Your IoTs Are (Not) Mine: On the Remote Binding Between IoT Devices and Users

Jiongyi Chen[*], Chaoshun Zuo[†], Wenrui Diao[‡§], Shuaike Dong[*], Qingchuan Zhao[†], Menghan Sun[*], Zhiqiang Lin[†], Yinqian Zhang[†], and Kehuan Zhang[*]

[*]The Chinese University of Hong Kong
[†]The Ohio State University
[‡]Shandong University
[§]Jinan University

***Abstract*—Nowadays, IoT clouds are increasingly deployed to facilitate users to manage and control their IoT devices. Unlike the traditional cloud services with communication between a client and a server, IoT cloud architectures involve three parties: the IoT device, the user, and the cloud. Before a user can remotely access her IoT device, remote communication between them is bootstrapped through the cloud. However, the security implications of such a unique process in IoT are less understood today.**

In this paper, we report the first step towards systematic analyses of IoT remote binding. To better understand the problem, we describe the life cycle of remote binding with a state-machine model which helps us demystify the complexity in various designs and systematically explore the attack surfaces. With the evaluation of 10 real-world remote binding solutions, our study brings to light questionable practices in the designs of authentication and authorization, including inappropriate use of device IDs, weak device authentication, and weak cloud-side access control, as well as the impact of the discovered problems, which could cause sensitive user data leak, persistent denial-of-service, connection disruption, and even stealthy device control.

I. Introduction

One of the fastest growing industries today is the Internet-of-Things (IoT), which connects the smart computing devices embedded in our daily lives and allows them to be sensed and controlled remotely via the Internet. Many applications with IoT have been developed over the past a few years, ranging from smart homes, smart health, to smart cities and beyond. Numerous benefits can be gained by using the IoT, such as improved efficiency and accuracy, and reduced human intervention.

However, not all IoT devices are connected to the Internet directly. To enable their remote management, typically there is a cloud service acting as a relay between the end user and the device. For instance, in a smart home solution, even when a user is not at home, she can still remotely operate the IoT devices using the corresponding mobile apps installed on her phone. Such convenience is made possible by the IoT cloud. Unlike the traditional server/client communication architecture, a typical IoT system involves three parties: the IoT device, the end user (or the mobile app as the user agent) and the IoT cloud. However, it is unclear today whether the introduction of a third party (i.e., the IoT device) comes with new security challenges, especially when bootstrapping and removing remote communication between the user and the device through the IoT cloud.

The bulk of work was performed while the first author was visiting The Ohio State University in spring and summer of 2018.

Remote binding of IoT. The remote communication bootstrapping process is also known as *remote binding*. In general, there are four steps in the life cycle of remote binding: (1) at first, the user and the device are authenticated to the cloud respectively; (2) Then, the user and the device need to bind with each other on the local network. After local binding, a device-specific secret such as the device ID is delivered to the user; (3) Next, they both talk to the cloud, and the user submits such a device ID to the cloud to create a binding with the specified device. At this moment, the user can remotely control her device; (4) Later on, when the user resets her device, the binding in the cloud will be revoked. Therefore, to support those operations, the cloud needs to authenticate the device and the user, and correctly assign the binding permissions to the user.

Our study and findings. Given the complexity of remote binding, we need a systematic methodology to decompose existing designs into primitives so that the security risk can be analyzed and understood clearly. To this end, we model and describe the functional design using a state-machine model, in which the procedures of remote binding are represented as cloud-side device state transitions in response to primitives messages sent by the device and the user. Such a process model captures the essential functional demands, which helps us demystify various design principles.

With the assistance of our state-machine model, we were able to inspect the remote binding designs of 10 IoT vendors and systematically evaluate their security risks. Our study reveals that, for most of the devices, security measures are either nonexistent or incorrectly designed and implemented. As such, remote attacks can be realistically orchestrated by abusing device IDs and exploiting flawed authorization, which allows the attacker to exfiltrate sensitive information remotely, launch denial-of-service to the user's binding, disrupt the user's connection, or even take absolute control of the device.

In fact, we found that such a threat is completely realistic, as the user's device ID could be easily leaked in practice: on the one hand, some vendors simply use weak device IDs, such as MAC addresses, allowing attackers to enumerate or brute-force them within a small search space (with vendor-specific bytes excluded, the search space of MAC addresses is often within 3 bytes). Even worse, recent attacks indicate that some device IDs only contain 6 or 7 digits [14], [18], allowing attackers to traverse all possible IDs within an hour. On the other hand, the IoT device is a "thing" in nature and the device ID could be leaked through ownership transfer in real life, for example, during shipping, distribution, or redistribution in a supply chain.

Today, security risks in the IoT remote binding are not well understood, and more secure practices are needed in IoT development. As such, we highlight two significant misunderstandings in our study: first, static identifiers should never be used for device authentication. Instead, a better approach is to adopt dynamic authentication tokens. Second, proper authorization mechanisms should be used in the remote binding and unbinding operations, to confirm a user's ownership to the device.

Contributions. We summarize this paper's contributions as below:

1) *New findings and understandings.* We conducted the first systematic study on the security of IoT remote binding. Notably, we use a state-machine model to decompose the remote binding process, further evaluating various designs and exploring the attack surfaces. Our study reveals multiple design flaws in the mainstream binding solutions, which could cause serious consequences such as user data leak, binding denial-of-service, connection disruption, as well as complete device control.
2) *Real-world case studies.* To confirm the potential design and implementation flaws, we studied ten real-world IoT remote binding solutions. The experimental results align with the systematic investigation and demonstrate that the attacks are serious and realistic. With the real-world case studies, our study sheds light on a new class of vulnerabilities and contributes to a better understanding of the rapidly growing IoT area.

Roadmap. The rest of the paper is organized as follows: Section II provides the background information for our study. Section III describes the adversary model, the investigation, and the state-machine model. Section IV elaborates on the existing designs of remote binding. Section V presents the attack surfaces in remote binding. Section VI gives the results of the real-world attacks. Section VII summarizes the lessons learned in our study. Section VIII details the limitations and future directions of this work. Section IX surveys the prior related research, and Section X concludes this paper.

II. BACKGROUND

In this section, we give the necessary knowledge of the typical IoT communication architecture and the procedures of the remote binding.

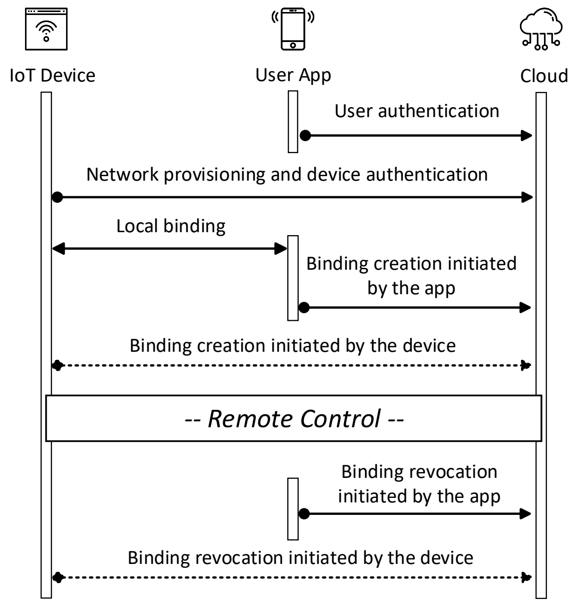

IoT Device
User App
Cloud
User authentication
Network provisioning and device authentication
Local binding
Binding creation initiated by the app
Binding creation initiated by the device
-- Remote Control --
Binding revocation initiated by the app
Binding revocation initiated by the device

Fig. 1: Procedures of remote binding

A. IoT Communication Architecture

The communication in IoT environments usually involves three parties: the IoT device, the user (or the mobile app as the user agent), and the cloud. Each of them takes different responsibilities:

- *The IoT device* acts as the information collector. Oftentimes, multiple devices can be deployed and work collaboratively in a smart home to monitor the environment status.
- *The IoT app* is developed by vendors to facilitate user operations. It is used to interpret or visualize sensor data of IoT devices and interact with users.
- As a connector between the device and the user, the *IoT cloud* enables the user and IoT device to remotely communicate with each other.

To connect the user with the device, there are two connection modes in IoT systems: local connection and remote connection:

- *Local connection* allows the user and the device to communicate within local networks, where home routers are typically used as local delegations to relay messages between the user and the device.
- In *remote connection*, the user's phone is not in the same LAN with the IoT devices. Therefore, the cloud is needed to relay messages between the user and the device.

B. Procedures of Remote Binding

Here we consider the entire life cycle of remote binding, involving user authentication, local configuration (i.e., device authentication and local binding), binding creation, and binding revocation. In particular, as Figure 1 shows, user first logs in and authenticates herself to the cloud. Then, she needs to configure the device to associate with her mobile app and access the local network. Next, both the phone and the IoT device communicate with the cloud to create a binding relationship in the cloud for subsequent remote connection. Finally, if the user resets the device, the binding in the cloud should also be revoked. To get an idea, we describe the typical design of remote binding as below:

- *User authentication.* IoT vendors usually deploy password-based schemes to authenticate users [52]. Specifically, this involves two steps: first, the user logs in the cloud; then the cloud returns a user token as the credential for subsequent steps.
- *Local configuration.* In this step, the IoT device is set up to access the LAN (Wi-Fi) and authenticate to the cloud. In the meantime, the device is also configured to pair with the user's IoT app.
  - *Network provisioning.* To provide network connection for IoT devices, network cable-based devices can directly connect with the home router using the DHCP protocol. For wireless devices, there are some well-known techniques, such as SmartConfig [13] and Airkiss [16], that can facilitate the operation.
  - *Device authentication.* Once the device can access the network, it is authenticated to the cloud by sending the authentication token that contains its device information. Meanwhile, it also reports the device status and attributes, such as the firmware version and the model name, to the cloud.
  - *Local binding.* When both the app and the device are connected to the same local network, they need to discover and associate with each other. In some solutions, service discovery protocols like Simple Service Discovery Protocol (SSDP) [12] are used to broadcast self-descriptions and exchange information between the device and the app. Alternatively, some vendors attach labels containing device information (e.g., Device IDs or pairing IDs) on devices, and ask users to input such IDs in their apps. When the app obtains the device information, the app will broadcast messages containing such information to locally bind with the device.
- *Binding creation.* Since the cloud relays the messages between a specific device and a specific user, a binding relationship of the device and the user should be maintained in the cloud. As such, a binding message that contains the device information and the user information (such as the user token) will be sent to the cloud by the app, or alternatively by the device (notated with dashed arrows in Figure 1). After the binding is created, the app and the device can remotely communicate with each other.
- *Binding revocation.* When the user resets a device or deletes the device in the app, the binding should be revoked in the cloud. In this case, to notify the cloud, an unbinding message should be sent by the app or the device (notated with dashed arrows in Figure 1).

Given that there are already mature solutions for user authentication and local binding, in the following sections, we focus on the security threats in device authentication, binding creation, and binding revocation.

III. Preliminaries

In this section, we first describe the adversary model. Then, we present the state-machine model that helps systematically decompose the remote binding designs and analyze security vulnerabilities.

A. Adversary Model

In this paper, the adversary aims to launch targeted attacks exploiting the remote binding between a device and a user. We assume that the adversary can obtain the device IDs, due to the weak protection in real life:

- *Inference of the device ID.* Attackers may infer, brute-force, or enumerate the device ID according to the regulation of ID sequence arrangement. For example, the device MAC addresses (as device IDs) contain a vendor-specific field, which only leaves small search space.
- *Off-site physical interaction of the device.* The IDs may be leaked through device ownership transfer, including device reuse, reselling, stealing, and so forth. For example, the attacker could purchase an IoT device from Amazon, record the device ID, and return it. What is worse, some IoT vendors attach the ID labels on devices or the packages (e.g., [5], [15], [19]) to facilitate local configuration. It also brings the risks that untrusted supply chain participants may copy device IDs during products transportation or distribution.

In practice, given that IoT devices are usually connected in local networks that are protected by firewalls or encryption like WPA2 [20], the bar of local attacks is exceptionally high. Therefore, unlike prior studies [42], [43], [58], we assume the adversary cannot access user's local networks. Additionally, we assume the device firmware and IoT apps are not compromised.

B. Decomposition

Before we analyze the security threat, we need to clearly understand how the remote binding functionality is built up. As such, we take a top-down approach to decompose the remote binding functionality into primitives, and then systematically analyze the implementation and design choices.

Top-down investigation. The ultimate goal of remote binding is to achieve remote communication between a specific user and a specific device. As the basic requirement for secure communication, the cloud needs to authenticate both the user and the device. In addition, the cloud should also maintain the binding relationship between the authenticated user and the authenticated device. Therefore, in order to achieve remote communication with a device, the cloud needs to confirm two kinds of status for a device[1]: whether the device is online (or

[1]In the following sections, we use the term "device shadow" [17] to represent the status of the device in the cloud.

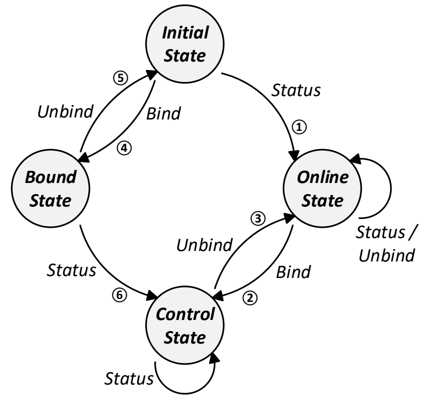


Fig. 2: State machine of a device shadow: ① and ⑥ represent device authentication; ② and ④ represent binding creation; ③ and ⑤ represent binding revocation.

logged in) and whether the device is bound[2]. In particular, a device shadow is online if the cloud has authenticated the real device and received messages from the device. The device shadow is bound if a binding of the device has been created. Additionally, although a device might be bound with several users (i.e. device sharing) and a user can also manage several devices, in this paper we only focus on how a binding of one user and one device is established and revoked, which can be easily applied to many-to-one (or one-to-many) bindings.

To investigate whether the above design demands of IoT remote binding are properly designed and implemented, we model the remote binding functionality using the state machine of the device shadow, whose state transitions represent the completion of procedures and are changed when receiving primitive messages. With its help, we then systematically analyze and discuss the designs in each procedure.

State-machine model. At a high level, the state-machine model consists of four states and receives three types of primitive messages (i.e. atomic actions) to achieve transitions. As we discussed earlier, the cloud maintains two kinds of status for device shadow during remote binding: whether the device is online and whether the device is bound. Therefore, there are four states for a device shadow:

- *Initial state.* The device in this state is offline and unbound (not bound with any users). This is the initial state of the device shadow.
- *Online state.* The device in this state is online and unbound. In this state, the device has been authenticated to the cloud but not yet bound with any users. This state is maintained when the device sends registration messages or heartbeat messages to the cloud. The device shadow goes into this state, for example, before device binding or after device unbinding.

[2]The binding status of a device is the same as the binding status of the bound user. Also, the cloud does not need to maintain the online status of a user because the user is assumed online during the entire remote binding process.

TABLE I: Notations in Section IV, Section V, and Section VI

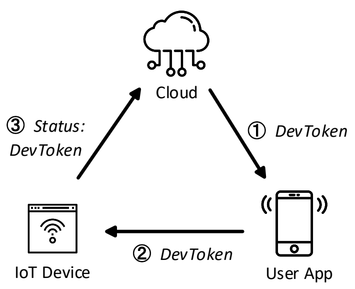
$Status$	Messages to report device status (sent by the device)
$Bind$	Messages to creating bindings in the cloud
$Unbind$	Messages to revoke bindings in the cloud
$DevId$	A piece of definite data for device authentication
$DevToken$	A piece of random data for device authentication
$BindToken$	A piece of random data for the authorization in binding creation
$UserToken$	A piece of random data for user authentication
$UserId$	Identifier (e.g. email address) of user account
$UserPw$	Password of user account

- *Control state.* The device in this state is online and bound. After device setup, the device is authenticated to the cloud and bound with the user. This is the only state that allows the user to control the device.
- *Bound state.* In this state, the device is offline and bound. The device shadow goes into this state (1) when the real device is powered off or the network connection is disrupted. However, the binding relationship is still maintained in the cloud; (2) Or when the binding is created in the cloud, but the device is not online yet.

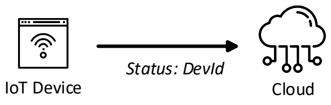
As can be seen in Figure 2, to achieve remote communication, a device shadow changes from the initial state to the control state. This means a binding can be created before the device authentication (initial state $\rightarrow$ bound state $\rightarrow$ control state) or after the device authentication (initial state $\rightarrow$ online state $\rightarrow$ control state). To achieve state transitions, the cloud receives three types of messages from the user or the device: status messages, binding messages, and unbinding messages. Below we describe the functions of them in details:

- $Status$: *status message*. Status messages could either be the registration message or the heartbeat message. In our process model, they share the same functionality: they change the online/offline state of a device shadow. The reception of such a message in the cloud indicates the online status of the real device (i.e. device authentication). If the message is not received within a certain time period, the device is considered offline. Although the fields of a registration message might be different from that of a heartbeat message, in our model they still achieve the same state transitions. Besides, this message is only sent from the device.
- $Bind$: *binding message*. In the message, it specifies which user is bound with which device. A binding is created in the cloud when the cloud receives such a message. This message can be sent from a user or a device with both the device identity and the user identity.
- $Unbind$: *unbinding message*. This message revokes an existing binding of a user and a device in the cloud. Besides, it can also be sent from a user or a device.

Note that except for the above three types of messages, there are other messages, such as control messages sent by the user. However, we do not consider them in this paper, as they do not change the states in binding.



(a) Device authentication using device tokens



(b) Device authentication using device IDs

Fig. 3: Device Authentication

IV. Existing Designs

In order to investigate and evaluate the remote binding designs of IoT vendors, we selected 10 representative IoT device pairs[3] that rank top on Amazon. They are the best-selling products offered by mainstream manufacturers from China and the U.S. In addition to those designs, we also refer to the solutions of remote binding from IoT solution providers, such as AWS, IBM, Google, and Samsung. In this section, we elaborate on the existing designs of remote binding and discuss potential misunderstandings.

A. Device Authentication

Device authentication is to verify the identity of a device in the cloud. As shown in Figure 3, there are two kinds of authentication modes based on the usage of different identifiers: device tokens ($DevToken$) and device IDs ($DevId$) in the status messages (notations are shown in Table I).

- *Type 1: $Status$: $DevToken$.* The IoT app requests a device token from the cloud and then delivers it to the IoT device during local configuration. After that, the device sends the token to the cloud for authentication. Given that the user app needs to locally negotiate with the device anyway before the device is put into use, using a user app for device authentication does not bring extra complexity. Among the devices that we evaluated, at least three of them (see Table III) use the device token mechanism. The tokens are put in the encrypted status messages directly.
- *Type 2: $Status$: $DevId$.* Also, some vendors assign a unique device ID to each of their devices, and such an ID will be used for device authentication (at least 4 of our evaluated devices use this design). The ID can be a device MAC address [10] or a device serial number [14], [18]. This design is actually a user-friendly feature: if the user app keeps such an ID, device binding can be completed even if the device and the mobile app are not on the same network [42]. Unfortunately, in this case, a device is under risk if its device ID is leaked. For instance, the attacker can report fake device data to the victim or receive sensitive information of the user, by forging the device status messages. Besides, the attacker may hijack a victim's device by exploiting the implementation flaws of the cloud (see Section V).

Apart from the above designs, there are some public-key-based authentication solutions specified by most IoT infrastructure providers such as AWS IoT [4], IBM Watson IoT [8], and Google Cloud IoT [7]. In their solutions, a key pair is generated during manufacturing. The public key is stored in the cloud, and the private key is embedded in the device. Although this allows the cloud to authenticate each message sent by the device securely, such a scheme is rarely used in commercial IoT products. The main reason is that it requires hardware support, like TPM, to protect the secret keys, which increases the cost and affects the execution efficiency. On the other hand, currently, those cloud service providers only have basic infrastructures for individual developers who manage a specific device and a specific app[4]. However, this is less suitable for IoT vendors who manage a bunch of devices and a bunch of registered users.

Our assessment: The use of static identifiers (i.e., $DevId$) for authentication will inevitably introduce security risks, although such implementation could bring some convenience of remote binding. Another observation is that the solutions provided by leading IT companies, such as AWS, IBM, and Google, require the support of trusted hardware, in most cases, which is not suitable for resource-constrained IoT devices. A more promising approach is to use dynamic authentication token (i.e., $devToken$), relying on the user to obtain such a token from the cloud and deliver it to the device via local communication.

B. Binding Creation

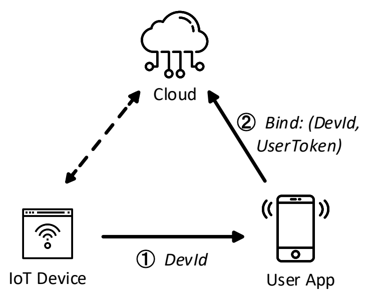
When the cloud receives the binding message, it will create a binding relationship between a device and the corresponding user account. Particularly, there are two types of binding mechanisms: ACL-based binding and capability-based binding.

ACL-based binding. In practice, most of the devices we studied use the device ID and the user token to indicate the relationship in a binding message: $Bind : (DevId, UserToken)$. The binding message can be delivered by the mobile app or the IoT device (see Figure 4):

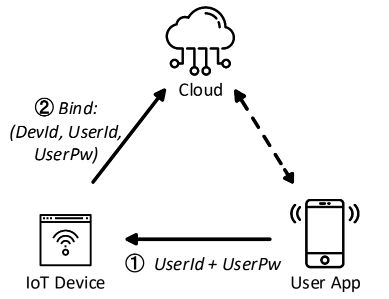
- *App-initiated binding*. The mobile app sends a binding message containing the device ID (obtained from the device) and the user token (obtained from the cloud) to the cloud. After receiving it, the cloud will create a matched binding relationship. Most of our experimental devices belong to this category.

[3]For each device type, we purchased a pair of devices (20 devices in total). For each pair, we assume one device belongs to the victim, and the other one belongs to the attacker.

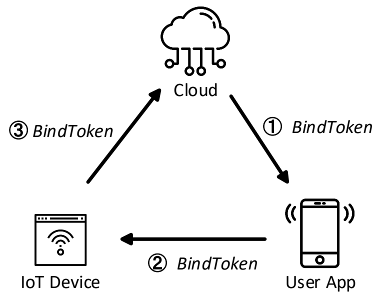
[4]Packages that contain key pairs should be installed on the device and the user app.



(a) ACL-based binding, binding message sent by app



(b) ACL-based binding, binding message sent by device



(c) Capability-based binding

Fig. 4: Binding Creation

- *Device-initiated binding*. For this design, first, the user credential (i.e. the username $UserId$ and the password $UserPw$) is delivered to the device during local configuration. Next, the device submits a binding message that contains both the user credential and the device ID to the cloud. After receiving the message, the cloud creates a binding relationship. However, delivering the user's credential to the device might put the user account in danger if the device is compromised.

Interestingly, in some ACL-based binding designs, we find that after the binding message is submitted to the cloud, the user and the device will execute an extra step for the post-binding authorization, which could prevent device hijacking attacks (see Section V-E). More specifically, when the binding messages are submitted to the cloud, a random token will be returned to both the user and the device by the cloud. In the subsequent interactions, this token will be included in every message of the device and the user app[5]. With such a mechanism, even if an attacker can forge a binding message that indicates the binding between the attacker and the victim's device, hijacking a victim's device is infeasible. The reason is that the attacker is unable to force the target device to submit the same token (as the attacker's). Note that, although the involvement of a random token can prevent the forgery of user messages and device messages in the control state, it cannot prevent the forgery of binding messages.

Capability-based binding. Under our adversary model, a secure binding mechanism should rely on the capability-based authorization, in which an authorization token (in this case, the binding message is: $Bind : BindToken$) is delivered from the cloud to the user app and then locally transmitted to the device [3]. After that, the device submits this token back to the cloud, to confirm the binding with that user. This design could guarantee the user's ownership of the device: to bind with a device remotely, the user must be locally bound with the device.

Our assessment: A significant misunderstanding of binding creation is the use of ACL-based binding, in which vendors combine $devId$ and $UserToken$ to confirm binding relationships. This design could result in the "ambient authority[a]" [1] and open the door to a series of attacks (as demonstrated in Section V). Instead, the best practice is to use capability-based binding, such as the solution of Samsung SmartThings [3]. That is, a binding token $BindToken$ represents the actual authority, and the authorization step is only achieved by locally communicating with that specific device (i.e., ownership confirmation).

[a]The device ID itself does not including authorizing information and could potentially be used by other users.

C. Binding Revocation

A binding is revoked when the device is reset or the user removes the device in her account[6]. We found that there are three kinds of unbinding messages to revoke bindings in the cloud (assuming that device i is bound with user j in the cloud):

- *Type 1:* $Unbind : (DevId_i, UserToken_j)$. To revoke the binding, the user or the device sends an unbinding message with the user token and the device ID to the cloud. When receiving such a message, the cloud first verifies the user token and then revokes the corresponding binding according to the submitted device ID. Besides, the cloud needs to check whether the message sender has been already bound with the device.
- *Type 2:* $Unbind : DevId_i$. Alternatively, an unbinding message can be sent from the IoT device. Since one

[5]For the device, such a token means the device token that we previously discussed.

[6]There could be multiple ways to achieve physical device reset. For example, a message can be sent from the device if the device has been physically reset and connects to the Internet. This message informs the cloud that the previous binding should be revoked.

device only belongs to a specific user, an unbinding message only containing the device identifier can also achieve unbinding function. This approach can bring convenience to the user because the unbinding message can be sent during the device reset and no extra action is needed. Unfortunately, this approach could also bring security risks because anyone obtaining the device ID can forge an unbinding message and revoke the binding.

- *Type 3:* $Bind : (DevId_i, UserToken_n)$. An interesting finding is that there is one device that does not support unbinding operations, and the user has to use new binding to replace the previous one in the cloud. In other words, whenever a binding message is received, the cloud replaces the bound user i with the new user n. This design decreases the development efforts for developers, but inevitably introduces new security risks: the attacker can forge binding messages to replace a user's binding, which could cause device unbinding or device hijacking (see Section V).

Our assessment: Binding revocation is also a critical authorization procedure that is often misunderstood or neglected by developers. Our study concludes that not only should it be correctly designed (e.g., using authorization token to manipulate cloud-side resources), but also the cloud should enforce strict policy to check whether a message sender indeed has the permission to revoke the claimed binding relationship.

V. Security Vulnerabilities

In this section, we analyze the security risks lying in the remote binding. Notably, we show how the design and implementation choices can be abused to launch several attacks, ranging from binding denial-of-service to device hijacking.

A. Overview

We aim to investigate the security risks in remote binding with respect to the procedures that we described earlier. To this end, *we systematically explore potential attack surfaces by considering that all three types of messages could be forged and sent to the cloud in all states of a device shadow*:

- When status messages are forged, the attacker could act as the user's device. As such, fake device data can be injected and user data can be stolen in the control state and the bound state. We call it data injection and stealing attack.
- When binding messages are forged, the attacker could create a binding with the user's device before the user binds with it. This causes binding denial-of-service attacks. Or alternatively, if the user is already bound with her device, the binding could be replaced by the attacker's binding. In this case, it is possible for the attacker to disconnect the user with her device or take control of the user's device.
- When unbinding messages are forged, the attacker could revoke the user's binding. This causes device unbinding to the user. Note that this attack could also be combined with binding message forgery to further hijack the user device.

The attacks can be categorized into four types: data stealing and injection, binding occupation, device unbinding, and device hijacking. We give a taxonomy in Table II and describe them in details in the following.

B. Data Injection and Stealing (A1)

This attack occurs when the user's device shadow is in the control state or the bound state. The attacker can forge status messages with the user's device ID. As a result, data from the "device" cannot be trusted, and the user data could also be leaked to the "device". One consequence is that the attacker can inject the sensor data to the cloud and the user. For example, if the user owns a fire alarm, the attacker can inject fake data to trigger alerts and annoy the user. Even worse, it will have a cascade effect when data from the device is involved in rules (e.g., IFTTT [9]). For instance, when an air conditioning system is associated with a temperature sensor, fake data of the sensor may turn on or turn off the air conditioning system. On the other hand, the attacker can also forge device messages to retrieve private user data from the cloud, for example, when the user sets a schedule for a smart lock, the attacker is able to obtain the opening and closing time of the door.

C. Binding Denial-of-Service (A2)

Binding denial-of-service occurs when the attacker occupies the binding of a user's device before the user binds with the device. The forged binding message consists of the attacker's token and the user's device ID. Once the attack is successfully launched, the user is unable to create the binding with her own device. Unfortunately, given that some vendors use sequential device IDs for its products, attackers can enumerate or brute-force the device IDs, and it could even cause scalable denial-of-service attacks to the entire product series of a vendor [14], [18].

D. Device Unbinding (A3)

The attacker can also leverage unbinding messages, binding messages or status messages to disconnect the user with the user's device. Specifically, For unbinding message forgery, the message $Unbind : DevId$ with the user's device ID can be utilized to revoke the user's binding in the cloud (A3-1). For unbinding message type $Unbind : (DevId, UserToken)$, the attacker will succeed only if the cloud does not check whether the submitted user token is the bound user (A3-2). On the other hand, binding messages can also be leveraged to cause device unbinding (A3-3). In this case, the attacker sends a binding message to replace the user's binding if the cloud does not check whether the device is already bound with a user. Therefore, after the attacker's binding is created, the device will disconnect with the user. In addition to that, forgery of status messages can also cause device unbinding (A3-4). In this case, the cloud takes the attacker as a new device and

TABLE II: The Taxonomy of Attacks in Remote Binding

Attacks		**Forged message types**	**Targeted states**	**End states**	**Consequences**
A1: Data injection and stealing		$Status : DevId$	*Control state* and *bound state*	*Control state*	The attacker can inject fake device data or steal private user data.
A2: Binding denial-of-service		$Bind : (DevId, UserToken)$	*Initial state*	*Bound state*	The attacker can cause denial-of-service to the user's binding operation.
A3: Device unbinding	A3-1	$Unbind : DevId$	*Control state*	*Online state*	The attacker can disconnect the device with the user.
	A3-2	$Unbind : (DevId, UserToken)$			
	A3-3	$Bind : (DevId, UserToken)$			
	A3-4	$Status : DevId$			
A4: Device hijacking	A4-1	$Bind : (DevId, UserToken)$	*Control state*	*Control state*	The attacker can take absolute control of the device.
	A4-2		*Online state*	*Control state*	
	A4-3	① $Unbind : DevId$ or $(DevId, UserToken)$ ② $Bind : (DevId, UserToken)$	*Control state*	*Control state*	

disconnect with the real device. Such attacks can cause denial-of-service to the user's operation of her devices, which may lead to serious consequences. For example, a user's property may be put in danger if a smart lock or a fire alarm stop reporting status to the user.

E. Device Hijacking (A4)

For this attack, the attacker can take absolute control of the device. In particular, there are three ways to achieve device hijacking. On the one hand, the attacker can send a binding message with the attacker's token and the user's device ID to the cloud. In this case, we consider the user's device shadow can be in two states: the control state and the online state. When the user is in the control state, the attack occurs (A4-1) if the cloud does not check the message sender and the bound user. This could either be an implementation flaw in which the cloud directly manipulate the existing binding without checks or a design flaw of unbinding in which a previous user's binding is designed to be replaced by a new binding. Note that such an attack happens in the entire control state, instead of the perfect timing of binding operations. When the user is in the online state, the attacker can bind with the user's device before the user does, by exploiting the time window during user's device setup (A4-2).

On the other hand, device hijacking can also be achieved (A4-3) by combining two vulnerabilities. In particular, the attacker first sends an unbinding message to disconnect the user and the device (A3-1). This turns the device into the online state. Next, the attacker can send a binding message to bind with the device and take control of it (A4-2). Note that above device hijacking attacks will fail if device tokens are used for device authentication. Because the device cannot authenticate to the cloud without receiving a device token from the attacker.

VI. Experimental Results

In this section, we first describe our experiment setup to evaluate the devices and perform our attacks. Then we give our evaluation results of the attacks. In the end, we further clarify the misunderstandings and discuss why existing designs fail.

A. Experiment Procedures

In our experiments, the user and the attacker have different network access (the attacker's access point is set up on an Ubuntu host machine with 8G RAM and Intel Core i7 2.81 GHz), different Android smartphones (Samsung Galaxy S5, Android 5.0) and different accounts. We installed the companion apps of the devices in both smartphones and logged in the apps with the user's account and the attacker's account respectively. Then we setup the devices and configure them with the corresponding apps.

As the first step, we need to identify device IDs of users' devices. Among the 10 devices that we studied, 6 of them directly attach the device IDs on the devices. 5 of them use MAC addresses (the first 3-bytes are ID number of the manufacturer) as their device IDs. For the rest, device IDs can be observed from the traffic or be easily obtained with a differential analysis of the messages.

Next, to launch the attacks, we substitute the user's device IDs with the attacker's device IDs in the targeted messages. To this end, we first identify the binding and unbinding messages through manual dynamic analysis of the apps (9 devices send binding messages by apps). To capture and analyze the HTTP/HTTPS messages from the attacker's app, we use a Man-in-the-Middle proxy [6]. Then we generate fake requests using the tool Postman [11]. For unknown protocols, we use the dynamic instrumentation tool Frida [2] to intercept and modify the source requests generated in the app. On the other hand, to forge device messages, we need to perform firmware reverse engineering. As we all know that firmware analysis is not easy [21], [27], [57] and not always feasible, we were only able to forge device messages for 3 devices (with firmware downloaded from the official websites). We either performed dynamic analysis to emulate the firmware images (for 1 of them) or perform static analysis to identify and manually craft device messages out of the box (for 2 of them).

B. Results

Table III shows the designs of the devices and the results of our attacks. As can be seen, at least 4 of the devices use device

TABLE III: Evaluation Results on Experimental Devices

Vendors	Device Types	Designs			Attacks			
		$Status$	$Bind:$ $(DevId, UserToken)$	$Unbind$	A1	A2	A3	A4
#1: Belkin	Smart Plug	$DevToken$	Sent by the app	$(DevId, UserToken)$	✗	✓	A3-2	✗
#2: BroadLink	Smart Plug	O	Sent by the app	$(DevId, UserToken)$	O	✓	✗	✗
#3: KONKE	Smart Socket	$DevToken$	Sent by the app	N.A.	✗	✗	A3-3	✗
#4: Lightstory	Smart Plug	$DevToken$	Sent by the app	$(DevId, UserToken)$	✗	✓	✗	✗
#5: Orvibo	Smart Plug	O	Sent by the app	$(DevId, UserToken)$	O	✓	A3-2	✗
#6: OZWI	IP Camera	$DevId$	Sent by the app	$(DevId, UserToken)$	O	✓	✗	A4-2
#7: Philips Hue	Smart Bulb	O	Sent by the app	$(DevId, UserToken)$	O	✗	✗	✗
#8: TP-LINK	Smart Bulb	$DevId$	Sent by the device	$(DevId, UserToken)$ & $DevId$	✗	✗	A3-1 & A3-4	A4-3
#9: E-Link Smart	IP Camera	$DevId$	Sent by the app	$(DevId, UserToken)$	O	✗	✗	A4-1
#10: D-LINK	Smart Plug	$DevId$	Sent by the app	$(DevId, UserToken)$	✓	✓	✗	✗

✓: attack successfully launched; ✗: attack failed to launch; O: Unable to confirm due to firmware challenges; N.A.: Not Applicable.

IDs for device authentication[7]. A favorite design is to send messages by the app, while we still found one exception that sends binding messages by the device. Most devices (90%) support message type $Unbind : (DevId, UserToken)$ sent by the app to revoke bindings. For device #3 that does not support binding revocation, we found out that it actually uses binding operations of a new user to replace a previous user's binding.

On the whole, our attacks were successfully launched on 9 devices. In particular, data injection and stealing (A1) was launched on device #10. We reverse engineered its firmware and found the device messages. Next, we forged such device messages by reconstructing the messages and establishing an OpenSSL socket connection with the cloud. To simulate data injection attacks, we forged messages that report fake power consumption to the user. For data stealing attacks, we setup a schedule on the app to turn on and turn off the smart plug. And the attacker could then successfully receive the response of the schedule from the cloud.

There are 6 devices suffering from binding denial-of-service attacks (A2). For device #7, when the binding is initiated on the user app, it requires the user to press a physical button on the device within 30 seconds. Under the hood, a device registration message is sent when the button is pressed. Then the cloud compares whether the source IP addresses of the device request and the user request are the same [30]. The binding fails if they are not the same. Interestingly, device #3 does not suffer from binding denial-of-service because of its broken unbinding mechanism: any new binding creation manipulates the previous binding.

To our surprise, there are 4 devices suffering from device unbinding attacks (A3). Since device #3 does not support binding revocation, the binding between the user and the device can be replaced by the attacker's new binding. Therefore, it suffers from device unbinding attacks. However, although the attacker can create such a binding, she is not able to hijack the device. Because it uses the device token for device authentication and the attacker cannot send a fresh token to the device. For device #8, we forged its device status messages and this also causes device unbinding with the user. We also forged an unbinding message with type $Unbind : devId$, and this can also successfully unbind the user with the device.

Besides, we successfully launched device hijacking attacks (A4) on 3 devices. Device #9 is hijacked with design flaw A4-1, by simply sending a new binding message to replace the user's binding in the cloud. Device #6 is hijacked when it is in the online state and not bound with any users. For device #8, we first sent an unbinding message to revoke its binding with the user. Then we forged a binding message to bind it with the attacker.

In prior research [43], device hijacking requires the attacker to send a message directly to the device to setup the device communication key. However, our attacks exploit the design flaws and cloud implementation flaws and thus does not require any local network access.

C. Ethics and Responsible Disclosure

We carefully designed our experiment to avoid ethical problems. When we conduct our attacks, we only exploit the devices that we purchased, meaning that we only obtain privacy data of our own devices and exploit the flawed designs that only affect our own devices. Although the device IDs could be sequential, we only substitute the IDs of our own devices and we do not brute-force them in case of affecting other users. Also, we have reported our discoveries and suggestions to the corresponding vendors. At the time of this writing, we have received acknowledgments from some of them, and they have promised to check the details and improve their designs of remote binding. To disclose vulnerability details responsibly, we release the vulnerabilities that have been fixed as case studies on our project website[8].

[7]For device #1, #8 and #10, we reverse engineered their firmware. For device #4, we checked its API document. For device #6 and #9, we successfully launched binding and unbinding attacks. Those attacks indicate that the devices use device ID for authentication.

[8]https://sites.google.com/view/iot-remote-binding

VII. Lessons Learned

In our research, we discovered a series of design failures and implementation flaws in IoT remote binding. In fact, vendors have not yet raised awareness to protect the device IDs in their solutions. For example, they may use fixed, sequential or predictable device IDs. Although they can adopt device IDs with sufficient length and randomness, the leak of device IDs is unavoidable. Since the device is a thing in nature and the ownership can be transferred in reusing or selling. Such loose protection of device IDs in practice actually poses security challenges in IoT remote binding.

More importantly, given the complexity, the security implications in remote binding are less understood by IoT developers. To this end, we systematically examined the designs and implementations from products of 10 IoT vendors, and successfully launched real-world attacks on the devices. Our study brings the following misunderstandings in IoT remote binding to the spotlight:

- First of all, the use of static device IDs for device authentication allows the attacker to compromise device authentication. As aforementioned attacks show, this could lead to serious consequences like data injection and stealing or device hijacking. Given that IoT devices are first configured by users in most scenarios, we suggest that a better solution is to request a dynamic device secret from the user.
- Second, the binding between a user and a device involves the user's authorization to access her device in the cloud. As such, proper access control mechanisms should be enforced in binding (i.e., both pre-binding and post-binding). Unfortunately, the devices that we studied do not have such proper design of authorization. In this case, the device ID is essentially an ambient authority, and it cannot be used to represent the user's ownership (i.e. the authority of binding) of the device.
- Third, binding revocation is also a critical authorization step, in which only the user who is already bound with the device can revoke its binding. However, some devices do not have correct designs and implementations in this step. A prominent example is that binding revocation is achieved by creating a new binding to replace the previous binding, which has been demonstrated to suffer from multiple attacks.
- Last, given that nowadays IoT devices are attracting more and more attention of attackers, a user's sensitive information, such as the user account, should never be delivered to the device during remote binding.

VIII. Discussion

We have present a systemtic study on the remote binding mechanisms together with multiple attacks against them. In this section, we discuss some of the limitations of our research in terms of the scope and the evaluation. We also highlight a few of the potential follow-up studies that could be performed based on our findings.

Problem scope. First, our study is based on the common communication architecture that only includes three parties: the device (or the hub), the user, and the cloud. In future work, it may be interesting to see if our study could be generalized to other communication architectures that involve four parties: the Zigbee/Bluetooth device, the IP-based hub device, the user, and the cloud. Second, although the goal of our research is to understand the protocol-level vulnerabilities that are specific to remote binding, the remote binding process could also be affected by other vulnerabilities during local configuration, for example, Man-in-the-Middle attacks that compromise the local binding by proximity [23], key reinstallation attacks that allow injection and manipulation of encrypted home Wi-Fi packets [51], and sniffing attacks that allow the attacker to obtain Wi-Fi credentials [41].

Experimental results. First of all, since we could not obtain the firmware images for some of our experimental devices, we were unable to confirm data injection and stealing attacks on them. Second, although the results of our attacks do not have false positives, there could be some false negatives. In our experiments, the attack failures were identified from: response messages or the success of other attacks. As an example, the success of attack A3-3 indicates that binding denial-of-service attack would fail. However, given that some implementation flaws are in the cloud, we were unable to confirm the root causes of some potential attack failures.

Automatic detection. As a starting point, our systematic study and manual analysis reveal a series of attack surfaces in remote binding. In the future, we plan to explore the feasibility to develop effective and automatic approaches. This could further help IoT vendors improve the security of their products and their clouds. Currently, our analyses require the presence of physical devices, which is less scalable. Therefore, we would also like to explore the feasibility to automatically discover remote binding threat without the presence of physical devices.

IX. related work

In this section, we introduce and compare prior studies that are related to ours. We discuss how our work is different from prior IoT authentication and authorization studies. Apart from that, our study is also related to the security analysis of IoT devices.

Authentication in IoT. Given the unique communication architecture, authentication schemes in IoT have been heavily studied in previous works, such as [35], [36], [44], [46], [48], [54]. In fact, the works that are the most related to our work focus on the authentication problems in Wireless Sensor Networks (WSN). In WSN, there are three parties: the sensor node, the gateway node, and the user. Users can communicate with either the gateway node or a sensor node. For those WSN studies [31], [47], [50], [53], [55], [56], researchers proposed arbitrated mutual authentication protocols under various scenarios, such as authentication between a user and the gateway node [53], [56], mutual authentication between all three parties [24], and authentication between a particular user and a particular sensor node [29], [55].

Different from WSN authentication scenarios, in IoT remote binding, the device and the user are not authenticated to each other. Instead, the user and the device are both authenticated to the cloud. Based on that, the cloud then enforces access control and eventually relay messages between them. On the other hand, the attacker in our adversary model is regarded as an insider who already obtains a device identity. This is different from the WSN studies that assume the attacker is an outsider.

To the best of our knowledge, there is no prior study about remote binding standard protocols. The main reason is that IoT is a new topic, and the remote binding of IoT has not been studied by researchers. Although nowadays some vendors provide remote binding solutions, those homemade solutions are not formally verified. It is our future work to formally verify their security properties.

Authorization in IoT. Prior studies also have been focusing on coarse-grained authorization in IoT apps and IoT clouds [32]–[34], [39], [49]. For example, SmartAuth [49] is an NLP-based framework to bridge the gap between real behaviors in code and high-level functionalities in the descriptions of IoT apps, providing fine-grained access control. ContexIoT [39] utilizes context information for more fine-grained access control of sensitive actions in IoT platforms. Moreover, Earlence et al. [32] performed an empirical security analysis of one emerging smart home programming platform and found that the cloud-side privilege separation model could lead to significant over-privilege. FlowFence [33] tries to address the problem that existing permission-based access control is ineffective at controlling how sensitive data flows in apps, by embedding user intended data flow patterns. Different from those works, our study takes the first step to systematically analyze the entire life cycle of remote binding in IoT, which involves device authentication and user authorization.

Security analysis of IoT. Besides, researchers also have a increasing interest in the security of IoT devices [22], [27], [28], [37], [38], [45]. At a high level, their works can be categorized into two directions: performing security analysis on IoT devices and proposing defense techniques to tackle coarse-grained privilege separation. On the one hand, given that security analysis of IoT devices faces huge challenges, researchers propose various techniques to discover implementation flaws and explore attack vectors. For instance, Costin et al. [27] performed a large scale analysis of implementation flaws in 32 thousand firmware images and discovered 38 previously unknown vulnerabilities, indicating that today's firmware of IoT devices is poorly implemented. For exploration of attack vectors, Müller et al. [45] performed a systematic study on network printers by summarizing existing attacks and designing a tool to detect known attacks. Ho et al. [38] studied 5 popular smart locks and discovered several new attacks to leak information and even unlock the doors. On the other hand, researchers propose various techniques [25], [26], [40] to separate privileges for IoT systems. For instance, Abraham et al. [26] implemented a runtime privilege overlay to provide protections (stack protections and diversification of code and data regions) for bare-metal systems. Different from those works, our study exploits cloud-side authorization and device authentication to take control of devices or to cause denial-of-service to users.

X. Conclusion

This paper reports the first systematic study on the life cycle of remote binding in IoT. To understand the security threat, we demystify various design principles of IoT remote binding with a state machine, in which the procedures of remote binding are represented as cloud-side device state transitions in response to primitives messages sent by the device and the user. This process model provided us a unique observation of the remote binding process, which enables us to systematically discover four types of attacks: sensitive user data leak, persistent denial-of-service, connection disruption, and stealthy device control. We also carried out case studies and successfully launched attacks on 10 popular IoT devices, which reveals the prevalence of our attacks in the real world.

Acknowledgements

We are grateful to our shepherd Kaustubh Joshi and the anonymous reviewers for their insightful comments. This work was partially supported by National Natural Science Foundation of China (Grant No. 61572415) and the General Research Funds (Project No. 14217816 and 14208818) established under the University Grant Committee of the Hong Kong Special Administrative Region, China, as well as NSF Award 1834215 and 1834216. Wenrui Diao was supported in part by the Fundamental Research Funds for the Central Universities (No. 21618330).

References

[1] "Ambient Authority," https://en.wikipedia.org/wiki/Ambient_authority, Accessed: April 2019.
[2] "Android Frida," https://www.frida.re/docs/android/, Accessed: April 2019.
[3] "ARTIK Cloud Development: Secure your device," https://developer.artik.cloud/documentation/security.html#secure-registration, Accessed: April 2019.
[4] "AWS IoT Authentication," https://docs.aws.amazon.com/iot/latest/developerguide/iot-authentication.html, Accessed: April 2019.
[5] "D-LINK Wi-Fi Smart Plug," http://us.dlink.com/products/connected-home/wi-fi-smart-plug/, Accessed: April 2019.
[6] "Fiddle: Web Debugging Proxy," https://www.telerik.com/fiddler, Accessed: April 2019.
[7] "Google Cloud IoT: Device Security," https://cloud.google.com/iot/docs/concepts/device-security, Accessed: April 2019.
[8] "IBM Watson IoT: Device authentication," https://www.ibm.com/developerworks/library/iot-trs-secure-iot-solutions1/index.html, Accessed: April 2019.
[9] "IFTTT: IF This Then That," https://ifttt.com/, Accessed: April 2019.
[10] "MAC addresses: the privacy Achilles' Heel of the Internet of Things," https://www.computing.co.uk/ctg/news/2433827/mac-addresses-the-privacy-achilles-heel-of-the-internet-of-things, Accessed: April 2019.
[11] "Postman API Development Environment," https://www.getpostman.com/docs/v6/, Accessed: April 2019.
[12] "Simple Service Discovery Protocol," https://en.wikipedia.org/wiki/Simple_Service_Discovery_Protocol, Accessed: April 2019.
[13] "SimpleLink Wi-Fi SmartConfig Technology," http://www.ti.com/tool/SMARTCONFIG, Accessed: April 2019.

[14] "Someone Is Taking Over Insecure Cameras and Spying on Device Owners," https://www.bleepingcomputer.com/news/security/someone-is-taking-over-insecure-cameras-and-spying-on-device-owners/, Accessed: April 2019.

[15] "TP-LINK Smart Plug HS100," https://www.tp-link.com/us/products/details/HS100.html, Accessed: April 2019.

[16] "Use Airkiss To Configure Wi-Fi Connection," https://www.amebaiot.com/en/standard-sdk-airkiss/, Accessed: April 2019.

[17] "Using Shadows - AWS IoT," https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-shadows.html, Accessed: April 2019.

[18] "Vulnerabilities in Fredi Wi-Fi baby monitor can be exploited to use it a spy cam," https://securityaffairs.co/wordpress/73848/hacking/fredi-wi-fi-baby-monitor.html, Accessed: April 2019.

[19] "WeMo Switch Smart Plug," http://www.belkin.com/us/p/P-F7C027/, Accessed: April 2019.

[20] "Wi-Fi Protected Access II," https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access#WPA2, Accessed: April 2019.

[21] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware." in *NDSS*, 2016.

[22] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X. Wang, W. C. Lau, M. Sun, R. Yang, and K. Zhang, "IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing," in *Proceedings of The Network and Distributed System Security Symposium*, 2018.

[23] J. Chen, M. Sun, and K. Zhang, "Security Analysis of Device Binding for IP-based IoT Devices," in *The Third International Workshop on Security, Privacy and Trust in the Internet of Things*. Percom, 2019.

[24] T.-H. Chen and W.-K. Shih, "A robust mutual authentication protocol for wireless sensor networks," *ETRI journal*, vol. 32, no. 5, pp. 704–712, 2010.

[25] A. A. Clements, N. S. Almakhdhub, S. Bagchi, and M. Payer, "ACES: Automatic Compartments for Embedded Systems," in *27th USENIX Security Symposium*. USENIX Association, 2018.

[26] A. A. Clements, N. S. Almakhdhub, K. S. Saab, P. Srivastava, J. Koo, S. Bagchi, and M. Payer, "Protecting bare-metal embedded systems with privilege overlays," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 289–303.

[27] A. Costin, J. Zaddach, A. Francillon, D. Balzarotti, and S. Antipolis, "A Large-Scale Analysis of the Security of Embedded Firmwares." in *USENIX Security Symposium*, 2014, pp. 95–110.

[28] A. Costin, A. Zarras, and A. Francillon, "Automated dynamic firmware analysis at scale: a case study on embedded web interfaces," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 437–448.

[29] A. K. Das, P. Sharma, S. Chatterjee, and J. K. Sing, "A dynamic password-based user authentication scheme for hierarchical wireless sensor networks," *Journal of Network and Computer Applications*, vol. 35, no. 5, pp. 1646–1656, 2012.

[30] N. Dhanjani, *Abusing the internet of things: Blackouts, freakouts, and stakeouts*. " O'Reilly Media, Inc.", 2015.

[31] M. S. Farash, M. Turkanović, S. Kumari, and M. Hölbl, "An efficient user authentication and key agreement scheme for heterogeneous wireless sensor network tailored for the Internet of Things environment," *Ad Hoc Networks*, vol. 36, pp. 152–176, 2016.

[32] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 636–654.

[33] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "FlowFence: Practical Data Protection for Emerging IoT Application Frameworks." in *USENIX Security Symposium*, 2016, pp. 531–548.

[34] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized Action Integrity for Trigger-Action IoT Platforms," in *Proceedings of The Network and Distributed System Security Symposium*, 2018.

[35] M. A. Ferrag, L. A. Maglaras, H. Janicke, J. Jiang, and L. Shu, "Authentication protocols for Internet of Things: a comprehensive survey," *Security and Communication Networks*, vol. 2017, 2017.

[36] K. Habib and W. Leister, "Context-Aware Authentication for the Internet of Things," in *The Eleventh International Conference on Autonomic and Autonomous Systems*, 2015, pp. 134–139.

[37] G. Hernandez, F. Fowze, D. J. Tian, T. Yavuz, and K. R. Butler, "FirmUSB: Vetting USB Device Firmware using Domain Informed Symbolic Execution," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 2245–2262.

[38] 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*. ACM, 2016, pp. 461–472.

[39] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, "ContexIoT: Towards providing contextual integrity to appified IoT platforms," in *Proceedings of The Network and Distributed System Security Symposium*, vol. 2017, 2017.

[40] C. H. Kim, T. Kim, H. Choi, Z. Gu, B. Lee, X. Zhang, and D. Xu, "Securing Real-Time Microcontroller Systems through Customized Memory View Switching," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium*, 2018.

[41] C. Li, Q. Cai, J. Li, H. Liu, Y. Zhang, D. Gu, and Y. Yu, "Passwords in the air: Harvesting wi-fi credentials from smartcfg provisioning," in *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2018, pp. 1–11.

[42] Z. Ling, J. Luo, Y. Xu, C. Gao, K. Wu, and X. Fu, "Security Vulnerabilities of Internet of Things: A Case Study of the Smart Plug System," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1899–1909, 2017.

[43] H. Liu, C. Li, X. Jin, J. Li, Y. Zhang, and D. Gu, "Smart solution, poor protection: An empirical study of security and privacy issues in developing and deploying smart home devices," in *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*. ACM, 2017, pp. 13–18.

[44] J. Liu, Y. Xiao, and C. P. Chen, "Authentication and access control in the internet of things," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*. IEEE, 2012, pp. 588–592.

[45] J. Müller, V. Mladenov, J. Somorovsky, and J. Schwenk, "Sok: Exploiting network printers," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 213–230.

[46] K. T. Nguyen, M. Laurent, and N. Oualha, "Survey on secure communication protocols for the Internet of Things," *Ad Hoc Networks*, vol. 32, pp. 17–31, 2015.

[47] R. Roman, C. Alcaraz, J. Lopez, and N. Sklavos, "Key management systems for sensor networks in the context of the Internet of Things," *Computers & Electrical Engineering*, vol. 37, no. 2, pp. 147–159, 2011.

[48] M. Saadeh, A. Sleit, M. Qatawneh, and W. Almobaideen, "Authentication techniques for the internet of things: A survey," in *Cybersecurity and Cyberforensics Conference (CCC), 2016*. IEEE, 2016, pp. 28–34.

[49] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "SmartAuth: User-Centered Authorization for the Internet of Things," in *26th Security Symposium*. USENIX Association, 2017, pp. 361–378.

[50] M. Turkanović, B. Brumen, and M. Hölbl, "A novel user authentication and key agreement scheme for heterogeneous ad hoc wireless sensor networks, based on the Internet of Things notion," *Ad Hoc Networks*, vol. 20, pp. 96–112, 2014.

[51] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in wpa2," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1313–1328.

[52] D. Wang, X. Zhang, J. Ming, T. Chen, C. Wang, and W. Niu, "Resetting Your Password Is Vulnerable: A Security Study of Common SMS-Based Authentication in IoT Device," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[53] R. Watro, D. Kong, S.-f. Cuti, C. Gardiner, C. Lynn, and P. Kruus, "TinyPK: securing sensor networks with public key technology," in *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*. ACM, 2004, pp. 59–64.

[54] D. J. Wu, A. Taly, A. Shankar, and D. Boneh, "Privacy, discovery, and authentication for the internet of things," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 301–319.

[55] K. Xue, C. Ma, P. Hong, and R. Ding, "A temporal-credential-based mutual authentication and key agreement scheme for wireless sensor networks," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 316–323, 2013.

[56] H.-L. Yeh, T.-H. Chen, P.-C. Liu, T.-H. Kim, and H.-W. Wei, "A secured authentication protocol for wireless sensor networks using elliptic curves cryptography," *Sensors*, vol. 11, no. 5, pp. 4767–4779, 2011.

[57] J. Zaddach and A. Costin, "Embedded devices security and firmware reverse engineering," *Black-Hat USA*, 2013.

[58] N. Zhang, S. Demetriou, X. Mi, W. Diao, K. Yuan, P. Zong, F. Qian, X. Wang, K. Chen, Y. Tian *et al.*, "Understanding IoT Security Through the Data Crystal Ball: Where We Are Now and Where We Are Going to Be," *arXiv preprint arXiv:1703.09809*, 2017.