Using SGX-Based Virtual Clones for IoT Security

Rashid Tahir
University of Prince Mugrin
r.tahir@upm.edu.sa

Ali Raza
Boston University
aliraza@bu.edu

Fareed Zaffar
LUMS
fareed.zaffar@lums.edu.pk

Faizan Ul Ghani
LUMS
mfgpk1286@gmail.com

Mubeen Zulfiqar
LUMS
18100211@lums.edu.pk

***Abstract*—Widespread permeation of IoT devices into our daily lives has created a diverse spectrum of security and privacy concerns unique to the IoT ecosystem. Conventional host and network security mechanisms fail to address these issues due to resource constraints, ad-hoc network models and vendor-centric data collection and sharing policies. Hence, there is a need to redesign the IoT infrastructure to secure both the device and the data. To this end, we propose a design where users are in the driving seat, devices are less exposed and data sharing models are flexible and fine-grained. Our proposal comprises hardware-secured data banks based on Intel Software Guard Extensions (SGX) to house the data in clouds without the need to trust the cloud provider. Virtual clones (shadows) of devices running on top of these data banks serve as competent proxies of actual IoT devices hiding away device weaknesses. The proposed infrastructure is scalable and robust and serves as a good first step for the community to build on and improve.**

***Index Terms*—Internet of Things, Software Guard Extensions, Security and Privacy**

I. Introduction

IoT devices are largely characterized by a substantial lack of computational resources. This fundamental limitation implies that devices cannot put up meaningful defenses (which tend to be resource-intensive) in the face of a motivated adversary. As a result, there has been an upsurge in the number of attacks reported for IoT devices, such as burglars breaking into homes by compromising smart door locks [1], arsonists causing a fire by abusing a smart oven [2] and complete strangers talking to babies by hijacking baby monitors [3].

IoT devices are also being misused at a macro level, such as the Mirai DDoS botnet [4], which allowed attackers to launch the largest DDoS attack in history solely off of hijacked IoT devices. Numerous other issues have been reported including traffic hijacking, weak/flawed encryption (one alphabet long keys), Man-in-the-Middle etc. [5], [6]. These problems stand to grow further in severity as IoT moves into the realm of critical applications, such as healthcare and life support where vital devices like insulin pumps and pacemakers are becoming increasingly more common [7] and attacks can prove to be fatal.

Furthermore, given that devices collect large amounts of fine-grained sensitive data using a wide array of sensors, privacy has emerged as a major cause for concern [8]. For instance, Samsung's Smart TV recorded all conversations in its vicinity to the extent that Samsung itself had to warn consumers to not have any sensitive discussions near the device [9], [10]. Similarly, We-Vibe, which manufactures smart vibrators and other adult toys, was found to be spying on its users and gathering data of an extremely personal and private nature without giving any warnings to the users [11]. In this case, the vendor was hit with a class action lawsuit and settled for 3.75 million dollars.

978-1-5386-7659-2/18/$31.00 ©2018 IEEE

The fact that there is a growing need for IoT devices to communicate with each other and share telemetry data further exacerbates the situation. For instance, if a light sensor installed on the window detects that it is getting dark, it needs to speak with the smart bulb and smart blinds so as to have them act accordingly. This implies that data of sensitive nature could end up being shared with untrusted third-parties. Smart home hubs [12] and connected vehicle platforms [13], which serve as an interoperability layer for a wide variety of devices to communicate are typical examples of devices sharing data with each other. Currently, lack of security standards implies that data is shared openly and in an all-or-none manner.

In light of these issues, we present the architecture of a cloud-based service for integrating security, privacy and trust into the IoT pipeline. Our proposal argues that users should own the data that IoT devices generate and should have explicit control over who to share the data with and at what granularity. To this end, we leverage Intel's Software Guard Extensions (SGX) [14] to create hardware-protected *enclaves* in the cloud, running virtual clones (shadows) of physical IoT devices and storing and processing device-generated data before sharing it with others. This allows us to put resource-intensive defenses in the cloud to protect devices when they communicate with malicious actors. Furthermore, sophisticated policy enforcement mechanisms can be used to perform strict access control and scrubbing of data before it is shared with others, giving users complete control of who gets their data and at what granularity. The design has several desirable properties and implications. First, moving intelligence to the cloud allows us to take advantage of more computing resources along with other cloud-based features such as scalability, fault tolerance and GPU-based analytics, all of which are lacking in IoT devices. Second, the SGX-based design allows for device management only via authorized users and removes the need to trust the cloud vendor. Third, users can explicitly control data sharing by selecting policies from a large collection of policies, ranging from simple to complex (developed by the service provider). Multiple policies can be compounded together to create a pipeline for highly processed data. For technically challenged users, the service provider can recommend default policies based on the nature of sensor data and the need of the

requesting party. Fourth, devices communicate with each other in a privacy preserving manner as data is processed, scrubbed and encrypted before being shared. Fifth, the communication model we adopt thwarts a large range of remote attacks, as the attack surface is moved from resource-constrained devices to their "bulky" clones in the cloud, where advanced authentication mechanisms and in-depth defenses are able to detect and prevent a variety of attacks from succeeding. The proposed design is a work-in-progress, hence we briefly touch upon implementation details and show a couple of preliminary results towards the end.

II. Problem Space

Before discussing the design, we first identify various prominent problems with IoT devices and in later sections attempt to address the highlighted concerns:

1) **Trust:** Within a closed ecosystem of IoT devices (such as a group of security cameras connected to a base monitoring station), there is an implicit trust assumption leading to numerous problems if one of the IoT devices or the base station gets compromised. Additionally, due to the ad-hoc nature of affiliated networks, trust boundaries become blurred resulting in a lack of checks and balances when data crosses from the trusted to the untrusted domain.
2) **Lack of authentication:** Insufficient authentication and authorization mechanisms plague almost all classes of IoT devices [5] such that some devices have a one alphabet long password.
3) **Insufficient encryption:** There is a severe lack of data transport encryption [5], [6] when devices communicate with each other, their corresponding apps and or send traffic over the Internet.
4) **Improper storage of credentials:** At times, data and credentials are stored on the device itself, often in plaintext form, or weakly encrypted due to lack of entropy/randomness [15].
5) **Denial of service:** IoT devices usually have medium to low resources making them vulnerable to resource exhaustion attacks even if the adversary is not in the vicinity of the device [16]. Furthermore, hijacked devices controlled by botnet herders, can be abused to flood users with unwanted packets (as in the case of the Mirai Botnet).
6) **Excessive data collection:** Devices collect more data than they need and at much finer granularity. Furthermore, data sharing models are static and cannot be updated given different contexts (for instance a user walking should share location data with a navigation app less often as opposed to a user driving a car) [17].
7) **Inflexible data sharing:** Similarly, when communicating with each other, devices often share data openly under rigid sharing schemes (typically all-or-none).
8) **Data Leakage:** Devices have poorly "sealed" storage banks and leaky APIs, which can allow attackers to exfiltrate data from the device. On the cloud side, there are vulnerabilities in the communication models which result in data leakage and repudiation on part of the data subscribers [5].

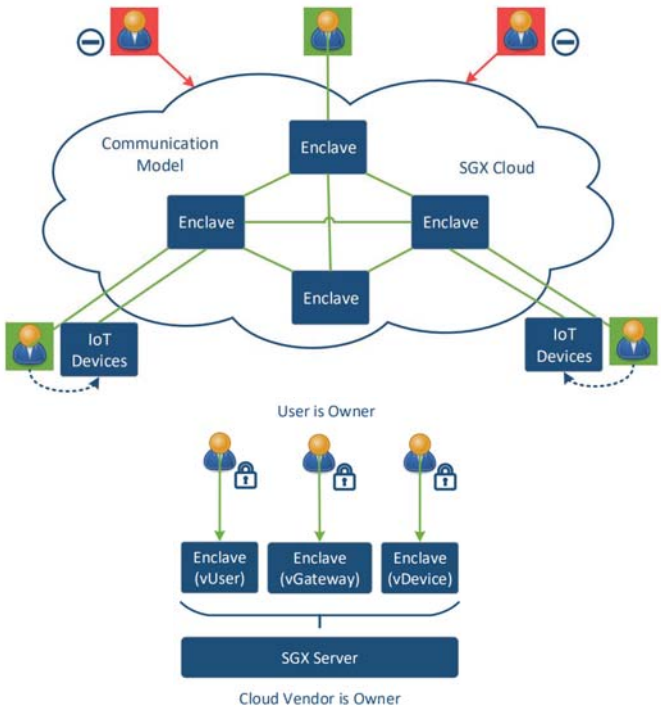

Fig. 1. SGX-Cloud: Each enclave is owned by a particular user and can either be a virtual user, gateway, device or simply a dashboard for a collection of physical devices. The communication model only permits communication between enclave owners and other enclaves by default. The users manage their IoT devices via the enclave.

There are a host of other issues however, in the interest of space we limit the discussion to the ones mentioned above.

III. System Design

Our system relies on Intel SGX to provide the basic root of trust. However, SGX is simply a building block and needs to be modeled onto the cloud as a service for our purposes and hence, we discuss the notion of an SGX-Cloud. As the name suggests, the provider would create SGX-instances or enclaves for the users. The host machine would be owned by the cloud service provider (CSP) but each SGX enclave would be temporarily "rented out" to a particular user who can cryptographically attest the software running inside. This would allow useful security properties in the system, such as confidentiality and integrity of user code and data.

Furthermore, each enclave would only be allowed to talk to the enclave owner, or to other enclaves running in the cloud creating a closed ecosystem of enclaves with well-defined entry points allowing for a very small attack surface (**Problem 2**). Of course, enclave owners can configure enclaves to talk to the outside world as well.

The communication model and working of the SGX-Cloud is shown in Figure 1. Each enclave instance is bound to a

corresponding physical entity e.g., a user or company (virtual user), or a companion device like a cell phone or hub (virtual gateway) or an IoT device itself (virtual device). The potentially unbounded resources on these virtual entities prevents resource exhaustion attacks (**Problem 5**). The IoT device itself should ideally have minimal code and functionality but the clone running inside the cloud can support a whole host of additional functionalities. The physical IoT device would gather the data and perform minimal computations on it (if necessary) and ship the data to its corresponding clone in the cloud by first authenticating the SGX-based clone and then encrypting all traffic in the clone's public SGX key (or exchange a session key). Once the data arrives to the clone, it is processed and saved as per the directives of the enclave owner (putting users in charge of their own data). Any party interested in data generated by a particular IoT device, would send a request to its corresponding SGX clone. However, the requesting party has to send this request using its own SGX enclave to enforce accountability and abide by the constraints of the communication model. At this point, the user's enclave would invoke any access control policies specified by the user. Furthermore, before the data is handed over to the SGX enclave of the requesting party, it would be passed through a scrubbing service to make sure that data is shared in the form explicitly specified by the user (**Problem 6**). Once all processing is done, data is sent to the requesting SGX using its public key from Intel, meaning that this data can only be decrypted inside an enclave on a particular receiving machine tied to a specific organization or company (**Problem 8**). If the requesting party does not own an SGX enclave, the user can still pass on the data to them, however this would weaken the security properties of the system. Data generated from the IoT devices, if needed, can be cryptographically sealed and stored directly on the cloud protecting against leaky storage as would be the case on an actual device (**Problem 4**). This also gives user full control of their data and store/share it as they deem appropriate. Finally, inter-IoT device communication is only done via the enclaves of the communicating devices, which removes the need for an implicit trust assumption between the same set of devices and is carried out using strong encryption schemes after data is scrubbed (**Problem 1, 3, 7**).

IV. Implementation

For the SGX-Cloud we used Dell Precision Tower 3620 with 32GB RAM running Ubuntu 16.04.2 (Linux Kernel 4.4.0). Given that SGX enclaves are extremely limited in what can be run inside them because of I/O and system call restrictions, we used the Graphene Library OS [18] to provide a basic runtime environment. Graphene has already been ported to run inside SGX encalves and provides support for functionality typically needed by Linux applications.

As a proof-of-concept access control system, we implemented a traffic server in Python (2.7.12) inside the Graphene environment. This traffic server allowed us to control sharing of data in the granularity specified by the user and acted us a virtual switch to enforce the constraints of the inter-SGX communication model. Furthermore, we also wrote a few advanced access control and data scrubbing modules in Python (e.g., to change granularity of accelerometer readings or hiding faces from video data or modifying audio frequency to prevent voice profiling on digital assistants like Alexa), to demonstrate how advanced data processing happens on the virtual clones. The traffic server and the aforementioned modules were all set up in different enclaves and constituted around 800 lines of code. Inter-enclave communication was realized using traditional network protocols (TCP/IP) however, we implemented an end-to-end encrypted pipeline to prevent any snooping and data leakage. We also developed a RESTful application using HTML and Javascript with a user-friendly interface to help users specify access control or scrubbing policies. Once a policy is selected, they are forwarded to a Python-based web server (also housed in an enclave), which parses the policies and forwards them to the traffic server in a format that it understands. The traffic server then intercepts the data streams arriving at the enclave-resident clone and applies policies accordingly and forwards the results to the clone. The components of our framework were geographically spaced apart (different continents) to mimic realistic performance overheads incurred by actual users of the service. Some results are shown in the subsequent section.

To mimic a physical IoT device communicating with its virtual clone, we set up two Graphene instances, one running natively (representing a physical device) and the other one inside an SGX (representing a clone) with a Python-based daemon for syncing the state between the clone and native instance. We chose Graphene for this primarily due to the fact that it is a library OS and very similar to custom firmware typically found in IoT devices such as Contiki OS [19] and hence, it is a decent approximation.

V. Evaluation

Since this proposal is still a work-in-progress, we discuss some rudimentary results below focusing on data sharing and privacy. Results pertaining to security have been omitted for brevity as they were quite intuitive, such as improved authentication, resilience to denial of service attacks or prevention against device hijacking.

A. Quantifying the Latency Overhead

The overall latency function of the architecture is characterized by the network/communication model as well as the nature and depth of data processing (such as the number and type of ACLs and scrubbing techniques deployed). To explore these relationships in more detail, we first look at the delay induced by the number of access control filters and scrubbing policies on incoming telemetry data from an IoT device (such as GPS coordinates or gyroscope readings). Figure 2 shows how increasing the numbers of rules (making the data sharing and processing policy more fine-grained) at the traffic server increases the delay. We started with one rule and increased all the way to 10k rules, where each rule was chosen at random (such as an ACL or a rule designed to control data

Fig. 2. Overhead due to increasing the number of packet level rules.

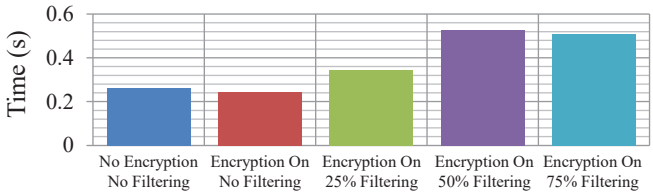

Fig. 3. Time taken for 100,000 packets to be sent to our service and received back, for different filtering scenarios. The first bar represents pure network overhead incurred by routing and link traversal without any encryption.

sharing or processing to requesting applications). As apparent from the figure, the overhead is not related to the number of rules applied on the data stream. This implies that a user can have multiple rules and policies to govern data sharing and processing without substantially impacting the latency overhead.

Another interesting aspect to explore is how the granularity of the control mechanisms impact delay. For instance, given a data stream how does the filtering intensity (such as mild vs. agressive) impact latency. To explore this issue, we measure the overhead introduced by our system in different scenarios including no filtering, 25%, 50% and 75% filtering (percentage of packets dropped from the stream). To this end, we send a total of 100k packets to the SGX enclaves and wait for the packets to be received back for each filtering scenario and repeat the entire experiment 5 times. As shown in Figure 3, the time taken for 100,000 packets (encrypted end-to-end) to arrive at the server, pass through the filtering policies, get decrypted and passed onto the application and then get encrypted again and return to the originating device (to cater to the scenario where the app returns some sort of response) does not depend on the degree of filtering intensity. This makes sense intuitively as well since packet filtering and routing is done at line speed and does not introduce noticeable overhead. For the same reason, additional hops on the path do not add any significant latency. We repeat the same experiment without any encryption involved to measure what part of the overhead is incurred due to encryption and what part is due to the networking operations. The blue colored bar shows the time taken in the former case. As evident, the bulk of the overhead is due to routing and link traversal, which devices will have to incur even without our service if they share data over the Internet. The part of the overhead introduced by our service is therefore, tolerable.

VI. Conclusion

IoT devices are radically refactoring the way we perceive our lives and have become commonplace. However, this spread has also led to the emergence of serious security and privacy challenges for IoT devices that need to be addressed urgently. We propose that IoT devices and gateways should be be thin clients with limited functionality. The core functionality should reside in secure and trusted domains in the cloud achieved via Intel SGX enclaves offered as a cloud service. We define a strict communication paradigm for the SGX cloud, which prevents a broad range of attacks on the infrastructure. Our design addresses numerous security and privacy concerns typically associated with common IoT devices by superimposing the strengths of hardware-centric secure clouds and foolproof mandatory access control frameworks on to IoT devices.

References

[1] T. Denning, T. Kohno, and H. M. Levy, "Computer security and the modern home," *Commun. ACM*, vol. 56, no. 1, pp. 94–103, 2013.

[2] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart locks: Lessons for securing commodity internet of things devices," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, 2016.

[3] *Watch out, new parents internet connected baby monitors are easy to hack*, https://tinyurl.com/yby6t6k2.

[4] "The Botnet That Broke the Internet Isn't Going Away," September 2016, https://tinyurl.com/h6uuzcm.

[5] "Internet of things research study," 2015, http://www8.hp.com/h20195/V2/GetPDF.aspx/4AA5-4759ENW.pdf.

[6] "House of Keys: Industry-Wide HTTPS Certificate and SSH Key Reuse Endangers Millions of Devices Worldwide," November 2015, http://blog.sec-consult.com/2015/11/house-of-keys-industry-wide-https.html.

[7] J. Radcliffe, "Hacking medical devices for fun and insulin: Breaking the human scada system." *Black Hat Conference presentation slides. Vol. 2011*, 2011.

[8] Ray Manpreet Singh Matharu, "Personal Data Management in the Internet of Things," thesis, University of Waterloo, Waterloo, Ontario, Canada, 2015.

[9] "The IoT threat to privacy," August 2016, https://techcrunch.com/2016/08/14/the-iot-threat-to-privacy/.

[10] "Complaint Letter by Electronic Privacy Information Center," July 2015, https://epic.org/privacy/internet/ftc/EPIC-Letter-FTC-AG-Always-On.pdf.

[11] "Maker of 'Smart' Vibrators Settles Data Collection Lawsuit for $3.75 Million," March 2017, https://www.nytimes.com/2017/03/14/technology/we-vibe-vibrator-lawsuit-spying.html.

[12] "SmartThings," June 2017, https://www.smartthings.com/.

[13] "Microsoft Connected Vehicle Platform," June 2017, https://www.microsoft.com/en-us/internet-of-things/connected-vehicles.

[14] "Intel Software Guard Extensions," June 2016, https://software.intel.com/en-us/sgx.

[15] D. He, M. Naveed, C. A. Gunter, and K. Nahrstedt, "Security concerns in android mhealth apps," in *AMIA Annual Symposium Proceedings*, vol. 2014. American Medical Informatics Association, 2014, p. 645.

[16] V. Desnitsky and I. Kotenko, "Modeling and analysis of iot energy resource exhaustion attacks," in *International Symposium on Intelligent and Distributed Computing*. Springer, 2017, pp. 263–270.

[17] "How to protect your privacy as more apps harvest your data," May, https://tinyurl.com/lw777qr.

[18] D. C. A. Bulterman, H. Bos, A. I. T. Rowstron, and P. Druschel, Eds., *Ninth Eurosys Conference 2014, EuroSys 2014, Amsterdam, The Netherlands, April 13-16, 2014*. ACM, 2014.

[19] "Contiki: The Open Source OS for the Internet of Things," June 2016, http://www.contiki-os.org/.