Towards Cloud-Based Compositions of Security Functions For Mobile Devices

Gaëtan Hurel, Rémi Badonnel, Abdelkader Lahmadi and Olivier Festor

INRIA Nancy Grand-Est - LORIA, France
Email: {hurel, badonnel, lahmadi, festor}@inria.fr

***Abstract*—In order to prevent attacks against smartphones and tablets, dedicated security applications are usually deployed on the mobile devices themselves. However, these applications may have a significant impact on the device resources, and users may be tempted to uninstall or disable them. In this paper, we propose a new approach to outsource mobile security functions and build transparent in-path security compositions for mobile devices. The functions are dynamically activated, configured and composed using software-defined networking and virtualization capabilities. We present a mathematical formalization to model the security compositions, and describe the functional architecture. We provide an implementation prototype and evaluate the solution through an extensive set of experiments.**

I. Introduction

The development of high-speed mobile networks has led to the large-scale deployment of mobile devices - such as smartphones and tablets [1] - offering multiple services and applications for end-users. These devices are typically used for personal or professional reasons, and companies are progressively moving from traditional user-supplied devices to BYOD[1]-related strategies. Like most of popular and widespread technologies, mobile devices are also an attractive target for malicious attackers. This trend can be explained by several reasons. For instance, these systems suffer from a larger attack surface than traditional computers due to their strong connectivity, and they are likely to store private and sensitive data about their respective owner. Consequently, the number and the types of malwares for mobile devices have dramatically increased these last years [7].

Most of mobile security solutions are available in the form of applications to be directly installed on the devices themselves. Such on-device approaches offer some advantages but induce generally significant resources consumption on the system, leading to the reduction of the battery lifetime. In the meantime, current cloud-based solutions try to offload the most of the workload on remote servers, while only requiring lightweight agents to be installed on the systems. Such solutions reduce the amount of used resources on the devices, but at least two major problems remains. Firstly, average users do not have the required knowledge to properly perform security decisions in case of settings or alerts for instance. Secondly, such solutions lack of flexibility and contextualization regarding the device's state to know how (e.g. which protections for which applications) and when (e.g. public or private networks) to use them.

In this paper, we propose a new strategy for outsourcing mobile security functions as cloud-based services for smartphones and tablets. The outsourced functions are dynamically activated, configured and composed using software-defined networking and virtualization capabilities. We consider the use of security compositions in order to dynamically fit the security requirements of mobile devices according to their current contexts and risks. This mechanism is performed in a transparent manner from an end-user point of view. We also investigate the different traversal schemes that can be exploited to drive the behavior of the compositions. We evaluate the benefits and drawbacks of our strategy through an intensive set of experimentations. Our main contributions are (i) a network architecture able to dynamically deliver security for mobile devices by building and deploying security compositions (ii) a mathematical model to formalize security compositions and express their properties regarding cost, quality and scalability, and (iii) a first implementation prototype of the network architecture and a series of experiment based on it.

The rest of this paper is structured as follows. Related work regarding cloud-based mobile security is discussed in Section II. Section III presents a mathematical formalization to model the security function compositions within our solution. Section IV explains our strategy for delivering composable and dynamic security for mobile devices, as transparent services in the cloud. Prototyping and evaluation of our solution are described in Section V. Finally, Section VII presents conclusions and points out some future research perspectives.

II. Related Work

Mobile devices security is a critical activity. In this section, we describe the related work regarding cloud-based security for mobile devices, as well as the possible mechanisms for outsourcing and composing security middleboxes to this end.

Mobile security

In addition to traditional on-device approaches [16], several cloud-based approaches have been proposed in order to provide security for mobile devices. Some solutions exploit cloning methods using virtualization to execute security checks in the cloud without resources constraints [18][24]. Other work directly outsource security functions of the mobile devices as remote services. For instance, a cloud-based applications firewall and a cloud-based antivirus for mobile devices are respectively introduced in [15] and [17]. Though these work give some strong contributions regarding the security of the devices, they still induce additional network communications that may be prohibitive from an end-user point of view.

[1]Bring Your Own Device

In that context, the use of software-defined networking for transparently delivering mobile security has been studied. An Openflow [5] appliance able to detect mobile malwares using traffic analysis is presented in [14]. The traffic is gathered by the OpenFlow wireless access point and then analyzed by the network controller. Overall, our work aims at differ from those previous ones in the sense that it does not focus on some specific threats in a fixed manner. Instead, we want to dynamically select and compose security functions according to the current threats to be mitigated.

Middleboxes outsourcing

Some early works regarding middleboxes outsourcing have been inspired by the limited control available to customers over the cloud network infrastructures. For instance, CloudNaaS [9] is an OpenFlow-based networking framework allowing customers to outsource line-of-business applications along with network functionalities - i.e. middleboxes - in the cloud. OpenADN [22] is a work in the same vein except for the main facts that it is able to deal with inter-cloud integration, dynamic resources scaling and application-level flow processing. In the meantime, [21] and [12] explore some new designs to dynamically and transparently outsource middleboxes across several cloud providers. Such work leverages NFV-like virtualization and different (e.g. Openflow-based, DNS-based) redirection mechanisms. Our work is similar to [21] since we outsource middleboxes in the cloud while being totally agnostic with respect to the location of the remote services. We perform traffic redirection using OpenFlow such as [12]. It is worth mentioning that our solution is designed to integrate both hardware standalone middleboxes as well as consolidated middleboxes on shared hardware resources in the vein of [20].

Service chaining

Several recent works have leveraged the software-defined networking paradigm in order to manage and compose middleboxes for building service chains. Among them, SIMPLE [19] is a SDN-based policy enforcement layer for middlebox-specific traffic steering. In SIMPLE, middlebox composition and routing policies are enforced by pushing the corresponding flow rules into the OpenFlow network. Flowtags [10] has similar goals but uses another approach, putting forward an architecture where middleboxes are extended to support OpenFlow and use tags in network packets for determining how to process them. Stratos [11] is another orchestration layer for virtualized middleboxes and differs from the aforementioned ones by its fully virtualized nature. Alternatively, Slick [8] is an SDN-based architecture where the data plane can be extended with middleboxes implemented on programmable resources such as FPGA. Though middlebox composition is not studied, such solution should allow to configure the extended data plane to this end. Currently, our work is close to [19] since we use flow rules to build middlebox chains.

III. Problem Statement

Our objective is to define a strategy for efficiently outsourcing and composing security functions for mobile devices according to their context and risks. First of all, we present a mathematical representation to formalize the security compositions within our approach. We then describe how this model

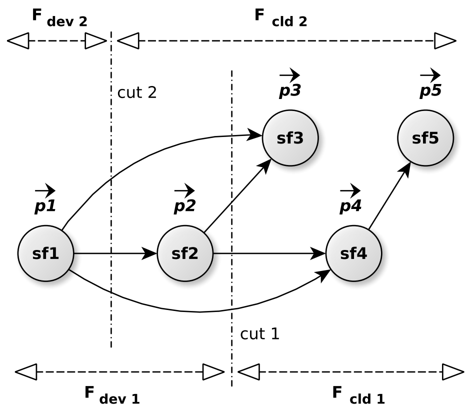


Fig. 1: An example of composition graph. The dotted line represents a cut between F_{dev} and F_{cld} as discussed in section III-C. This separation can dynamically change over time, according to the device's resources for instance.

can be used for a given composition in order to determine its cost, its quality and its scalability. Finally, we explain how to exploit this modeling to dynamically provide a trade-off between *on-device* and *in-cloud* security - that is, for a given composition, the part of security functions to be deployed on the mobile devices and the part to be deployed in the cloud.

A. Security compositions

Each composition C can be expressed as a directed acyclic graph $C = (F, T)$ where $F = \{sf_1, sf_2, ..., sf_n\}$ is the set of security functions that are part of the composition C (i.e. the vertices), and $T = \{t_1, t_2, ..., t_m\}$ the set of control flow transmissions among the security functions (i.e. the edges). An example of composition graph including a set of five security functions is given by Figure 1. Several properties must be enforced for such a graph:

- **Edge-points**: a composition graph includes at least one entry point and one exit point - we call such points *edge-points*. A single flow to be analyzed can enter and exit the composition through only one entry point and one exit point. This is mainly due to synchronization purposes and duplicated traffic avoidance. On the composition example, sf_1 acts as the only entry point while sf_3 and sf_5 are the two possible exit points of the composition.
- **Connectivity and directivity**: there must exist at least one directed path from each function within the composition - including the entry point - towards any exit point of that composition. This ensures that a valid traffic will always reach its final destination after all the security treatments have been performed within the composition.
- **Weighting**: some weights can be applied on the security functions that are part of a given composition as explained in the next subsection. Depending on the number of chosen metrics, these weights can be single scalar values as well as n-ary vectors.

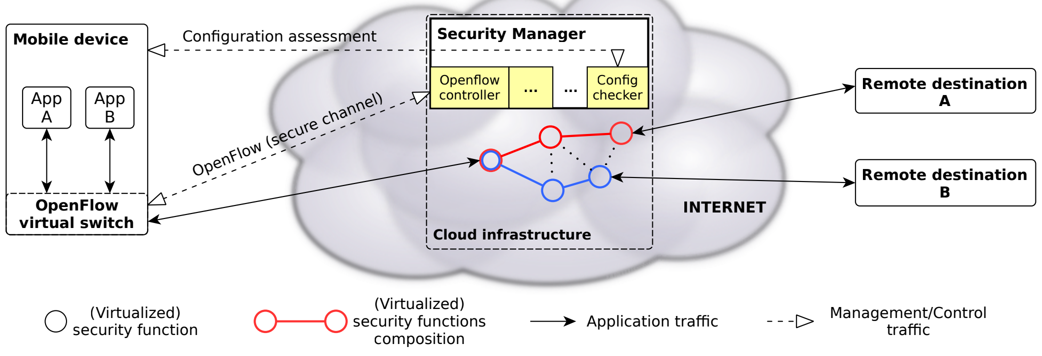


Fig. 2: A cloud-based composable security architecture for mobile devices.

B. Composition characterization

Each composition involves a set of security functions to be deployed either on the device or in the cloud. We explain now how to determine the cost, the quality and the scalability of a given composition according to its set of security functions.

1) Cost of a composition: The cost of a composition defines the amount of used resources on the mobile device when it employs the given composition for analyzing its traffic. This cost depends on the ratio between the number of security functions deployed on the device and the number of the ones deployed in the cloud. We use the CPU usage as the basic metric to estimate the cost on the mobile device. Thus, each security function sf_i of a given composition graph C is weighted by a weight $p(i) = p_i$ expressing the amount of CPU usage it would require if it would be executed on the device. Equation 1 characterizes the overall usage $P(C)$ of a given composition $C = (F, T)$ with $n = |F|$, assuming (i) all the security functions within C are run on the device, and (ii) all the security functions have the same execution duration. The overall usage consists in the average of the different CPU usages induced by the security functions within C.

$$P(C) = \frac{\sum_{i=1}^{n} p(sf_i)}{card(F)} \text{ with } sf_i \in F \tag{1}$$

2) Quality of a composition: The quality of a composition is likely to depend on numerous factors, such as the end-to-end delay and the security effectiveness. We mainly focus on the end-to-end delay in this work. In our context, we define the end-to-end delay as the sum of (i) the transmission delay overhead due to the redirection in the cloud, and (ii) the treatment delays induced by the different security functions on the traffic to be analyzed. As a result of the high-performance networks usually provided within current cloud infrastructures, we assume the transmission delays between the security functions in the cloud are negligible. In contrast, the treatment delay represents the required time for a security function to perform all its security tasks on a given flow. Thus, each security function sf_i of a composition graph C is weighted by a weight $d(i) = d_i$ expressing the treatment delay it would induce if it would be run on the device. Equation 2 characterizes the overall end-to-end delay $D(C)$ of a given composition $C = (F, T)$ with $n = |F|$, assuming (i) all the security functions of C are run on the device, and (ii) all the security functions are run sequentially. This overall delay consists in the sum of the redirection overhead δ and the sum of the treatment delays induced by the functions of C.

$$D(C) = \sum_{i=1}^{n} d(sf_i) + \delta \text{ with } sf_i \in F \tag{2}$$

3) Scalability of a composition: The scalability of a composition is its ability to correctly handle multiple network flows (e.g. multiple applications, multiple devices) at the same time. We are mainly interested in measuring the scalability of a composition on the device-side in function of (i) the CPU usage and (ii) the end-to-end delay. Thus, for a given composition, the only difference between the two previous points is that each weight on the different nodes has to be multiplied by the number of different flows. Equations 3a and 3b characterize the scalability of a given composition C respectively in terms of CPU usage and end-to-end delay, for a given number K of network flows to be analyzed.

$$S_{CPU}(C) = K.P(C) \tag{3a}$$
$$S_{E2E}(C) = K.D(C), \tag{3b}$$

C. Finding the right deployment of security functions

For a given composition $C = (F, T)$, the set F can be divided into two subsets F_{dev} and F_{cld}, respectively containing the security functions to be deployed on the device and the ones to be deployed in the cloud. We call such a separation a *cut* for the composition. Determining the right cut is a critical step to provide the right deployment of security functions according to end-user requirements. For example, one may prefer to reduce the overall delay when possible by automatically running all the security functions of some given short compositions on the device only. Another one may just want to preserve the battery life of his device and to outsource the entire compositions in the cloud.

1) Constraints on the cut: In order to ensure the completeness of the cut and to avoid duplicated function executions at the same time, F_{dev} and F_{cld} must be defined in such a way that $F_{dev} \cup F_{cld} = F$ and $F_{dev} \cap F_{cld} = \{\}$. Going back to Figure 1, possible values for F_{dev} could be for example $F_{dev} = \{sf_1, sf_2\}$ or $F_{dev} = \{sf_1\}$ as respectively represented by the dotted line *cut1* and *cut2*. Several graph-related constraints must also be respected by an efficient cut:

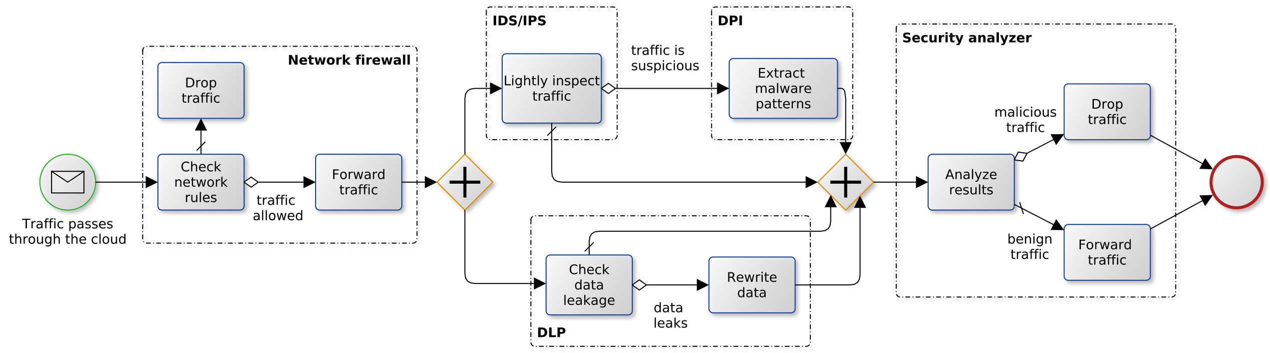


Fig. 3: An example of composition for the policy "Block malicious outgoing traffic and prevent data leakage".

- **Round trip minimization:** in order to avoid additional control flow round trips when analyzing a single flow, there must be only one control flow transmission between F_{dev} and F_{cld}, which would arise only when all the required security functions of a first subset would have been executed; note that depending on the traffic direction, the subset order can change.
- **Edge-points partitioning:** for consistency purposes and duplicated traffic avoidance, the entry points and exit points of a given composition must not be included in a same subset.

2) Distributing functions: Choosing the right subset for each security functions of the composition must be balanced between - *at least* - cost and quality requirements. A possible way of reaching those trade-offs can be to find optimal cuts on the composition graph according to the different metrics discussed in the previous subsection and respecting the constraints introduced above. For instance, one simple cut could define F_{dev} such as the overall CPU usage cost of F_{dev} - similar to Equation 1 since F_{dev} can be seen as a sub-composition - would be less than a given threshold. In that case, the weights associated to each functions within the composition would be scalar values, since only one metric (CPU usage) is taken into account. Conversely, these weights would be vector values if the chosen metrics were both CPU usage and end-to-end delay.

IV. Cloud-based Composable Security Strategy for Mobile Devices

We propose a new strategy for delivering composable and dynamic security functions for mobile devices, as a transparent service in the cloud. In comparison to traditional on-device models, security is no more performed through a relatively static heap of functions which are executed on mobile devices *only*. Instead, it is *mainly* based on a set of security functions that may be hosted in the cloud and dynamically composed depending on the current context and risk level. By doing so, our strategy aims at addressing different constraints in terms of resource consumption, dynamicity, and maintenance. We first describe the architecture supporting our security strategy. We then present how such an architecture can be used to implement security policies for mobile devices.

A. Architecture

Our proposed architecture [13] is depicted by Figure 2 and involves three distinct entities: (i) on the left, the mobile device with several running applications and an integrated OpenFlow switch for redirection purposes; (ii) in the middle, a cloud provider infrastructure hosting the outsourced mobile security functions as well as a security manager and some additional modules including an OpenFlow controller; (iii) on the right, the remote destinations interacting with the mobile device applications. When an application wants to communicate with a remote destination, all the messages from and to that application pass through the virtual switch of the device. At the beginning of the communication, this switch may probe the OpenFlow controller from the cloud provider in order to know how to redirect the related messages for applying security treatments on them. Depending on the risks and context, the manager activates the appropriate security functions and designs a dedicated security composition. By pushing the necessary OpenFlow rules within the cloud provider network, the controller then chains these security functions to finalize the given composition building and notifies the switch. This one finally makes the chosen incoming and outgoing traffic pass through the composition before reaching the final destination. Therefore, most of the security checks may be applied *in the cloud* instead of *on the devices*. Security compositions are designed and/or chosen by the manager according to several factors such as the originating application, the remote destination and the network properties. For example, a mobile application requiring access to the enterprise intranet would need to use a security composition including at least an anti-malware and a data leakage prevention mechanism. The features are not limited to traffic analysis - the security manager can also host additional security functions such as a configuration checker capable of controlling the proper configuration of the mobile devices. Using our approach, most of the security intelligence will be moved at the security manager level, potentially minimizing user involvement.

B. Policies and scenarios

Security compositions built within our proposed architecture can be used to define and enforce security policies. Security functions included in such composition can be run sequentially, conditionally or concurrently, thus characterizing the composition traversal. We describe the different categories of traversals and show an example of policy implementation.

1) Composition traversal: Within our approach, the control flow transmission when a security function sf_i of a composition $C = (F, T)$ has terminated can use several schemes:

- **Sequential scheme:** the transmission is sequential if the control flow always goes to the function sf_j after sf_i has finished. In the composition graph C, sf_i has only one outgoing edge, which goes towards sf_j.
- **Conditional scheme:** the transmission is conditional if the control flow goes from sf_i to sf_j after sf_i has finished *and* a specific condition is respected. In the composition graph C, sf_i has as many outgoing edges as the number of potential conditions N_C, and these edges respectively go towards sf_j,sf_k ... sf_{i+N_C}
- **Concurrent scheme:** the transmission is concurrent if the control flow goes simultaneously to multiple function sf_j...sf_n after sf_i has finished. In the composition graph C, sf_i has as many outgoing edges as the number of next functions N_F, and these edges respectively go towards sf_j,sf_k ... sf_{i+N_F}.

2) Scenario example: Let consider the security policy ”Block malicious outgoing traffic and prevent data leakage”. The Figure 3 describes the composition associated to this policy using a typical composition language, called BPMN[2] language. When the mobile device outgoing traffic enters the composition, it is first checked against several firewall rules. If the traffic is allowed, it is then simultaneously inspected by a lightweight intrusion detection system (IDS) and a data leakage prevention engine (DLP). If the IDS flags the traffic as suspicious, a Deep Packet Inspection (DPI) is then used in order to extract all malware-related data within the given traffic. If the traffic is not suspicious, the next security function after the IDS is the security analyzer. In contrast, the next security function after the DLP is always the security analyzer. This latter is actually in charge of synchronizing and correlating the security information revealed by the IDS, the DLP, and potentially the DPI. At the end of this synchronization and correlation step, the security analyzer is able of determining whether or not the traffic is safe and can be forwarded to the remote destination. The corresponding graph representation of that composition is given by Figure 4, where sf_1, sf_2, sf_3, sf_4 and sf_5 respectively represent the firewall, the DLP, the IDS, the DPI and the security analyzer. Control flow starts at the firewall (entry point) and finishes at the security analyzer (exit point). Note that all composition traversal schemes are exploited in this composition: concurrent traversal right after the network firewall (red edges on the picture), conditional traversal right after the IDS (blue edges), and sequential traversal right after the DLP and the DPI (black edges).

V. Prototyping & Evaluation

In this section, we detail the experimental testbed we setup to prototype our strategy. We then relate the experiments we conduced to evaluate the performances of the security compositions, in terms of resource cost and end-to-end delay.

A. Experimental setup

The prototype we developed is composed of three distinct parts in link with the functional architecture, namely (1) the mobile device to be protected, which runs an OpenFlow client, (2) the cloud infrastructure hosting mobile security functions,

[2]Business Process Model and Notation

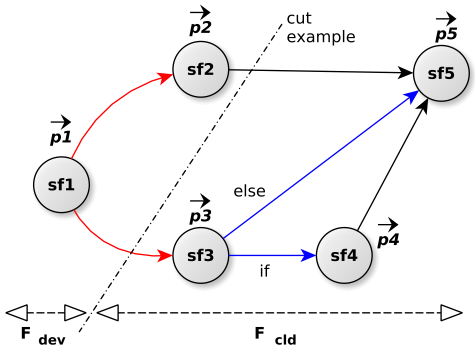


Fig. 4: Composition graph for a security policy ”Block malicious outgoing traffic and prevent data leakage”.

and (3) the remote destinations interacting with the mobile device. Along this prototype, our setup includes two different types of security functions and a small Android application to generate network traffic.

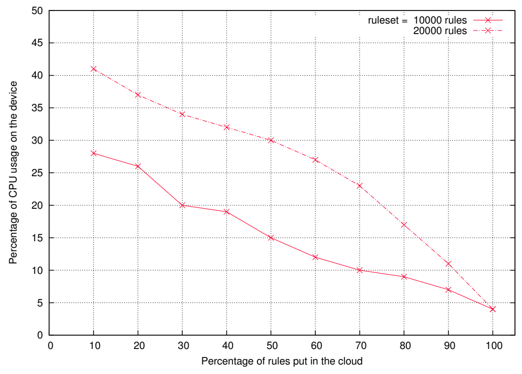
1) Mobile device: We used a Samsung Galaxy S3 device with a custom CyanogenMod ROM (10.2.1 intl) running Android Jelly Bean (4.3.1). The OpenFlow client embedded in the device was an Open vSwitch (OVS) version 2.1.0, which runs in the kernel space. Our device setup is strongly inspired from [23], the main differences being that (1) we do not need to setup a virtual ethernet interface, and (2) we do not embed the control plane (i.e. the OpenFlow controller) on the device.

2) Cloud infrastructure: We used the Mininet emulator [2] in order to build and configure the cloud infrastructure. We emulated an OpenFlow-based network containing OVS switches (version 1.10.0) to forward the traffic and standard Linux hosts to host security functions. The OpenFlow controller was a dedicated controller running within the Mininet virtual machine. We chose the POX controller for our experiments. Thus, the traffic redirection and forwarding logic required within our architecture was implemented as a simple POX module written in Python.

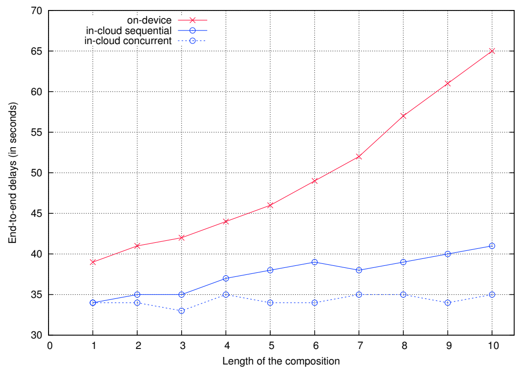
3) Security functions: We chose the iptables/netfilter [3] firewall and the Suricata IDS/IPS [6] for our experiments. While the former is a built-in future in both Android (device) and Ubuntu (emulated hosts), we had to cross-compile Suricata for Android (without NFQUEUE support, thus only in IDS mode). For both of these security functions, we developed custom scripts to automatically generate a large number of rules. Those scripts were leveraging a large list of malicious IP addresses taken from a public database to generate corresponding rules in order to block traffic to/from these IP addresses.

4) Remote destinations: Due to some Mininet-related limitations, we chose to emulate additional Linux host within Mininet to act as the remote destinations (e.g. HTTP server).

5) Workload: We needed to generate a significant amount of network traffic from the smartphone to evaluate the performances of the security functions. To this end, we developed a small Android application able of sending a given amount of HTTP requests - and receive the associated HTTP responses - to a web server at a rate of one request each 500 ms. The total number of HTTP requests to be sent and the destination web server were dynamically specified according to the experiment.



(a) Maximum CPU usage induced on the device by the IDS function according to the ratio of rules it carries.



(b) Average end-to-end delay induced by firewall compositions up to ten firewalls (each with 1000 rules).

B. Experimental results

We have performed an extensive set of experiments to evaluate the performances of our solution. We mainly focus on the cost and the quality for a given composition on the device side. For each experiment, the compositions were proactively built and theirs security functions were placed as *inline* middleboxes - that is, directly between two OpenFlow switches.

1) Maximum CPU usage and security cover: Our first set of experiment focuses on the resources cost induced on the device by an IDS function for which the set of rules is shared across the cloud and the device. We divide the experiment phase in rounds according to the number of rules the device carries. For each round, the same amount of traffic is generated by the mobile device. We monitor the maximum CPU usage induced by the IDS on the device using adb. Results on Figure 5a show that the maximum CPU usage reduces linearly on the device as the number of IDS rules carried in the cloud increases. For instance, the maximum CPU usage induced by the IDS on the device is about 28% when its carries 9000 rules, and 7% when it carries 1000 rules. We also note that when no rule at all is put on the device, the IDS still induce a CPU usage of 4% - for comparison, the OpenFlow client induce an average CPU usage of 1-2%. In addition, we observe that the maximum CPU usage remains almost the same on the device whether it carries 80% of 10000 rules or 40% of 20000 rules for example. We therefore conclude that, for a same amount CPU usage on the device, security cover is better when the most of firewall rules is outsourced in the cloud.

2) Average end-to-end delay: Our second set of experiment deals with the average end-to-end delays induced when using firewall compositions of variable length, both in the cloud and on the device. Each firewall within the compositions contains a set of 1000 rules. Cloud-based firewall compositions are distributed across several hosts and either use sequential or concurrent traversal. For each round, the same amount of traffic is generated by the mobile device. As stated in III-B2, we consider that transmission delays are negligible within our testbed, and only treatments delays are therefore likely to influence the end-to-end delays. Results on Figure 5b show that average end-to-end delays induced by sequential compositions grow significantly on the device according to the number of firewall rules it carries. This growth is also slightly visible for cloud-based sequential compositions, yet it offers much lower latency compared to on-device compositions. Cloud-based concurrent compositions show the best results since security functions can adequately scale-out. Such performances may however be balanced with the overhead induced by an eventual synchronization step required after the concurrent processing. We conclude that average end-to-end delays induced by security compositions can be significantly reduced when most of the compositions functions are ran in the cloud, both in sequential and concurrent ways.

VI. Conclusions

We have presented a novel solution for outsourcing security functions in the cloud, and dynamically deploying security compositions for protecting mobile devices. These compositions are built by chaining the security functions using OpenFlow rules pushed into the cloud network. Using our approach, compositions are transparently deployed between the mobile devices and the remote destinations they are interacting with, thus providing the basis for an efficient and transparent network-based security. We built a mathematical model that permits to formalize the security compositions and to define their different properties, with respect to their cost, quality and scalability. We designed an OpenFlow-based network architecture supporting our approach, and developed a first implementation prototype serving as a base for performance evaluation. Experimental results show the benefits of our strategy on the device side: the maximum CPU usage induced by a standard IDS can be reduced up to 20-25% according to the number of IDS rules that are outsourced in the compositions; similarly, firewall treatment delays are shorter when most of their rules are outsourced from the devices. For a given amount of CPU usage on the devices, we also show that the security coverage may be significantly improved when the most of the security processing is done in the compositions.

For future work, we plan to explore to what extent we can automate our solution, from the security policy specification to the deployment of associated security compositions. To this end, we are interested in analyzing different machine-learning algorithms on a large dataset of mobile communications. In the meantime, redirection mechanisms are a key part of our solution, and we want to study possible alternatives to OpenFlow (e.g. NETCONF [4]). Finally, we plan to explore how the security functions within a same composition can interact each other and share some security information, thus allowing a consistent processing of the network traffic.

ACKNOWLEDGMENTS

This work was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Program, and by the AKD STIC-AmSud Project.

REFERENCES

[1] More smartphones were shipped in Q1 2013 than feature phones, an industry first according to IDC. http://www.idc.com/getdoc.jsp?containerId=prUS24085413. Last visited in august 2014.

[2] The Mininet emulator. http://mininet.org/. Last visited in september 2014.

[3] The Netfilter firewall. http://www.netfilter.org/. Last visited in september 2014.

[4] The Network Configuration Protocol (NETCONF). http://tools.ietf.org/html/rfc6241. Last visited in september 2014.

[5] The OpenFlow specifications . https://www.opennetworking.org/sdn-resources/onf-specifications/openflow. Last visited in september 2014.

[6] The Suricata IDS/IPS. http://suricata-ids.org/. Last visited in september 2014.

[7] Mobile threats report from Juniper. http://www.juniper.net/us/en/forms/mobile-threats-report/, 2013. Last visited in august 2014.

[8] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford. A Slick Control Plane for Network Middleboxes. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 147–148, New York, NY, USA, 2013. ACM.

[9] T. Benson, A. Akella, A. Shaikh, and S. Sahu. CloudNaaS: A Cloud Networking Platform for Enterprise Applications. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 8:1–8:13, New York, NY, USA, 2011. ACM.

[10] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. FlowTags: Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 19–24, New York, NY, USA, 2013. ACM.

[11] A. Gember, A. Krishnamurthy, S. St. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. Stratos: A Network-Aware Orchestration Layer for Middleboxes in the Cloud. *CoRR*, abs/1305.0209, 2013.

[12] G. Gibb, H. Zeng, and N. McKeown. Outsourcing Network Functionality. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 73–78, New York, NY, USA, 2012. ACM.

[13] G. Hurel, R. Badonnel, A. Lahmadi, and O. Festor. Outsourcing Mobile Security in the Cloud. In *Monitoring and Securing Virtualized Networks and Services - 8th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2014, Brno, Czech Republic, June 30 - July 3, 2014. Proceedings*, pages 69–73, 2014.

[14] R. Jin and B. Wang. Malware Detection for Mobile Devices Using Software-Defined Networking. In *Proceedings of the 2nd GENI Research and Educational Experiment Workshop (GREE 2013)*, pages 81–88, 2013.

[15] C. Kilinc, T. Booth, and K. Andersson. WallDroid: Cloud Assisted Virtualized Application Specific Firewalls for the Android OS. In *Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2012)*, pages 877–883, 2012.

[16] M. La Polla, F. Martinelli, and D. Sgandurra. A Survey on Security for Mobile Devices. *Communications Surveys Tutorials, IEEE*, 15(1):446–471, First 2013.

[17] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian. Virtualized In-Cloud Security Services for Mobile Devices. In *Proceedings of the 1st Workshop on Virtualization in Mobile Computing (MobiVirt'08)*, page 31–35, 2008.

[18] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: Versatile Protection for Smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC'10)*, page 347–356, 2010.

[19] Z. A. Qazi, C.-C.n Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 27–38, New York, NY, USA, 2013. ACM.

[20] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and Implementation of a Consolidated Middlebox Architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[21] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 13–24. ACM, 2012.

[22] P. Subharti, R. Jain, J. Pan, J. Iyer, and D. Oran. OpenADN: Mobile Apps on Global Clouds Using Software Defined Networking. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, MCS '12, pages 1–2, New York, NY, USA, 2012. ACM.

[23] K-K. Yap, T-Y. Huang, M. Kobayashi, Y. Yiakoumis, N. McKeown, S. Katti, and G. Parulkar. Making Use of All the Networks Around Us: A Case Study in Android. In *Proceedings of the 2012 ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design*, CellNet '12, pages 19–24, New York, NY, USA, 2012. ACM.

[24] S. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. Sanders. Secloud: A Cloud-based Comprehensive and Lightweight Security Solution for Smartphones. *Comput. Secur.*, 37:215–227, September 2013.