Securebox: Toward Safer and Smarter IoT Networks

Ibbad Hafeez[†], Aaron Yi Ding[‡], Lauri Suomalainen[†]
Alexey Kirichenko[⋆], Sasu Tarkoma[†]
[†]University of Helsinki, [‡]Technical University Munich, [⋆]F-Secure Corporation

ABSTRACT

In this paper we present Securebox, an affordable and deployable platform for securing and managing IoT networks. Our proposal targets an alarming spot in the fast growing IoT industry where security is often overlooked due to device limitation, budget constraint, and development deadline. In contrast to existing host-centric and hardware-coupled solutions, Securebox empowers a cloud-assisted "charge for network service" model that is dedicated to budget and resource constrained IoT environments. Owing to its cloud-driven and modular design, Securebox allows us to 1) flexibly offload and onload security and management functions to the cloud and network edge components; 2) offer advanced security and management services to end users in an affordable and on-demand manner; 3) ease the upgrade and deployment of new services to guard against abrupt security breakouts. To demonstrate Securebox[1], we have implemented the platform consisting of a plug-n-play frontend, a Kubernetes-powered backend cluster, and a smartphone mobile application. Based on the testbed evaluation, we show that Securebox is robust and responsive. Its collaborative and extensible architecture enforces rapid update cycles and can scale with the growing diversity of IoT devices.

Keywords

IoT, Cloud-Assisted Security, SDN, Docker

1. INTRODUCTION

Internet of Things (IoT) has transformed from a buzzword to a reality in recent times. With the trend of integrating IoT to home, office and enterprise networks, the number of connected IoT devices is projected to reach tens of billions in the next few years.

Due to the security and privacy risks exposed by IoT devices, it is crucial to protect IoT networks. However, it has become increasingly difficult to secure IoT networks due to the its scale, diversity of devices and deployed environments, and lack of expertise and resources for IoT users.

[1]Demo video: https://www.cs.helsinki.fi/group/close/secDemo/securebox.html

It is clear that the conventional security mechanisms from IT domain (e.g., antivirus, firewalls) are insufficient to address the challenges posed to IoT networks [7]. The IoT devices are typically developed by fast moving teams in large enterprises or independent startups that have limited budget and hard deadlines to launch their devices. Their software design do not follow strict security guidelines. More alarmingly, these devices are often resource constrained, which complicates the adoption of existing host-centric security solutions. The large number of IoT manufacturers also make it hard to enforce cross-device security policies and life cycle support (e.g., security patches).

As typical IoT users (e.g., home users with insufficient expertise to manage and update IoT devices by themselves) have limited budget to deploy expensive hardware-based security solutions, there is an urgent demand for an affordable solution that is flexible and can scale to the exponential growth of IoT installment.

In this paper, we present Securebox, a cloud-assisted platform dedicated to budget and resource constrained IoT environments. It combines the advantages of SDN and NFV (i.e., Docker container) to offer management and security services. Our main contributions can be summarized as follows:

- We concretize the concept of outsourcing home network management and security functionalities via Securebox. It enables a cloud-assisted "charge for network service" model dedicated to budget and resource constrained IoT environments. To the best of our knowledge, Securebox is the first platform dedicated to IoT networks that allows IoT users to subscribe various security and management services in an on-demand manner. Our solution addresses several limitations of current state of the art and can be deployed incrementally in the existing infrastructure.

- Through experimental measurements, we demonstrate the applicability of the solution. The observations and lessons from our experimentation serve as valuable input for the community to avoid potential pitfalls in system implementation, and shed lights on the future development and deployment.

In the rest of the paper, we illustrate the design and implementation of Securebox in Section II and present our evaluation[2] in Section III. We further describe the related work in Section IV and conclude in Section V.

2. THE SECUREBOX APPROACH

To tackle the challenges in IoT environments, we propose a new platform for improving security and management in these networks. It consists of two primary components, *Securebox Frontend* (SF) and *Security and Management Service* (SMS)[3]. The platform facilitates a set of cloud-based security and management services, including traffic analysis, device state and user preference management, and automated security updates. It is designed to scale for deployment in various networks ranging from SOHO to enterprise environments.

2.1 Securebox Frontend

Securebox Frontend (SF) is a programmable gateway for securing devices connected to the network. It applies SDN to dynamically configure the network policies for each device based on its context and security preferences. Figure 1 presents the internal architecture of SF which consists of an SDN controller managing the traffic from devices connected to wired/wireless interfaces, a local policy database (pol-db) as a cache for security policies, and a management console for user interaction accessible via smartphone or web application.

SF acts as a security enforcer at the network edge, which uses SMS to perform traffic analysis operations, maintain user and device state etc. SF provides a number of features including device isolation, controlling device to device (D2D) communications, device discovery and profiling etc. SMS assists SF in managing and automation of all these features to minimize the need of user intervention.

SF is designed to be lightweight, low cost and easy to deploy. Therefore, it offloads traffic analysis and state management tasks to SMS. SF requests SMS to perform desired traffic operations on live-traffic or connection metadata information. The security policies received in response to these requests are cached locally to minimize redundancy. This approach reduces the latency experienced by the user and improves the cost efficiency and scalability of the platform.

Any connection request from a connected device to a remote destination is intercepted by SF. It checks pol-db for a security policy matching the requested connection. If a matching policy is found, the connection is allowed/blocked depending on the policy decision. If not found, SF requests SMS to analyze this connection request. SMS responds to this request with a security policy (with allow/block decision) which is enforced in

[2]High resolution image are available at http://goo.gl/PN2nv9

[3]We leave out the design details of smartphone application due to page limit.

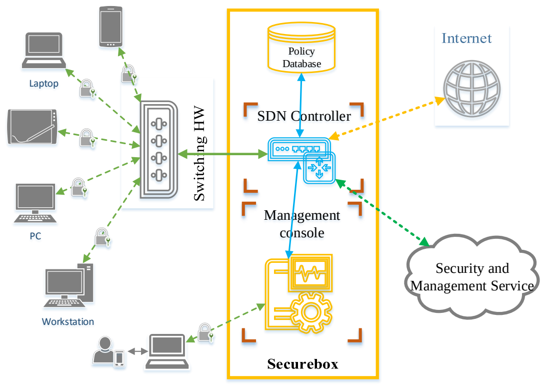


Figure 1: **Securebox frontend internal architecture**.

the network and cached in pol-db for later use. SF also receives periodic security policy updates from SMS to increase policy hits for upcoming connection requests.

All policies received from SMS have an associated *time-to-live* (ttl), which is refreshed every time a policy is used. If the ttl expires before policy is reused, it is removed from pol-db. This prevents the ballooning of pol-db size and lookup time.

SF can be configured to use any SMS and corresponding services depending on user preference. The management console is provided via a smartphone application to improve user experience. This application provides security ranking for connected devices based on device activity. It also generates warnings when a suspicious activity is detected and quarantined, to increase user awareness about device and data security.

2.2 Security and Management Service

SMS is a flexible and highly scalable alternative to hardware based middleboxes used for network security. It deploys (virtual) software middleboxes for analyzing user traffic and provides automated configurations updates for SF to prevent recently discovered attacks in edge networks. SMS also supports a number of other services including malware, botnet, spam detection etc.

SMS is designed to assist SF and enable the "charge for network service" model through cloud-based services. Figure 2 shows the architecture of SMS where *cloud manager* is responsible for handling traffic analysis requests from SFs. Upon receiving a new analysis request, *cloud manager* validates the request using *certification server*.

Based on user preferences, *cloud manager* either checks *database* to find a matching security policy for requested metadata and return this policy to SF where it is cached and enforced in the network. Otherwise, it requests *middlebox manager* to provide a middlexbox instance to handle the service request. User can tunnel their traffic through this middlebox for analyzing live traffic. SMS also maintains state-aware live replicas for cloud man-

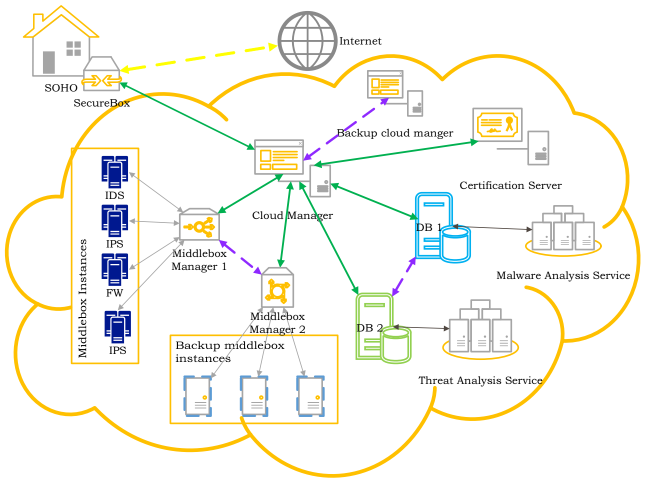


Figure 2: **Architecture for cloud-based Security and Management Service.**

ager and middlebox instances for fault tolerance and minimizing service downtime.

SMS receives large amount of information from all connected SFs. It combines this information with knowledge from external sources e.g. virus signature databases, malware databases etc. to run various analysis services and detect new threats. This approach helps in detecting the tiny traces of malicious traffic which usually goes undetected through the traditional network perimeter security systems. Based on the aggregated knowledge, SMS generates security policies to prevent newly discovered networks attacks. These security policies are sent via pol-db updates to SFs.

SMS empowers a collaborative approach to minimize the chances of an attacker using same techniques or compromised machines to launch successful attacks against disjoint networks. Our results have shown that the collaborative approach significantly minimizes the risk of network attacks by sharing attack related information across different networks.

Policy-DB updates Policy database updates are generated by aggregating the information obtained from the various analysis services shown in Fig. 2. These updates consist of frequently accessed security policies. Therefore, sending these policies to all SFs will result in handling most of the traffic locally. It will minimize the latency experienced by the users and the load on SMS as well.

In the proposed system design, high priority policies are immediately updated to the SFs whereas policies having less priority are bundled together and sent to the SF during hours of less network activity e.g. nighttime.

2.3 Prototype

We have implemented a prototype system and tested it for a number of scenarios to evaluate system performance. In this section, we present implementation details about SF and SMS.

2.3.1 Securebox Frontend

We have implemented two different variants of SF using Raspberry-PI (R-PI) i.e. low cost ($35) small form factor computer, and Fit-PC3 with better hardware resources costing $533. Our evaluation results show that the design of Securebox allows us to use minimal hardware without taking a performance hit. Results discussed in Section 3 show that both R-PI and Fit-PC3 based versions are able to achieve similar performance and user experience.

We have written a new module in Floodlight SDN controller for Securebox, which intercepts all traffic to perform traffic filtering, device management and other Securebox operations. This module offers REST APIs for external communication with SMS, smartphone or web applications. We have used hostapd to setup R-PI as Wi-Fi AP. Pol-db is currently setup using JSON file but it can also use SQLite or NeDB.

2.3.2 Security and Management Service

We have implemented SMS from the scratch using Flask Framework [4]. SMS provides management and security services for user, device and SF. It provides device and user registration, setting up preferences, contextual policies for device and Securebox-based security and management etc. via web or smartphone application.

We have deployed SMS (application module and middlebox instances) in our laboratory on Kubernetes cluster using Docker containers. We have written custom APIs for SMS to communicate with Kubernetes cluster for dynamic provisioning and state management of middlebox instances. Our evaluation results show that SMS can be deployed using commodity hardware.

We also implemented a smartphone application to allow users to interact with Securebox platform. It provides information for all user device, their security rating, associated risks and threats, data used etc. Users can also configure their preferences as high level commands e.g. enable parental control on a device, limit network connectivity for guest devices etc. via smartphone application. SMS also generates notifications for the users about current state of their network using this application.

3. EVALUATION

This section gives a detailed discussion over system performance in terms of user experience and scalability. For testing purpose, we have setup Kubernetes cluster using Dell Optiplex 960 workstations. We deploy SMS and middleboxes (custom written dynamic IP blocking service and SNORT [5]) using Docker containers on this cluster.

3.1 Latency

[4]http://flask.pocoo.org/

Latency is very crucial for end users and deployability, as it can critically affect user's experience for any system or service. The proposed system is susceptible to increase the latency experienced by the user as the traffic analysis tasks are offloaded to remote service. Section 2 lists a number of design choices adopted for Securebox to minimize the impact on latency.

$$L = \sum_{i=1}^{n} l_i + bl \tag{1}$$

In Eq. 1, L is the total latency experienced with Securebox setup and bl is the baseline (using no Securebox) latency between client and an online webserver. $\sum_{i=1}^{n} l_i$ is the total latency added by Securebox (while getting a security policy for the requested connection). When a matching policy is found in pol-db, $\sum_{i=1}^{n} l_i$ is less than 80μ seconds at $229.54 MB/s$ using a Micro SD storage card in R-PI. Combining policy database updates and local pol-db helps minimize latency by increasing chances to find policy hits in local cache.

l_i is the latency introduced by each operation performed in traffic request analysis process e.g. connectivity to SMS, request verification, database lookup, middlebox provision etc.

$$L = \sum_{i=1}^{k} l_i + \sum_{j=k}^{n} l_j \tag{2}$$

$$L = l_1 + l_2 + l_3 + ... + l_k + \sum_{j=k}^{n} l_j \tag{3}$$

Equation 3 expand $\sum_{i=1}^{n} l_i$ where $l_i,\ i \epsilon \{1, ..., k\},\ k \geq 2$ are the latencies which can be bounded e.g. l_1 is the latency between Securebox and central server, l_2 is the time taken for request verification using certification authority, l_3 is the time taken by database lookup operation for relevant security policies. $\sum_{j=k}^{n} l_j = 0$ when traffic is not analyzed using any middlebox or traffic analysis service.

$l_j,\ j \epsilon \{k, ..., n\}$ are the latencies for middlebox operations. These latencies vary according to user preferences e.g. the latency would be different when traffic is not processed in any middlebox compared to when it is processed in FW, IDS and malware analysis service.

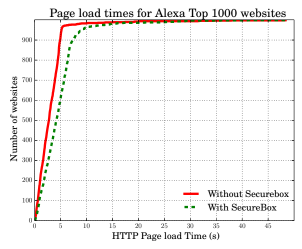
$$L = \lceil C \rceil + \sum_{j=k}^{n} l_j \tag{4}$$

Equation 4 combines all latencies which can be bounded into C. We try to minimize C by changing SMS design, operations sequence etc. In this section, we evaluate our system prototype in a number of scenarios to study the impact of latency caused by the system on user experience.

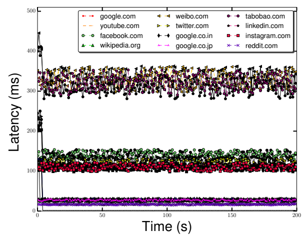
3.1.1 Internet browsing

Web browsing is a common use-case for SOHO users deploying IoT and its experience is greatly affected by latency. It is known that increased latency can cause a significant decrease in user satisfaction and affect traffic directed to the service [3].

Figure 3a shows the CDF plot for the time taken to load top 1000 websites ranked by Alexa (as of 31st July 2016). It shows that Securebox only increases latency by $\leq$ 20% which is not substantial in terms of common user experience e.g. Securebox increases latency for Youtube[5] to 3.81 ± 0.25 ms compared to 3.53 ± 0.30 ms using traditional network setup.



(a)



(b)

Figure 3: **Comparison of latency experienced during web browsing**. (a) CDF plot for HTTP page load times for Alexa Top 1000 websites. (b) Latency experienced when browsing top websites with and without using Securebox.

Figure 3 shows the latency experienced by the user when he first browses a website (i.e. without pol-db cache). Once the policies are cached in pol-db, user will experience the same latency as experienced in normal setup without Securebox.

Figure 3b further shows the latency experienced during browsing of individual websites. Again, the latency increased only when webpage was requested for first time and subsequent requests had (almost) no added latency because all requests were addressed locally using pol-db.

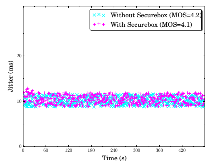
The (extra) latency introduced by Securebox setup is nominal because we only use minimal data from the connection requests and analyze it using lightweight services. Equation 1 shows that overall latency is directly related to the kind of analysis performed in the cloud e.g. running DPI on live traffic will introduce much high increase in latency. However, these results show that the latency introduced by communicating with SMS (and internal operations), is very small compared with the latency experienced due to traffic analysis using middleboxes e.g. IDS/ IPS etc.

3.1.2 VoIP performance

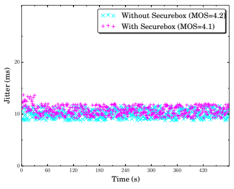
VoIP is another use-case where user experience is greatly affected by latency. We have tested our proto-

[5]www.youtube.com

type using Skype[6], which is a popular VoIP application. We have setup two different scenarios in which either or both of users (i.e. Alice and Bob) use Securebox-based network setup and we compare the jitter experienced in each of these scenarios with the jitter experienced in baseline scenario where both of them use traditional network setup.



(a)



(b)

Figure 4: **Performance evaluation for VoIP performance when using Securebox.**

Figure 4a and 4b show that intially the jitter was high because the connection requests were analyzed by SMS during call setup and once the SF receives security policies from SMS, the jitter experienced is same as that of traditional network where Securebox is not in use. We achieve MOS score ≥ 4 in both setups which shows that Securebox supports VoIP applications with excellent QoE.

3.1.3 *File transfer performance*

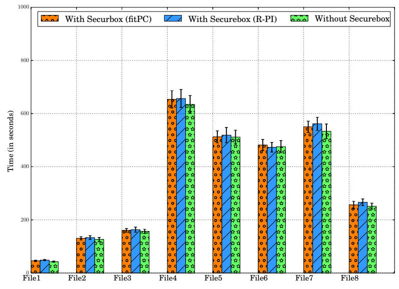
Phishing attacks, where an attacker sets up fake webpages (similar to original webpage) to redirect users to compromised server hosting malicious content which can infect user machines. Securebox actively inspects connection requests and blocks any such attempts to connect user to untrusted websites setup for phishing attacks.

While preventing users from phishing attacks, Securebox introduces a slight delay in file transfers as it analyzes the sources of this data. We have tested our system to identify the amount of delay Securebox introduces during file transfer over FTP/HTTP protocol.

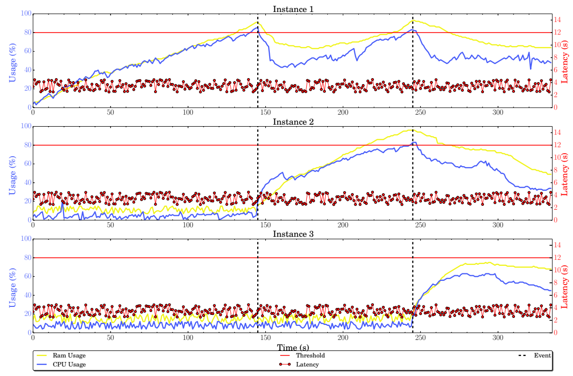
3.1.4 *Using FTP/HTTP protocol*

We have compared the time taken to download a file with Securebox-based network setup and with traditional network setup. The files were downloaded from publicly available servers on the Internet. Figure 5a shows that Securebox only introduces a slight increase ($\leq$ 1%) in completion time. The difference is little because in general each file is downloaded from (logically) single server using parallel connection, therefore, Securebox only needs to analyze few data sources. Securebox then caches the security policy locally and nomore analysis requests are made unless the data source changes.

[6]www.skype.com



(a)



(b)

Figure 5: (a) Comparison for time taken to transfers files over FTP/HTTP protocol with and without using Securebox. (b) SMS scales the number of instances depending on incoming traffic analysis requests.

3.1.5 *Using Bittorrent protocol*

Bittorrent is an interesting use-case as it downloads file contents from a number of peers who can not be fully trusted. These peers are not static and a client may need to switch between a number of peers depending on the data availability and transfer rate of the peer. Therefore, the number of requests made to SMS can be much greater compared to that of FTP/HTTP file transfer.

Our results show that Securebox increases the transfer time for file1 (1280MB) and file2 (280MB) from 691 to 707 seconds, and 120 to 131 seconds, respectively.

3.2 Scalability

Section 2.2 suggests that one of the key advantages of using remote analysis services (i.e., from the cloud) is its ability to scale during heavy traffic such as flash crowds. Figure 5b shows the capability of SMS to scale with the load of incoming traffic requests. Initially all requests are served by instance I. When the CPU usage crosses a threshold at Event 1, SMS redirects some of the requests to instance II. As the incoming request volume keeps increasing, SMS can dynamically launch instance III to share the request processing load.

3.3 Fault tolerance

Section 2.2 explains that Securebox is robust and can maintain state aware replicas of middleboxes and cloud manager to improve fault tolerance of the system. Figure 6 depicts one scenario where an operational middlebox (OPM) breaks down during operations. When the OPM goes down, SMS elevates its backup instances to become master and handle the traffic analysis tasks. SMS either launches a new replica for this master and the previous master (if recovered) starts serving as backup replica node.

The number of backup instances to maintain depends on the criticality of services. For example, when OPM is processing normal traffic, one replica could be enough, but if OPM is analyzing latency sensitive traffic then more than one replicas should be maintained. If there are more than one replica, any one of them can become a master as SMS ensures that all of them are same replicas of the master.

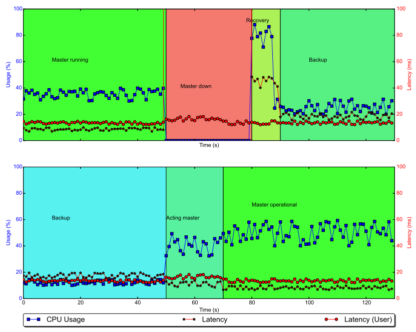


Figure 6: **Fault tolerance in middlebox operations using backup replicas**.

Since the replicas can perform same traffic analysis operations, if the result of these analysis is different between master and backup replicas, it alarms SMS for a state inconsistency. SMS then takes relevant actions to update middlebox configurations to ensure accuracy of traffic analysis operations.

4. RELATED WORK

The idea of offloading home network management tasks was first introduced by Nick Feamster [4]. Various systems have been proposed improve home network management [2, 3]. Researchers have also proposed the use of virtualized software middleboxes for securing the networks to reduce cost and improve scalability [6]. These systems suggest that user traffic can be redirected through remotely deployed middleboxes to provide better security [7]. However, these systems requires users to define what traffic should be processed through remotely deployed middleboxes [1].

Some commercial products e.g. Cujo[7], Dojo[8] have also been designed for improving network gateways. These products promise to secure the network by using remotely deployed services and machine learning techniques. However, they do not provide any details (until now) about performance, device functioning, user data collected from network etc. They also do not provide any control how the user traffic should be handled.

5. CONCLUSION & FUTURE WORK

While many other IoT projects concentrate on device management and policy enforcement, Securebox is different: through the cloud-assisted platform and virtualized modular design, it enables a “charge for network service” model to augment IoT security and management. The goal is to alleviate the impending risk caused by the large installment of insecure IoT devices with potentially unfixable flaws. It is our endeavor to realize the vision of outsourcing network management and security services to third parties [4, 7]. Instead of replacing existing solutions, Securebox exhibits a cost-effective alternative that can be deployed incrementally in the existing infrastructure. We are currently enriching the features of Securebox such as to support password-free WiFi access and data cap per IoT device. Our next step is to integrate Securebox with the commercial F-Secure Sense product for real-network experimentation and deployment. We plan to release the source code after the integration process.

Acknowledgements

This is work is carried out in TAKE-5 and Cloud Security Services (CloSe) projects funded by Tekes and Academy of Finland in collaboration with F-Secure and Nokia. We thank Seppo Hätönen and Ashwin Rao for their feedback and technical supports.

6. REFERENCES

[1] A. Alwabel, et al., “SENSS: Observe and Control Your Own Traffic in the Internet”. In *Proceedings of SIGCOMM 2014*.

[2] I. Bozkurt, T. Benson, “Contextual Router: Advancing Experience Oriented Networking to the Home”. In *Proceedings of SOSR 2016*.

[3] M. Chetty, et al., “uCap: An Internet Data Management Tool For The Home”. In *Proceedings of CHI 2015*.

[4] N. Feamster, “Outsourcing Home Network Security”. In *Proceedings of HomeNets 2010*.

[5] M. Roesch, “Snort - Lightweight Intrusion Detection for Networks”. In *Proceedings of LISA 1999*.

[6] J. Sherry, et al., “Making Middleboxes Someone else’s Problem: Network Processing As a Cloud Service”. In *Proceedings of SIGCOMM 2012*.

[7] T. Yu, et al., “Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things”. In *Proceedings of HotNets 2015*.

[7]www.getcujo.com
[8]https://www.dojo-labs.com/product/dojo/