infrastructures cloud. Ces fonctions peuvent ensuite être composées au travers de chaînages tels que présenté dans [9]. La programmabilité offertes par les réseaux SDN, facilite la construction de ces chaînes de sécurité et permet d'automatiser leur déploiement, leur activation et leur configuration en fonction du contexte [8]. Cependant, la multiplication et la complexité de ces chaînes avec leurs fonctions de sécurité pour protéger un ou plusieurs équipements risquent d'introduire des erreurs de configuration. L'un des principaux défis consiste à vérifier automatiquement ces chaînes de fonctions de sécurité de sorte à s'assurer qu'elles garantissent bien les propriétés de sécurité que l'on attend d'elles, nous proposons d'utiliser les méthodes formelles pour assurer ce genre de vérification.

Il existe déjà un certain nombre d'approches, notamment [3], [10], [1], traitant de la vérification de chaînes de services réseaux. Cependant, ces travaux ne prennent pas en compte les besoins de protection des équipements intelligents tel que par exemple la préservation de leurs ressources ou la minimisation des temps de latence pour des chaînes déployées dans une infrastructure cloud. Ce manque pourrait être comblé par la conception et l'implantation d'une procédure de vérification en lien avec les contraintes propres à Android. En particulier, l'optimisation des performances d'une telle procédure demande une attention toute particulière : en effet, les méthodes formelles sont souvent caractérisées par une complexité exponentielle en terme de temps de réponse et de consommation d'espace mémoire, ce qui peut

Nicolas Schnepf est doctorant en première année de thèse au sein des équipes de recherche Madynes/Veridis au LORIA - INRIA Nancy Grand Est. Il est titulaire d'un diplôme d'ingénieur en informatique de Telecom Nancy obtenu en 2016.

être acceptable si on les utilise hors ligne mais pas si nous considérons des scénarios où les chaînes seraient générées et vérifiées dynamiquement.

II. Problématique et objectifs

Cette thèse porte sur la conception, l'implémentation et l'évaluation de nouvelles stratégies pour la génération et la vérification automatique de chaînes de sécurité capable d'assurer dynamiquement la protection d'équipements intelligents, notamment sous Android, et s'articule en trois axes principaux.

Le premier axe de notre travail concernera la conception d'un orchestrateur de fonctions de sécurité capable d'assurer la protection d'équipements intelligents. Cet orchestrateur aura la responsabilité de construire, vérifier et déployer automatiquement ou semi-automatiquement un ensemble de fonctions de sécurité dans une infrastructure cloud. A partir des propriétés et du comportement des applications installées sur les équipements intelligents, il sélectionnera et composera les fonctions de sécurité appropriées, telles que pare-feux et systèmes de prévention de fuite de données. Cette architecture profitera en particulier des possibilités de programmabilité réseau apportées par le paradigme SDN et par le protocole OpenFlow, ainsi que de l'introduction d'un langage de haut niveau pour la spécification de chaînes de fonctions de sécurité. Notre effort sera en particulier centré sur la conception des différents composants d'un tel orchestrateur ainsi que sur leurs interactions dans une infrastructure cloud.

Un second axe de notre travail sera dédié à la vérification des chaînes de sécurité. La complexité de ces chaînes est accrue par le fait qu'elles seront dynamiquement générées et configurées par l'orchestrateur suivant différents paramètres. Ceci est encore amplifié par le fait que plusieurs chaînes peuvent être activées en même temps à cause de différents aspects de la configuration des équipements intelligents, par exemple différentes chaînes pourraient être actives en même temps pour protéger plusieurs catégories d'applications contre certaines attaques, et ce pour de multiples équipements. Plus spécifiquement, nous suggérons de modéliser les fonctions de sécurité de sorte à pouvoir appliquer des méthodes formelles pour valider qu'elles garantissent bien les propriétés de sécurité adéquates. Les deux techniques que nous allons principalement appliquer dans ce contexte sont le SMT solving et le model checking. En SMT solving, à la fois la chaîne de sécurité et les propriétés qu'elle doit garantir sont représentées sous la forme d'un ensemble de propositions logiques. Ce modèle est ensuite reçu en entrée d'un SMT solver qui vérifie si les propriétés sont garanties par la chaîne. En utilisant les méthodes de model checking, le système est représenté sous la forme d'un automate à états finis et les propriétés à garantir sous la forme de formules en logique temporelle. A nouveau, ce modèle est reçu en entrée d'un model checker qui vérifie si les propriétés sont garanties par l'automate : si ce n'est pas le cas, le model checker construit une trace d'exécution qui explique les raisons de l'échec. Un effort particulier sera apporté dans la découverte de méthodes formelles permettant la représentation de chaînes de sécurité ainsi que de leurs propriétés de sorte à permettre leur vérification à la volée.

Un troisième axe sera focalisé sur l'optimisation du déploiement et de la maintenance des chaînes de sécurité. Les fonctions de sécurité peuvent être déployées et exécutées directement sur les équipements intelligents, être externalisées à la frontière du réseau (en considérant les techniques de fogging) ou simplement être déployées dans une infrastructure cloud standard. Ce travail consistera à automatiquement ajuster le déploiement des fonctions de sécurité composant une chaîne donnée en déterminant l'emplacement où elles doivent être exécutées. L'objectif est de garantir le niveau de sécurité le plus approprié, déterminé par l'orchestrateur, tout en minimisant le coût du déploiement des chaînes. Plusieurs stratégies seront envisagées afin d'améliorer la factorisation des fonctions de sécurité dans ce sens ou en vue d'améliorer la robustesse des chaînes. Une approche possible consisterait à représenter le problème sous la forme d'un problème d'optimisation de flux dans un graphe correspondant aux chaînes de fonctions de sécurité. En utilisant ce type de modèle, il est possible de déterminer l'ensemble des fonctions de sécurité qui peuvent être déployées directement sur l'équipement intelligent sans pour autant compromettre l'autonomie de sa batterie, et celles qui doivent être externalisées au sein d'une infrastructure cloud. Une telle approche pourrait être implémentée en intégrant des logiciels de traitement arithmétiques supportant des algorithmes de simplex. Les performances des stratégies proposées seront quantifiées de sorte à évaluer si elles sont compatibles avec une reconfiguration dynamique des chaînes de sécurité.

III. Résultats préliminaires

Au cours de nos travaux préliminaires, nous avons d'ores et déjà exploré une première stratégie de vérification formelle de chaînes de fonctions de sécurité. Ce travail s'appuie sur le langage pyretic qui fait lui-même partie de la famille des langages frenetic dont le but est d'assurer la programmation des contrôleurs SDN. Plus précisément, nous utilisons l'extension kinetic [11] du langage pyretic, dans laquelle une politique du plan de

contrôle peut être traduite sous la forme d'un automate à états finis pouvant être vérifié par application de model checking. Notre stratégie complète cette approche en y ajoutant la validation de propriétés du plan de données telles que l'autorisation ou l'exclusion de paquets. Notre prototype, appelé synaptic, a été développé comme une nouvelle extension du langage pyretic. Il exploite ce langage intuitif pour spécifier des chaînes, et l'étend en y ajoutant un formalisme permettant d'exprimer les propriétés qui doivent être garanties. Pour intégrer des méthodes formelles dans synaptic, nous avons défini des règles de réécriture qui convertissent les chaînes définies avec le langage pyretic dans un formalisme pouvant être vérifié par application de SMT solving ou de model checking. Durant nos expérimentations, nous avons utilisé les solveurs SMT CVC4 [4] et veriT [5] et le model checker nuXmv [6].

Pour évaluer les performances de ces différentes méthodes en terme de temps de réponse et de consommation mémoire, nous avons défini plusieurs critères caractérisant la taille de la chaîne reçue en entrée. En particulier, nous avons évalué l'impact du nombre de configurations possibles pour une même chaîne, la taille d'une chaîne en termes de longueur et de largeur, c'est-à-dire respectivement le nombre de règles composées en séquence et en parallèle, et enfin le nombre de propriétés à vérifier avec la chaîne. Pour chacun de ces paramètres, nous avons implanté un module python permettant la génération synthétique des différentes chaînes de sécurité : ces résultats sont reportés dans le tableau I où la seconde colonne indique la valeur maximale de chaque paramètre et où les colonnes suivantes donnent le temps de réponse des différents algorithmes de vérification.

Critère	Max	CVC4	veriT	nuXmv
Nombre de config.	110	1.27	0.84	3.82
Largeur de la chaîne	1000	6.44	15.92	0.75
Longueur de la chaîne	100	103.56	0.26	0.13

TABLE I
COMPARAISON DES RÉSULTATS EXPÉRIMENTAUX OBTENUS AVEC DIFFÉRENTS OUTILS DE VÉRIFICATION (CVC4, VERIT ET NUXMV)

Ces résultats indiquent clairement que nuXmv fournit de meilleurs résultats que veriT et CVC4 pour des chaînes importantes en longueur et largeur, même si la différence est moins significative avec un nombre de configurations important. Les résultats obtenus lorsque nous avons évalué l'influence du nombre de propriétés, non présentés ici, confirment que nuXmv apporte de meilleurs résultats que CVC4 et veriT en terme de temps de réponse. De manière générale, nuXmv semble être le meilleur candidat pour la vérification de chaînes de fonctions de sécurité à la volée, au cours de ces expérimentations. Néanmoins, notre stratégie requiert encore d'être optimisée dans le cas où il faudrait vérifier des chaînes caractérisées par un nombre élevé de configurations.

IV. CONCLUSIONS

Dans ce document, nous avons présenté les grandes lignes de notre travail autour de l'orchestration et de la vérification de chaînes de fonctions de sécurité pour la protection des environnements intelligents : Le premier axe de ces travaux porte sur la conception d'un orchestrateur chargé de composer, configurer et déployer les chaînes de sécurité ; notre second axe traite l'intégration de méthodes formelles pour permettre la validation automatique de ces chaînes ; enfin nous avons abordé la mise en oeuvre de techniques d'optimisation pour assurer leur déploiement et leur reconfiguration à l'exécution.

Au cours de nos premières expérimentations nous avons comparé les performances apportées par différentes méthodes formelles pour la validation de chaînes de fonctions de sécurité à l'exécution : au sortir de cette phase d'évaluation c'est le model checker nuXmv qui est apparu comme le meilleur candidat pour l'implantation d'un module de vérification à l'exécution ; néanmoins cette approche demande encore à être améliorée au vu des performances qu'elle apporte dans le cas de chaînes de fonctions de sécurité comportant de nombreux états de configuration. Parmi nos travaux futurs, nous pouvons notamment mentionner la sélection dynamique des chaînes de sécurité à déployer en fonction des applications installées sur les équipements, ainsi que l'exploitation de nouvelles méthodes formelles pour améliorer notre procédure de vérification.

RÉFÉRENCES

[1] Ehab Al-Shaer and Saeed Al-Haj. Flowchecker, configuration analysis and verification of federated OpenFlow infrastructures. In *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration (CCS 2010)*, 2010.

[2] IDC, Smartphone OS Market Shares 2015-2016. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[3] Thomas Ball and Nikolaj Bjorner. Vericon : towards verifying controller programs in software defined networks. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design (PLDI 2014)*, 2014.

[4] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proc. 23rd Intl. Conf. Computer Aided Verification (CAV 2011)*, USA, 2011.

[5] Thomas Bouton, Diego Caminha Barbosa De Oliveira, David Déharbe, and Pascal Fontaine. veriT : An Open, Trustable and Efficient SMT-Solver. In *Proc. of the 22nd Int. Conference on Automated Deduction (CADE-22)*, Montreal, Canada, 2009.

[6] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv symbolic model checker. In *Proc. 26th Intl. Conf. Computer Aided Verification (CAV 2014)*, Vienna, Austria, 2014.

[7] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan. Android Security, a Survey of Issues, Malware Penetrations and Defenses. In *IEEE Communications Surveys & Tutorials*, 2015.

[8] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The Road to SDN : An Intellectual History of Programmable Networks. *SIGCOMM Comput. Commun. Rev.*, 44(2) :87–98, April 2014.

[9] Gaëtan Hurel, Rémi Badonnel, Abdelkader Lahmadi, and Olivier Festor. Behavioral and Dynamic Security Functions Chaining for Android Devices. In *Proc. of the 11th International Conference on Network and Service Management (CNSM 2016)*, 2016.

[10] Ahmed Khurshid, Xuan Zou, Winxuan Zhou, Matthew Caesar, and P. Brighten. VeriFlow : verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks (HOTSDN 2012)*, 2012.

[11] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhamad Shahbaz, Nick Feamster, and Russ Clark. Kinetic : Verifiable Dynamic Network Control. In *Proc. of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI 2015)*, May 2015.

[12] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A Survey on Security for Mobile Devices. In *IEEE communications surveys & tutorials*, 2012.