Towards Cloud-Based Compositions of Security Functions For Mobile Devices

Gaëtan Hurel, Rémi Badonnel, Abdelkader Lahmadi, Olivier Festor

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 can be deployed on the mobile devices themselves. However, these applications may have a significant impact on the device resources. Users may be tempted to uninstall or disable them with the objective of increasing battery lifetime and avoiding configuration operations and updates. In this paper, we propose a new approach for outsourcing mobile security functions and building transparent in-path security compositions for mobile devices. The outsourced functions are dynamically activated, configured and composed using software-defined networking and virtualization capabilities. We present a mathematical model to formalize 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. In addition, the Internet of Things (IoT) increases the number of connected devices at a fast pace. As most of popular and widespread technologies, mobile devices are also an attractive target for 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, but also because they are likely to store private and sensitive data - and thus, valuable - about their respective owner. As a direct consequence, the number and the types of mobile malwares have significantly increased during the last years, ranging from some simple invasive spywares to transparent fraudulent premium-charge calls or trojan horses in the context of APT[2] campaign [8].

Most of current security solutions for mobile systems are available in the form of applications or packages to be directly installed on the devices themselves. Such approaches that we can call as on-device approaches offer some advantages, including a consistent view of the system's state during security operations, as well as the self-contained aspect they adopt. However, these approaches generally induce significant resources consumption on the devices leading to the reduction of the battery lifetime. In the meantime, current cloud-based solutions for mobile security try to deal with this issue by offloading the most of the workload on a remote server, while only installing lightweight agents on the devices. Such solutions permit to reduce the amount of used resources on the devices, but it remains at least two major problems. The first one is the implication of the users, who generally do not have the required knowledge to properly perform security decisions in case of settings or alerts for instance. The second one is the flexibility of such solutions and their capacity to contextualize 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 representation to model the security function compositions within our solution. Section IV explains our strategy for delivering composable and dynamic security compositions 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

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

[1]Bring Your Own Device
[2]Advanced Persistent Threat

Mobile security

In addition to traditional on-device approaches based on dedicated applications deployed on the mobiles [17], several cloud-based approaches have already been proposed in order to provide security to these devices. Some solutions exploit cloning methods using virtualization to execute security checks without resources constraints, such as Portokalidis et al. [19] and Zonouz et al. [25]. The critical point of such solution relies in the ability for each clone in the cloud to synchronize its internal state with the corresponding real device. Other approaches directly outsource security functions of the mobile devices as cloud services. For instance, Kilinc et al. [16] introduce a cloud-based applications firewall for Android devices, while Oberheide et al. [18] present a cloud-based antivirus for mobile devices considering different platforms. 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. In that context, the use of software-defined networking for transparently delivering mobile security has also been studied. In [15], Jin et al. focus on an appliance based on OpenFlow [6] to detect mobile malwares using traffic analysis methods performed at the network controller level. The analyzed traffic is gathered by the OpenFlow-enabled wireless access point. Our work aims at differ from those previous ones in the sense that it does not focus on some specific threats in a static 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. Benson et al. provides CloudNaas [10], an OpenFlow-based networking framework allowing customers to outsource line-of-business applications along with network functionalities - i.e. middleboxes - in the cloud. Subharti et al. perform a similar work with OpenADN [23], except for the main facts that OpenADN is able to deal with inter-cloud integration, dynamic resources scaling and application-level flow processing. In the meantime, Sherry et al. explore a new design to dynamically and transparently outsource middleboxes *only* across several cloud providers using NFV[3]-like virtualization and different redirection mechanisms [22]. Gibb et al. present an alternate work where a cloud-based architecture is designed for outsourcing network functionalities [13]. The main differences with [22] are the emphasized use of SDN-based mechanisms for redirecting the traffic through the middleboxes, and the use of a dedicated API for policy specification and service configuration. Our work is similar to [22] 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 [13]. 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 [21].

[3]Network Function Virtualization

Service chaining

In our context, chaining services is equivalent to composing middleboxes. Several recent works have leveraged the software-defined networking paradigm in order to manage and compose middleboxes in a consistent manner. Qazi and al. put forward SIMPLE [20], a SDN-based policy enforcement layer for middlebox-specific traffic steering. In this work, middlebox composition and routing policies are enforced by pushing the corresponding flow rules into the OpenFlow network. Fayazbakhsh et al. propose a similar work with Flowtags [11], an architecture where middleboxes are extended to support OpenFlow and use tags in network packets for determining how to process the corresponding traffic. Stratos [12], an orchestration layer for virtualized middleboxes in the cloud, is another possible solution for dynamically composing middleboxes. Like SIMPLE and Flowtags, Stratos uses SDN-based technologies to configure the forwarding plane in a centralized way. However, while the two previous ones respectively employ flow correlation and packet header tags, Stratos leverages its virtualized environment to address flow mangling due to dynamic middleboxes. Alternatively, Anwer et al. put forward Slick [9], 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 [20] 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 function compositions within our approach. We then describe how this model can be used for a given composition in order to (i) evaluate its cost in term of resources such as CPU usage, (ii) quantify its quality in terms of end-to-end delay for instance, and (iii) determine its scalability according to the amount of data to be processed. In that context, we explain how to exploit this modeling in order 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). We justify the fact that each composition graph must be acyclic because any loop among security functions of a given composition would induce additional latency regarding the delivery of the initial service. 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 one entry point and at least one or more exit point(s) - for the

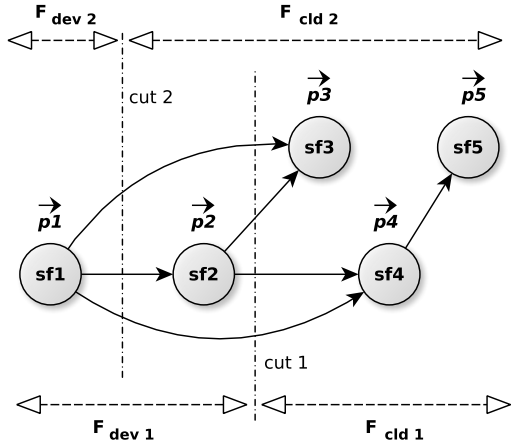


Fig. 1: An example of composition graph. The dotted line represents a possible separation (cut) between F_{dev} and F_{cld} as discussed in section III-C. This separation can dynamically change over time, according to the device's resources for instance.

remaining of this paper, we call such points *edge-points*. Entry and exit points are security functions within the composition that are respectively located at the start and at the end of that composition. 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_4 and sf_5 are the two possible exit points of the composition.

- **Connectivity and directivity**: each security function in a given composition graph must be directly (i.e. one hop) or indirectly connected with any exit point included in the composition. Also, 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 chosen metrics to characterize the security functions, these weights can be single scalar values as well as n-ary vectors.

B. Composition characterization

Each composition involves a set of security functions to be deployed either on the device or in the cloud. We explain here how to leverage the graph-based modeling of our approach in order 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 resources used 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. Using a composition graph, each security function sf_i of a given composition 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 use the CPU in a constant manner and have the same execution duration. The overall usage consists in the average of the different CPU usages induced by the security functions within C.

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

2) Quality of a composition: The quality of a composition is a quite broad term and may depend on several factors, such as the end-to-end delay and the security accuracy offered by the given composition. We mainly focus on the end-to-end delay in this work in order to know whether it is a real drawback of our approach due to the redirection in the cloud. 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 functions in the cloud are not significant as long as all those functions are located in the same area. In addition, the treatment delay represents the required time for a security function to perform all its security tasks on a given flow. Using a composition graph, each security function sf_i of a given composition 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 within 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 different treatment delays induced by the security functions within C.

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

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 3 and 4 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.

$$S_{CPU}(C) = K.P(C) \tag{3}$$

$$S_{E2E}(C) = K.D(C) \tag{4}$$

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 not want to be too cloud-dependent and prefers having crucial security functions running on his device. In the meantime, one may just want to preserve the battery life of his device and to outsource all the security functions in the cloud.

1) Constraints on F_{dev} and F_{cld}: 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} - note that we could also have chosen F_{cld} - could be for example $F_{dev} = \{sf_1\}$ or $F_{dev} = \{sf_1, sf_2\}$ as represented by the dotted line. Several graph-related constraints must also be respected by F_{dev} and F_{cld}:

- **Round trip minimization:** in order to avoid additional data and/or control round trips when analyzing a single flow, vertices (i.e. security functions) contained in F_{dev} - respectively F_{cld} - must be able to reach each others using a subset of directed edges from T without involving intermediate nodes in F_{cld} - respectively F_{dev};
- **Control flow transmission:** deriving from the previous point, 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 - depending on the traffic direction, the first subset could change (typically F_{dev} for the outgoing traffic and F_{cld} for the incoming traffic);
- **Edge-points partitioning:** deriving from the previous point, the second subset must include all the exit-points of the composition, for consistency purposes and duplicated traffic avoidance; conversely, the first subset always contains the entry-point of the composition.

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 the sake of simplicity, 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. Based on this modeling, we will detail in the next section our strategy for efficiently distribution security functions across the devices and the cloud.

IV. Cloud-based composable security strategy for mobile devices

We propose in this paper 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. The rationale behind moving security functions in the cloud is that setting up a large number of applications and maintaining them to entirely cover the security threats set is a difficult and overwhelming task, even for expert users. Furthermore, users requirements regarding security of their devices and more generally mobile security threats may vary significantly over time and depending on the context. Our strategy aims at addressing these 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 and compose dedicated functions for mobile devices.

A. Architecture

Our proposed architecture [14], depicted by Figure 2, involves three distinct entities in order to deliver dynamic and composable security to mobile devices: (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 such as an OpenFlow controller communicating with the switch of the mobile device; (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 are handled by the virtual switch of the device. At the beginning of the communication, the switch may probe the OpenFlow controller configured by the security manager 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 designss a dedicated security composition in a proactive or reactive manner. 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 all the incoming and outgoing traffic pass through that

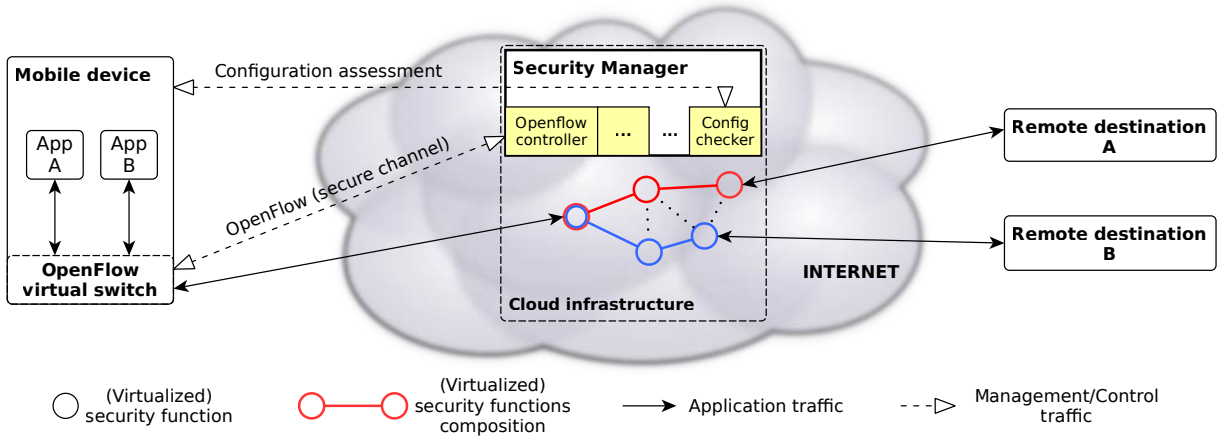


Fig. 2: A cloud-based composable security architecture for mobile devices.

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. In the meantime, a well-known gaming application should deserve less requirements from a security point of view, and some tradeoffs regarding whether or not to use on-device mechanisms could be considered in order to prevent unnecessary communication delays for instance. 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 first describe the different categories of traversals and how they may be implemented using our solution.

1) Composition traversal: As introduced in section III-A, a security composition can be seen as a directed graph $C = (F, T)$ where F is the set of involved security functions, and T the set of control flow transmissions among those functions. For the sake of simplicity, we assume that the data (e.g. the flow) to be analyzed is available to all the security functions within the composition - hence, we do not have to focus on the data flow transmissions. Within our approach, the control flow transmission at the end of a security function sf_i can use several schemes:

- **Sequential scheme:** a control flow transmission is sequential if and only if the control flow always goes to the next function sf_j after sf_i has finished. In the composition graph representation, sf_i has only one outgoing edge, and this edge goes towards sf_j.
- **Conditional scheme:** in its simplest case, a control flow transmission is conditional if and only if the control flow goes from sf_i to sf_j after sf_i has finished *and* a specific condition is respected, or sf_k otherwise. In the composition graph representation, 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 ... f_{i+N_C}. While most of the time only one outgoing edge will be used, conditional transmissions can lead to a concurrent control flow transmission - e.g. or-inclusive condition. It is worth mentioning that the conditions used to determine the next function to be executed may use differently grained semantics, ranging from a value to match in the packet header to a flag put on the flow by a previous security function.
- **Concurrent scheme:** a control flow transmission is concurrent if and only if the control flow always goes to multiple next function sf_j...sf_n after sf_i has finished. In the composition graph representation, 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 ... f_{i+N_F}. The major difference with the conditional scheme is that all the next functions are always executed after sf_i has finished.

2) Scenario examples: Composition traversal schemes within our architecture can be leveraged in order to build specific security compositions for some given security policies. Such a security policy could be "Block malicious outgoing traffic and prevent data leakage" for instance. The Figure 3 describes the composition associated to this scenario using a typical composition language, called BPMN[4] language. When the mobile device outgoing traffic enters the composition in the cloud provider infrastructure, it is first checked against several firewall rules in order to know whether it is allowed. If the traffic is authorized, it is then inspected by a lightweight intrusion detection system (IDS) and a data loss prevention

[4]Business Process Model and Notation

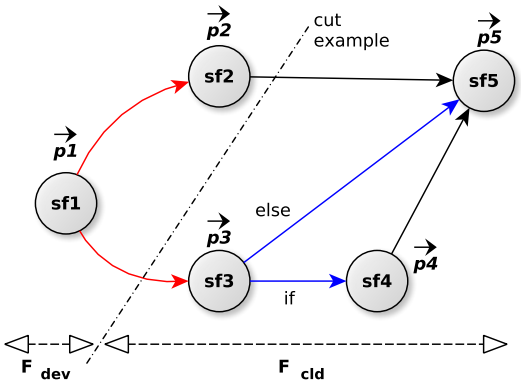


Fig. 4: Composition graph for a security policy "Block malicious outgoing traffic and prevent data leakage". sf_1 is the entry point, $sf5$ is the exit point.

engine (DLP) simultaneously. When the lightweight IDS flags the traffic as suspicious, a Deep Packet Inspection (DPI) is then executed in order to extract all malware-related patterns within the given flow outgoing traffic. If the traffic is not suspicious, the next security function after the IDS is the security analyzer. 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 - the composition entry-point - and finishes at the security analyzer - the composition exit-point. It is worth mentioning that in such a composition, all composition traversal schemes are exploited: 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 different experiments we conduced using this prototype in order to evaluate the performances, in terms of resource cost, end-to-end delay and scalability of the security compositions.

A. Experimental setup

We have developed an implementation prototype to evaluate our solution. This prototype 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, and (3) the remote destinations interacting with the mobile device. Along this prototype, our experimental setup includes two different types of security functions and a small Android application to generate network traffic (workload). The next paragraphs describe our configuration choices for setting up the testbed. We also explain how we addressed several connectivity constraints due to the closed nature of the emulated environment.

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. In order to integrate properly the OVS on the device, we had to patch and cross-compile the associated kernel module (openvswitch.ko) and userland tools (vswitchd, ovsdb-server, ovs-vsctl, ovs-ofctl) on a separate computer using the Android NDK[5]. Then, we deployed them on the device using the Android Debug Bridge (adb). Once started, the OVS switch acts between the internal networking stack and the built-in WiFi interface of the device. Our device setup is strongly inspired from [24], 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: In order to build and configure the cloud infrastructure, we used the Mininet emulator [3] - which supports the OpenFlow 1.0 specification - in a virtual machine built with VirtualBox. The host computer was a Linux Ubuntu 12.04 LTS with an Intel Xeon CPU E5-1620 and 32 Gb of DDR3 RAM. We configured the Mininet VM to use 16 out of the 32 Gb available RAM and one out of the eight core processors. We leveraged Mininet to build an OpenFlow-based network by emulating OVS switches (version 1.10.0) to forward the traffic, and standard Linux hosts to host security functions. The OpenFlow controller was not emulated by Mininet and was instead a dedicated controller running within the Mininet VM - therefore, in the same network namespace as the emulated switches. We chose the POX controller for our experiments, since it had all the capabilities we required and is well-known by the community. 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 originally chose to use iptables/netfilter [4] as the only security function since it was a built-in feature both in the Android device and the Linux hosts within Mininet. The idea was to be able of having identical security functions on both these types of systems in order to effectively compare their performances. Then, we have succeeded in porting the Suricata IDS/IPS [7] to Android, thus adding a new security function for our testbed. Note that we used Suricata as an IDS since we were unable to cross-compile it with the NFQUEUE support in time. 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. Those emulated hosts were offering specifically configured services for our

[5]Native Development Kit

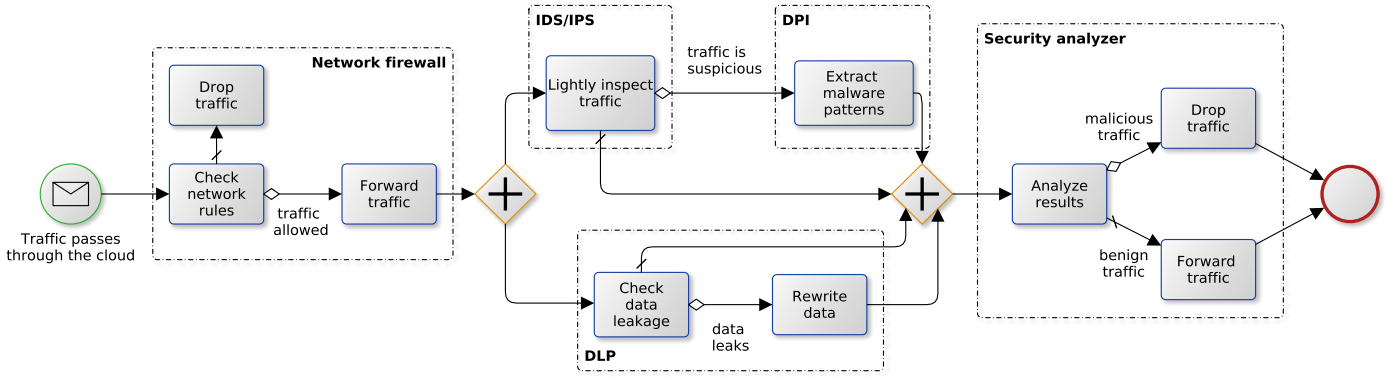


Fig. 3: An example of composition for the policy "Block malicious outgoing traffic and prevent data leakage".

experiments, and were in the same network as the mobile device.

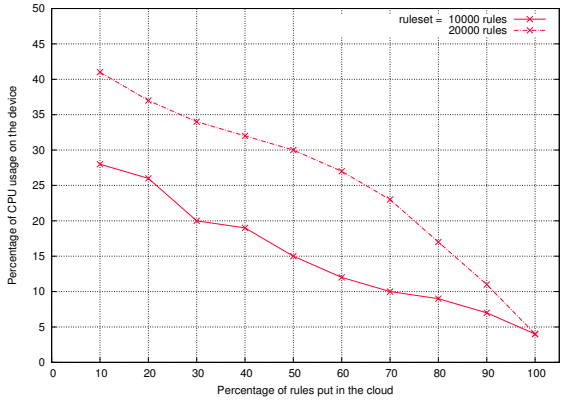
5) Workload: We needed to generate a significant amount of network traffic from the smartphone to evaluate the processing performances of the security functions - whether they were put on the device or in the cloud. To this end, we developed a small Android application able of sending a given amount 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. We typically used this application on the smartphone to send a large number of HTTP requests to the web servers within the Mininet environment.

Based on this setup, we have addressed several connectivity issues as follows (1) find a way for the mobile device to interact with the OpenFlow controller, (2) find a way for the mobile device to send outgoing traffic - and receive incoming traffic - through security compositions within the Mininet environment, and (3) find a way for the security compositions edge-points to send traffic to - and receive traffic from - the smartphone and the remote destinations. Regarding the issue (1), we configured the host computer running VirtualBox to set up a local NAT-ed network with wireless support where the mobile device and the OpenFlow controller in the Mininet VM could be attached. Hence, the device and the controller were on the same local network and were able to talk each other. For the issues (2) and (3), a specific emulated host within the Mininet environment was also attached to the local network introduced in the previous point. This specific host was running an OVS switch in its own network namespace, thus allowing (i) the mobile device to directly send traffic to the specific host, which injected it in the composition according to some specific OpenFlow rules, and (ii) the specific host to send the device traffic to the remote destination after the security treatments are done. Said another way, this specific host was the only edge-point of each security composition, acting as both entry and exit point. It is important to note that, by default, Mininet emulates hosts in a closed environment and only NAT mechanisms can be employed for accessing Internet. However, our requirements were slightly different, since we needed the mobile device outside the Mininet environment to be able of initiating communications and interacting with specific hosts in the Mininet environment. In summary, OpenFlow is used within our approach for two separate reasons, namely (1) perform traffic redirections between smartphone, cloud and remote destinations using address rewriting features, and (2) enforcing traffic pass through the wanted functions in a given composition with concurrent and conditional paths using specific rules and possibly header tags. As stated above, the corresponding redirection and forwarding logic was implemented as a simple POX module written in Python.

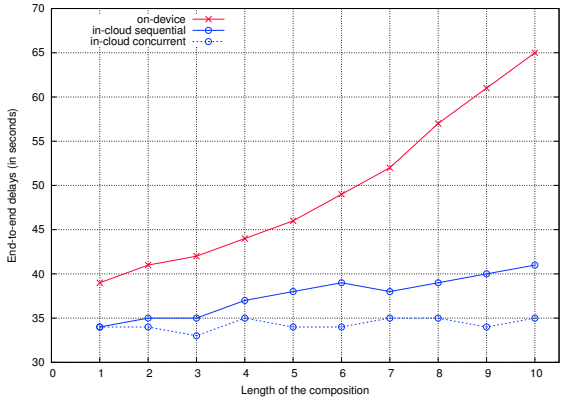
B. Experimental results

We have performed an extensive set of experiments using our prototype in order 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 - which means, directly between two OpenFlow switches and thus on the direct path of the traffic to be analyzed. The compositions we used were either IDS-only compositions for CPU usage measurement or firewall-only compositions for end-to-end delay measurement. Indeed, an IDS does not affect latency since it works by inspecting packet capture files, while a firewall directly works on the traffic in a real-time manner.

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 (i.e. signatures) 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 according to the number of IDS rules it carries. As expected, the more the IDS rules are outsourced in the cloud, the better are the resources savings performed on the device. 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%



(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 sequential and concurrent firewall compositions up to ten firewalls (each with 1000 rules).

Fig. 5: Experimental results regarding maximum CPU usage and average end-to-end delays of the compositions.

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. For the device, we simply increase the set of rules of the firewall according to the length of the composition. For the cloud, firewall compositions are distributed across several hosts, and either use sequential traversal or concurrent traversal. Once again, we send and receive a same amount of network traffic on the device. As stated in III-B2, we consider that transmission delays are not significant within our testbed, and only treatments delays are therefore likely to influence the end-to-end delays. Results for this set of experiments are shown in Figure 5a. We observe that the average end-to-end delays induced by sequential compositions grow significantly on the device, from 38ms up to 65ms, according to the number of firewall rules it carries. This grow is also slightly visible for cloud-based sequential compositions, yet which offer much lower latency compared to on-device compositions. In the meantime, cloud-based concurrent compositions show the best results since security functions can scale-out in such a way that each of them process a same amount of rules simultaneously. Such performances must however be balanced with the overhead induced by an eventual synchronization step required after the concurrent processing. We did not take into account the synchronization step during this set of experiment. We conclude that treatment delays - and thus 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.

3) Scalability: Due to some difficulties and constraints related to the Mininet environment, we were unable to properly evaluate the scalability of the compositions in time. We keep for later this task and plan to this end to leverage an other emulator system such as the Distributed OpenFlow Testbed (DOT) [2], or a dedicated hardware testbed.

VI. Conclusions

In this paper, 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. While we were unable to properly assess the scalability of the compositions, we keep for later this task and plan for this to leverage an other emulator system such as DOT.

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 flows - including malware communications - in order to automate the policy specifications themselves. In the meantime, redirection mechanisms are a key part of our solution. We therefore want to study possible protocols (e.g. NETCONF [5], encapsulation) in order to find alternatives to the OpenFlow client. 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.

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 Distributed OpenFlow Testbed. http://dothub.org/. Last visited in september 2014.

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

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

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.