Security Vulnerabilities of Internet of Things: A Case Study of the Smart Plug System

Zhen Ling*, Junzhou Luo*, Yiling Xu*, Chao Gao†, Kui Wu‡ and Xinwen Fu†
*Southeast University, Email: {zhenling, jluo, ylxu}@seu.edu.cn
‡University of Victoria, Email: wkui@uvic.ca
†University of Massachusetts Lowell, Email: {cgao, xinwenfu}@cs.uml.edu

***Abstract*—With the rapid development of the Internet of Things (IoT), more and more small devices are connected into the Internet for monitoring and control purposes. One such type of devices, smart plugs, have been extensively deployed worldwide in millions of homes for home automation. These smart plugs, however, would pose serious security problems if their vulnerabilities were not carefully investigated. Indeed, we discovered that some popular smart home plugs have severe security vulnerabilities which could be fixed but unfortunately are left open. In this paper, we case study a smart plug system of a known brand by exploiting its communication protocols and successfully launching four attacks: device scanning attack, brute force attack, spoofing attack, and firmware attack. Our real-world experimental results show that we can obtain the authentication credentials from the users by performing these attacks. We also present guidelines for securing smart plugs.**

Keywords*—*Internet of Things, Vulnerabilities, Attacks, Countermeasures

I. Introduction

The emergence of Internet of Things (IoT) provides the capabilities of connecting smart devices, small actuators, and people anywhere and anytime to the Internet. Gartner forecasts that the number of IoT grows 31% from 6.4 billion in 2016 to 8.4 billion in 2017, and will reach 20.4 billion by 2020 [1]. Smart plugs, as one type of fast emerging IoT devices, are gaining increasing popularity in home automation, with which users can remotely monitor and control their homes. Figure 1 shows an example smart plug, i.e., Edimax SP-2101W, and the iPad that is installed with the control application, i.e., EdiPlug. Various applications can be implemented over such a system. For instance, in winter time users can turn on the heater with a smartphone in advance to warm up their homes before they return home. They can also rely on smart plugs with the energy management function to accurately monitor the energy consumption. Medical equipment may also be connected to smart plugs for smart health. Due to the tremendous benefit, smart plugs have been deployed worldwide in millions of homes.

Security concerns come along with the popularity of smart plugs. Compromised smart plugs would lead to both security and privacy breach of home users. If smart plugs are used in commercial or industrial buildings for demand response [2], the consequences of smart plugs being compromised and controlled by attackers could be disastrous. A disrupted medical equipment connected to smart plugs may threaten a patient's life. In recent years, the security concerns of smart plugs have received substantial consideration in both industry and academic communities [3], [4].

Fig. 1. Edimax plug and iPad installed with the control application

Despite the importance and broad concern of security problems in smart plugs, we found their vulnerabilities are still prominently exposed. As an evidence, we in this paper case study the security problems of a typical smart plug system, i.e., Edimax SP-2101W. With reverse engineering, we disclose its entire communication protocols and identify its vulnerabilities that could open the door to different attacks. We propose four attacks: device scanning attack, brute force attack, spoofing attack and firmware attack. Extensive real-world experiments show that we can effectively and efficiently obtain a victim's authentication credentials via these attacks. As a remedy, we present defense guidelines to mitigate these attacks.

The goal of this paper is to send out a strong message to the IoT community and hopefully to enforce smart plug manufacturers/developers to put security at a higher priority. As such, the code of our attacks will not be disclosed.

Our main findings regarding the vulnerabilities of the Edimax plug system in question can be summarized as follows.

Insecure communication protocols. Since the communication protocols do not rely on cryptographic mechanisms, an attacker could capture network traffic and reverse engineer the communication protocols. In this case, the system is subject to various eavesdropping attacks.

Lack of device authentication. The remote server used by an app communicating with plugs does not authenticate the plugs. This widely opens the door for an attacker to perform our four attacks.

1) The Edimax plug system uses the MAC address of a plug as the identity of the plug. We are able to use the *device scanning attack* and scan the MAC address space of the vendor in order to find the online status of all smart plugs made by the vendor. The device scanning attack can also reveal if users use the default password of a plug since many users do not change the default password of their smart devices [3] due to lack of security awareness.
2) If the plug is online and the password is changed, we can perform the *brute force attack* to infer the passwords. Given a default password of "1234", it is likely that a user may change it to a 4-digit one since the vendor does not explicitly list their password policy in their documentation. The remote server does not limit the password attempts by an app.
3) If long passwords are employed by users, we can launch the *device spoofing attack*, which blocks the genuine plug and pretends to be a legal one, waiting for the remote application to send the authentication credential of a user for login and use of the plug. In this attack, the users leak their authentication credentials once opening the plug control applications. The attack is also stealthy and the users can hardly realize that they are attacked. Using the credential, the attacker can completely control the genuine plug.
4) Moreover, we study the firmware update process and perform the *firmware attack* to upload a malicious firmware to the plug. With such a malicious firmware, an attacker can create a reverse tunnel from the plug to a desired server and gain the root access on the plug system.

For countermeasures to the potential attacks exploiting the above vulnerabilities, we present the following guidelines to protect smart plug systems, including secure communication protocols to block eavesdropping attacks, mututal authentication between the control app and plug through the remote server, intrusion detection system for abnormal behavior detection, anti-bot mechanisms, and validation of data integrity.

The rest of this paper is organized as follows: We give an overview of our protocol analysis strategy and the discovered Edimax plug system architecture in Section II. In Section III, we present the detailed communication protocol, including the registration phase, authentication phase, control phase, and firmware update. In Section IV, we introduce four attacks. In Section V, we perform extensive empirical experiments to demonstrate the feasibility and effectiveness of our attacks. In Section VI, we discuss the corresponding countermeasures. Related work is presented in Section VII. Finally, we conclude the paper in Section VIII.

II. Protocol Analysis Overview

In this section, we first briefly introduce the smart plugs we will exploit. We then introduce our platform that is used to analyze this Edimax plug system. We also discuss strategies to analyze the content of the communication traffic. Finally we introduce the big picture of the smart home plug system architecture of interest.

A. Smart Plugs of Interest

A smart home plug is an electric device that can be plugged into an ordinary outlet. It provides outlets for other electronic devices, e.g., lamps and fans. It is often designed to connect to the wireless home network so that a user can install an app on her smart device, e.g., smartphone, and control the electronic device plugged into the smart plug over the Internet. Smart plugs are gaining popularity for building a home automation system.

We selected a typical smart plug, i.e., Edimax SP-2101W, as shown in Figure 1. The device is available from Amazon and Walmart and has a rating of more than 4.0 out of 5.0. The app for controlling the plug supports both Android and iOS platforms. The plug provides the power meter functionality and allows users to manage the energy consumption. For instance, a user can monitor the power usage of the plugged appliance and make a schedule to turn on/off the appliance via the app. Moreover, a user can set up email information, including username, password, SMTP (Simple Mail Transfer Protocol) server, etc., for the plug so that the plug can send alert emails to the users.

B. Network Traffic Acquisition and Analysis Platform

To analyze the network traffic and learn the architecture of the Edimax plug system, we establish an experiment network platform to capture the traffic at both the smart plug and the app. We use two machines to set up two wireless APs with wireless USB adapters. We install the Ubuntu 14.04 operation system on these two machines, which are connected to the Internet through Ethernet network cards to obtain public IP addresses. To establish wireless local area network (WLAN), the NAT function is configured using the Linux firewall, i.e., iptables. Moreover, we set up a dynamic host configuration protocol (DHCP) service to automatically assign local IP addresses to the devices connected to our APs. We then use the network traffic sniffer, *tcpdump*, to capture the incoming and outgoing traffic at the smartphone and the smart plug.

One of these two APs is used for the smart plug while the other is for the smartphone. Since these APs obtain two different public IP addresses, we can use the platform to study the remote control communication protocol of the Edimax plug system. We can also connect the smartphone directly to the AP used by the smart plug in order to investigate the local control communication protocol of this plug system.

C. Reverse Engineering Smart Plug Communication Protocols

We observed two types of network traffic in the Edimax plug communication, i.e., the plaintext packets and obfuscated packets. We first study the plaintext packets and then present the solution to decode obfuscated packets.

Analyzing plaintext packets. When the plug and the controller are located in the same WLAN, we find that the traffic between the plug and the controller is not encrypted. HTTP is used as the communication protocol between the plug and controller. The plug uses the HTTP basic authentication method to authenticate the controller.

The controller sends a HTTP POST request that contains an authentication field. The authentication field contains the username and password that are concatenated with a single colon and are encoded with the Base64 scheme. The URL link in the HTTP header is *http://host:10000/smartplug.cgi*, where *host* is the local IP address of the plug.

The payload of the HTTP POST packet contains a plaintext message encapsulated in the XML format. The message consists of several fields, including *command*, *the state of device*, *power*, *current*, *schedule*, *time*, *SMTP server information*, and so on. By enumerating all of the plug operations, we can learn the entire control protocol when the plug and the controller are in the same WLAN. To be specific, the controller sends two types of command messages, the *get* command and the *setup* command, where the former is used to obtain information from the plug and the latter is to control the plug, e.g., turning it on/off. The plug will respond with the corresponding messages to the command messages from the controller. The plaintext messages also help us decode the obfuscated packets among the remote servers, the plug, and the controller.

Decoding obfuscated packets. We study the firmware of the plug and find that the messages are not encrypted. Since the latest firmware can be downloaded from the plug's official website, we can extract the binary code that is responsible for communication between a plug and remote servers. We use IDA Pro to carefully inspect the binary code and find that bitwise shifting is used to obfuscate the messages. The message is encapsulated using the XML format. Brackets "<>" are used to separate key-value pairs. The first byte of the message is <. To obfuscate the message, a sender (e.g., the plug or the controller) randomly selects the number of positions, between 1 and 7, and then performs the bitwise right shift operation to the message except the first byte <. She then adds the number of positions for the left shifting to the ascii code of <. A receiver (e.g., the controller or the plug) compares the first byte of the content with the ascii code of < to derive the number of positions for right shifting, and then performs the corresponding right shifting for the rest of the bytes in the message in order to decode the entire message.

D. Architecture of Smart Home Plug System

By analyzing the communication between the plug and smartphone, we find that the Edimax plug system consists of three components: smart plugs, controllers, and remote cloud servers. Figure 2 illustrates the architecture of the Edimax plug system. The plug can connect to a wireless home access point (AP) for Internet access. The controller is a smart device, e.g., smartphone, that is installed with the plug app. If the controller and the plug are in the same local network, the app can directly communicate with the plug through the local AP. If the controller and the plug are in different networks, the controller can communicate with the smart plug through the cloud servers on Amazon EC2. We find two types of servers in the cloud, i.e., authentication server and command relay server. The authentication server is used to authenticate both the plug and the controller. Since most of the plugs located in home networks are behind a network address translation (NAT), the command relay server can work as a relay server to forward command messages between the plug and the controller.

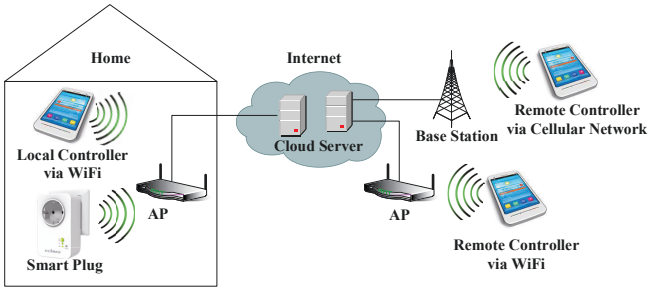

Fig. 2. The architecture of the Edimax plug system

III. Detailed Communication Protocol of Smart Plug System

By performing extensive exploratory experiments, we find that the communication protocol of the Edimax plug system has three phases: smart plug registration phase, authentication phase, and communication phase. In this section, we present these three phases in detail.

A. Smart Plug Registration Phase

When the plug connects to an outlet and powers up for the first time, it works as a wireless access point (AP). We call it the *plug AP* to differentiate it from a home wireless router. A user can use her smartphone to connect to the plug AP. After the smartphone is associated with the plug AP, the smartphone searches the home wireless router and can connect the plug to the home wireless router. Once the plug can access the Internet, it will register with a remote authentication server. The detailed procedure is introduced as follows.

STEP 1: The smart plug establishes a TCP connection to *www.google.com*. In this step, the smart plug performs the network reachability test to check if the plug can access the Internet. If it could not access the Google website, the plug will stop working. Apparently, this strategy of testing Internet connection is not robust since a number of countries such as China block Google services [5].

STEP 2: The smart plug connects to a time server *pool.ntp.org*, using the network time protocol (NTP) to synchronize the clock of the plug. A synchronized clock is necessary since the plug system uses the time information for its communication and task scheduling service, e.g., periodically turning on/off the plug on time.

STEP 3: The smart plug sends datagram packets to a remote server and registers with the server. According to our analysis, this remote server is deployed on Amazon EC2 and is only used for relaying UDP packets for authentication, we call it

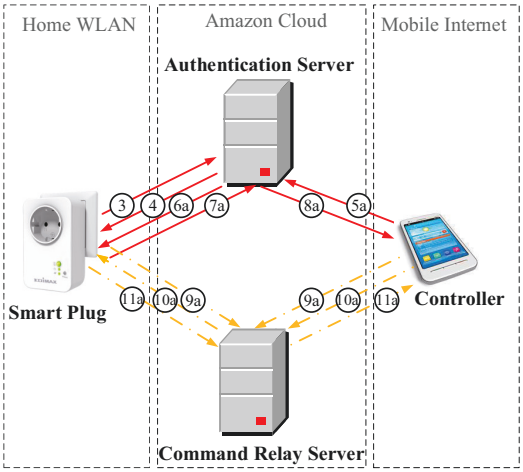

Fig. 3. Edimax Plug and controller in different networks

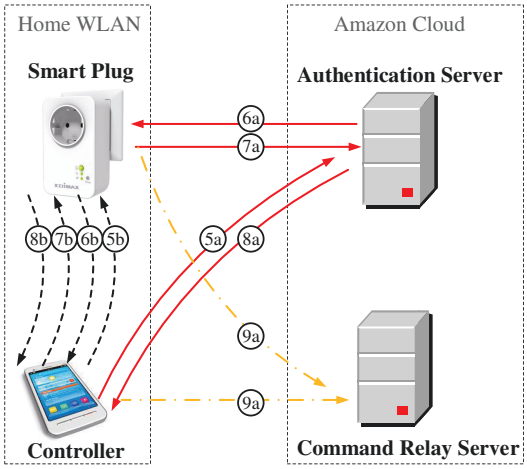

Fig. 4. Edimax Plug and controller in the same WLAN

the *authentication server*. The UDP port of the authentication server for the plug is 8765.

The smart plug sends two consecutive UDP packets to the authentication server. The content in these packets are encapsulated using the XML format. The first datagram packet sent by the plug contains a value of "3000" in the "code value" field to inform the authentication server for the registration. This field is referred to as "command type" in our paper. The second datagram packet includes a command type of "1010" and the plug information including the plug model, *MAC address*, type, alias, LAN IP address and port of this plug and device firmware version. The second packet is used to notify the authentication server that the plug is online. The plug sends a "1010" datagram packet every 20 minutes periodically to keep the server informed of the online status of the plug.

STEP 4: Upon receiving the messages with the command type "1010" from the plug, the authentication server sends a response UDP packet. The command type of this response packet is "1020". This packet contains the smart plug's MAC address (sent to the server in STEP 3) and the status value.

B. Authentication Phase

There are two different scenarios in the authentication phase. In the first scenario, the plug and the controller are located in different networks. In the other scenario, they are in the same wireless local area network. We elaborate the communication procedure of these scenario.

1) Plug and Controller in Different Networks: Figure 3 illustrates the authentication procedure between the smart plug and the controller that are in different networks. For example, the smart plug is located in the home network and connects to the home WLAN while the controller accesses the Internet through the cellular network or another WLAN. In this case, the smart plug authenticates the controller via the authentication server.

STEP 5a: The controller sends a UDP request with a command type of "1030" to the authentication server. The UDP port of the authentication server for the controller is 8766. The request packet contains a credential in the "auth value" field for authentication and information of the MAC address. The value of the credential is hashed with the MD5 hashing algorithm. The credential consists of the user account and password. The format of this value is *username:password*. The default username and password is *admin* and *1234* respectively. A user can change the password via the app installed on the smartphone. However, the username, i.e., *admin*, is hard-coded into the application. The request packet also contains the plug's MAC address and timestamp.

STEP 6a: In this step, the authentication server processes the UDP request and forwards it to the right smart plug. Once receiving the datagram request from the controller, the authentication server first checks the status of the plug with the MAC address sent from the controller. If the plug is online at that point, the server changes the command type to "1040" and then adds additional information to the original request and forwards it to the smart plug. The additional information includes the IP address and port of the controller, the IP address and port of the command relay server, the relay ID, and the credential from the controller. The relay ID is a 24 character hex string that is generated by the authentication server. It is used at the control phase to correlate the TCP connections between the controller and the plug. If the plug is offline, the authentication server will send back a datagram packet with a command type "5000" to the controller.

STEP 7a: In this step, the smart plug authenticates the controller and sends back a datagram response to the authentication server. After receiving the request from the authentication server, the smart plug will check the credential to authenticate the identity of the remote controller. If the credential from the controller is correct, the plug will send a response packet with a command type "1060". The "1060" packet includes the IP address and port of the command relay server, relay ID, etc. If the credential is incorrect, the plug will send a datagram packet with a command type of "1120" to the authentication server. However, the server will not forward this message to the controller.

STEP 8a: In this step, the authentication server forwards this datagram response packet (except the "1120" packet) to the controller. Upon receiving the UDP response from the

plug, the authentication server modifies the command type to "1070", adds additional information, and then forwards the response package to the controller. The additional information includes the IP address and port of both the plug and the command relay server, the relay ID, and the information of the plug including the model, type, alias, firmware version, etc. Once the controller receives this "1070" packet, the entire authentication procedure between the plug and the controller completes.

2) Plug and Controller in the Same WLAN: Assume that the plug and controller are in the same wireless local area network as shown in Figure 4. The detailed authentication process between the plug and the controller is introduced below.

STEP 5b: Once a controller manages to connect to a WLAN, the controller broadcasts two consecutive 22-byte datagram packets in order to determine if the plug and the controller are in the same WLAN. These packets are used for discovering the plug, in case both the plug and controller are located in the same WLAN. The destination port of this broadcast packets is 20560. The controller also continues the authentication phase introduced in Section III-B1 sending the credential to the smart plug via the remote authentication server.

STEP 6b: Upon receiving the broadcast datagram packets, the plug will respond immediately. The plug sends back a datagram packet to the controller. The information in the packet includes model, MAC address, IP address, firmware version, and alias of this plug. In this way, both the plug and the controller know that they are located in the same WLAN.

STEP 7b: The controller establishes a TCP connection to a server deployed on the plug and leverages the HTTP protocol to communicate with the local smart plug. The destination port of this HTTP server is 10000. Once the TCP connection is built, the controller sends the authentication information, i.e., user name and password, using the HTTP basic authentication method. The payload contains a *get* command to obtain the state of the plug so that the app will be able to display the current status of the plug.

STEP 8b: The smart plug responds with a HTTP packet to the controller. The message shows the state of the power, (i.e., on or off). The controller obtains this response message and shows the information to the user via the app. Therefore, after the authentication phase, the user can perform various control operations through the app. For instance, the user can reset the password or the SMTP server.

C. Control Phase

In the control phase, if the controller and the plug are located in the same WLAN, they can directly communicate with each other using the HTTP protocol introduced in **STEP 7b** and **STEP 8b**. If the controller and the plug are in different networks, the communication traffic between the controller and the plug passes through a remote server as shown in Figure 3. The command messages are encapsulated using the XML format. The content of the traffic from and to the plug, the controller, and the server is obfuscated but not encrypted.

STEP 9a: Both the smart plug and the controller establish TCP connections to a rendezvous server deployed in the Amazon cloud. We call this server as a command relay server as it is used to relay the commands between plugs and controllers. Recall that authentication server selects this rendezvous server and generates a relay ID, and then sends the relay ID to both the plug and the controller. Thus, after the plug and the controller build the connections to the command relay server, both the plug and the controller send a message composed of the MAC address of the plug and the relay ID to the command relay server. The relay server correlates these two TCP connections using the relay ID and MAC address. The relay server does not respond to either the plug nor the controller after receiving the messages.

STEP 10a: After sending the relay ID and plug MAC address to the relay server, the controller sends a command to the relay server. The message uses the XML format that is the same as the one used in the WLAN. For instance, to get the status of the plug, the controller will send a *get* command to the relay server. After receiving the command from the controller, the relay server will forward the command message to the plug without any change.

STEP 11a: Upon obtaining the message from the controller, the plug responds to this command. According to the command message, the plug sends back the corresponding information to the relay server. For example, the plug may report the state of the plug. The command relay server forwards the response to the controller without any change.

The controller can send a *setup* command to control the plug. For instance, if the state of the plug is *on*, the controller can send an *off* command to the plug through the relay server in order to change the plug's state. The plug executes the *setup* command and turns off the plug after receiving the command message. The plug will then send the execution result to the controller via the relay server to inform that the *setup* command has been successfully executed. After the controller receives the execution result from the relay server, the control phase completes.

D. Firmware Update

The firmware of the plug can be updated through a firmware upgrade tool, which is designed for the Microsoft Windows operating system. The Windows system and the plug should connect to a same local network. Once the tool is opened, it performs the operation in **STEP 5b** to determine if the plug and the controller are in the same WLAN. The plug performs **STEP 6b** to send the information of the plug to the tool. After receiving the plug information, the tool displays the plug model, MAC address, IP address, firmware version and the upgrade status. If a new version is available, the upgrade status shows that a new firmware version can be used. The user can click the new version on the tool, which pops up a prompt box and asks the user to input the password of the plug. After gaining the password from the user, the firmware upgrade tool generates a firmware (i.e., a Linux *bin* file) in a temp file folder and then uploads this firmware to the HTTP server on this plug. The password is encoded in the HTTP header using the HTTP basic authentication method. The plug installs this firmware after receiving the file and restarts. Once this firmware upgrade

process is completed, the plug automatically connects to the AP and the user can use the controller to access the plug again.

IV. Security Vulnerabilities of Smart Plug

In this section, we introduce four attacks, i.e., device scanning attack, brute force attack, device spoofing attack and firmware attack in detail. We also discuss the possible impact after an attacker can access the plugs. Please note that we use our own smart plugs for security analysis in order to avoid legal issues.

A. Device Scanning Attack

In a device scanning attack, the attacker can scan all plugs by enumerating possible MAC addresses of the smart plugs from this vendor. According to recent research [3], many users do not change the default password after deploying their IoT devices. They expect the vendor takes care of the security. Recall that in the authentication phase between the plug and the controller, the controller can receive the "1070" packet as discussed in **STEP 8a** if the plug is online and the password is correct. An attacker can craft an authentication message that specifies the plug MAC address, the default username and password, i.e., *admin:1234*, and check if any victim is using a plug with the specified MAC and the default password. Here "admin" is hard coded and actually does not play the role as a username since the username is not used to differentiate different controllers or users. The MAC address of the plug works as kind of username.

The key to a successful device scanning attack is to know the MAC address space of the smart plug. Luckily for the attacker (unluckily for the manufacturer), MAC addresses are predictable. We can search the MAC address spaces allocated to a company/manufacturer on the Internet [6]. The first 6 digits of a MAC address indicate the device manufacturer and the other 6 digits refer to a specific MAC address given to the manufacturer. A manufacturer often gives a block of sequential MAC addresses to the same product. Therefore, if we buy a few smart plugs, we can guess at least portions of MAC addresses allocated to smart plugs of this model. The attacker can enumerate the whole MAC address space of a manufacturer in a brute force attack.

Table I shows the possible responses to a controller that sends an authentication message to a plug with a specified MAC address and password. If the plug with the specified MAC address is online and the password is correct, the adversarial controller can receive the "1070" packet. If the plug with the specified MAC address is online and the password is wrong, the plug sends a packet with a command type of "1120" to the authentication server and the authentication server will not forward the message to the controller. To deal with this case in programming, the attacker should set a timer and try more times if the attacker does not obtain a response packet in case that the "1070" UDP packet is lost during the transmission. If the plug with the specified MAC address is offline or does not exist, the authentication server sends a "5000" packet to the attacker. Therefore, when a "5000" packet is received, the attacker cannot tell if the plug with the specified MAC address is offline or there is no plug with that MAC address. However, this does not affect the device scanning attack. Based on Table I, the attacker can leverage the server response in order to find plugs with default passwords and plugs not using the default password/specified password.

TABLE I. Response to controller that sends authentication messages to plug

	Password Correct	Password Wrong
Plug Online	1070	no response
Plug Offline or N/A	5000	5000

B. Brute Force Attack

After deploying the scanning attack, the attacker can discover all the online plugs using non-default passwords. Then the attacker can select those plugs, construct "1030" packets, and enumerate all possible passwords. The attacker just needs to wait until she receives the right response. At the time of the writing, our experiments show that the authentication server does not block this brute force password attack.

However, our experiments show that the Edimax plug system actually allows a password of 20 characters, including digits and upper-case and lower-case alphabetic letters. This password policy is not written in any of the provided manual and we cannot find it online either. If a user indeed inputs a long and complicated password, the brute force attack does not work anymore. Unluckily, the plug system suffers from the following device spoofing attack, which can expose any plug credential.

C. Device Spoofing Attack

1) Attack Process: In the device spoofing attack, we create a software bot that mimics a plug and performs the authentication with the remote controller in order to directly obtain the credential from the controller. It works as follows.

a) The attacker first selects a target plug with a specific MAC address. Recall the attacker knows this plug is online and this plug does not use the default password by using the device scanning attack. If the attacker has sufficient resources, she can simultaneously choose as many targets as she wants.

b) The attacker registers the spoofed plug by performing **STEP 3** in Section IV. In particular, the attacker can emulate the communication behavior of a real plug and send a packet with a command type of "1010" to the authentication server. Since the server does not provide any authentication method to authenticate the plug, it registers this spoofed plug and sends back the response packet with the command type of "1020" as introduced in **STEP 4** in Section IV. At this point, the spoofed plug is online.

c) When a victim opens her app on the smartphone, the app will automatically send the "1030" packet to the authentication server as introduced in **STEP 5a** in Section IV. The authentication server will forward this message to the attacker's spoofed plug as introduced in **STEP 6a** in Section IV. Since the "1030" packet

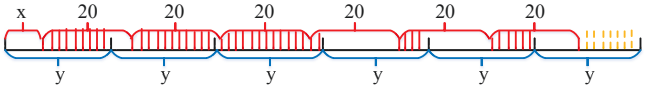

Fig. 5. An example of device spoofing attack

contains the credential, the attacker can effectively derive the credential.

d) To keep the victim from discovering the abnormal authentication process, the attacker continues to execute **STEP 7a** in Section IV. The app will receive the desired packet as introduced in **STEP 8a** in Section IV. At this time, the entire authentication phase completes.

Regardless of whether the plug and the controller are in the same WLAN or not, the controller always authenticates with the plug through the remote authentication server! As a result, the spoofed plug can always derive the credential during the authentication phase.

If the attacker wants to hide the spoofing attack from the victim, more has to be done by the spoofed plug. There are two cases. In the first case, the real plug and the controller are in the same WLAN. In this case, actually, the spoofed plug will not affect the real plug control procedure at all, since the controller will authenticate with the real plug in the WLAN as introduced in **STEP 5b** and **STEP 6b** in Section IV. The controller can control the plug with **STEP 7b** and **STEP 8b** in Section IV. Consequently, the victim will not realize this attack at all.

In the second case, the plug and controller are in different networks. In this case, when the spoofing attack is deployed, the victim controller communicates with the spoofed plug. The challenge for the attack being stealthy is how the spoofed plug relays the victim's commands to the real plug so that the plug control looks normal to the victim, who will not realize she is being attacked. To this end, the attacker first needs to stop sending the "1010" packets to the authentication server. This is to stop the spoofing attack and allow the real plug to register to the authentication server as soon as possible. The attacker should then move on to build a TCP connection to the command relay server and send the MAC address of the target plug and the right relay ID with **STEP 9a** in Section IV. Therefore, the victim will send the command message to the spoofed plug. The attacker should record the victim's operations, e.g., turning on/off the plug. Recall that the real plug sends a "1010" packet every 20 minutes with **STEP 3** in Section IV. When the authentication server receives this packet, the real plug is online again. At this time, the attacker can then access the real plug using the credential and replay the victim's commands to the real plug.

2) Issues: In the spoofing attack, the real plug sends the "1010" packet to the authentication server every 20 minutes in order to keep its online status. To address this issue, the attacker should periodically send "1010" packets to the authentication server so as to keep the spoofed plug online. The attacker may want to keep the spoofed plug online as long as possible in order to increase the success rate of the attack.

We now compute the attack success rate when a user opens the plug control app. Denote x as the time between the first "1010" packet from the attacker and the first "1010" packet of the genuine plug sent during the attack process, where $x < 20$. Denote y as the time interval between two consecutive "1010" datagram packets sent by the attacker. During the spoofing attack, assume that the total number of "1010" datagram packets sent by the genuine plug is n. Then the total time of this attack is $\lceil (\mathrm{x}+20*\mathrm{n})/\mathrm{y} \rceil *\mathrm{y}$. As shown in Figure 5, the shaded parts marked with solid lines and dash lines are controlled by the genuine plug, while the blank parts are controlled by the spoofed plug. The time duration corresponding to the shaded parts can be computed by

$$\sum_{i=0}^{n-1} \mathrm{T(i)} = \sum_{\mathrm{i}=0}^{\mathrm{n}-1} \min\left\{ \lceil (\mathrm{x}+20\mathrm{i})/\mathrm{y} \rceil \,\mathrm{y} - (\mathrm{x}+20\mathrm{i}), 20 \right\} \quad (1)$$

The time duration corresponding to the dash line shaded part can be computed by

$$\mathrm{T(n)} = \lceil (\mathrm{x}+20\mathrm{n})/\mathrm{y} \rceil \,\mathrm{y} - (\mathrm{x}+20\mathrm{n}) \quad (2)$$

Hence, we can derive the average online rate of the genuine plug during the attack by

$$G = \int_0^{20} \frac{\sum_{i=0}^{n-1} \mathrm{T(i)} + \mathrm{T(n)}}{20 \lceil (\mathrm{x}+20\mathrm{n})/\mathrm{y} \rceil \,\mathrm{y}} dx, \ (\mathrm{n} \geq 1) \quad (3)$$

We define the success rate of this attack as the online time of the spoofed plug over the length of the attack process. The success rate evaluates the probability that the spoofed plug gets a victim's credential when the victim randomly wakes up and sends a command during the attack process. Therefore, success rate S can be computed as follows,

$$S = 1 - G. \quad (4)$$

We will evaluate the success rate in Section V.

D. Firmware attack

The attacker can install a malicious firmware on the smart plug so that she can remotely control it. Once the malicious firmware is installed to the plug, it can establish a reverse tunnel back to a remote malicious server and open a reverse shell. Therefore, the attacker can remotely access the plug system and perform further attacks, e.g. installing various malware into the plug. In this attack, we assume the attacker can access the local network of the plug and monitor the traffic between plug and controller so as to derive the encoded WiFi username and password in the HTTP header as presented in **STEP 7b**. The attacker can then leverage the username and password for the authentication purpose and upload the malicious firmware to the HTTP server on the plug as illustrated in III-D.

The attacker is capable of modifying the firmware in order to add the malicious functionality since the vendor of the smart plug provides the open source code of the firmware. We find that some of the crucial functionalities of the plug, e.g., the light weight HTTP server and communication protocol of the plug, is prebuilt so as to hide the plug communication protocol and functionalities to some extent. Moreover, we find

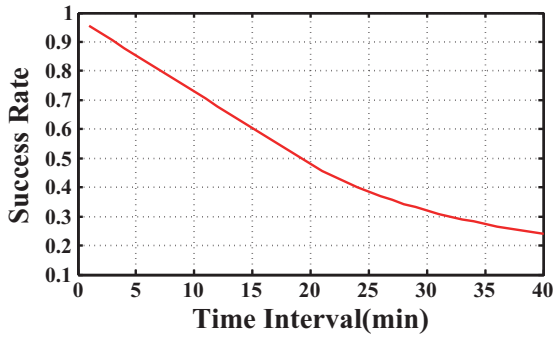

Fig. 6. Relationship between the success rate and the time interval

that *BusyBox* is used to provide some basic Linux tools and its source code is available. As a result, we can reconfigure *BusyBox* to enable *Netcat*. To establish a reverse tunnel to the attacker's desired server, she can utilize a *Netcat* command like *nc* [*IP address*] [*port*]-*e /bin/sh*, where the IP address and the port are those of the attacker's remote server.

The attacker can embed a piece of malicious code into the source code of the DHCP service so that the DHCP can execute the malicious command of *Netcat* at startup. We find that the system of the plug uses the DHCP service provided by BusyBox to assign an IP address to the associated controller. Consequently, the attacker can modify the source code of the DHCP service to add the malicious code and then recompile the source code of the firmware. In this way, the attacker can have a customized firmware and upload it to the HTTP server of the plug. The plug will automatically install the malicious firmware and restart the system. The DHCP service automatically starts at the boot up time and executes the malicious command of *Netcat*.

V. Evaluation

We have implemented the four attacks introduced in Section IV and performed real-world experiments to demonstrate the feasibility and effectiveness of the attacks against the Edimax plug of interest. In this section, we first introduce the experiment setup and then present the experimental results.

A. Experiment Setup

We deployed 5 plugs and connected them to the Internet via wireless routers. Three plugs were deployed on a university campus in North America while the rest were deployed in Asia. iPads are installed with the plug control app and are used as the controller. Our attack program was implemented in Python.

B. Experimental Results

We first test the scanning attack on the 5 plugs. Two plugs use the default password, i.e. "1234", two plugs use non-default passwords, and the fifth plug is not connected to the Internet. Our Python controller program sends the "1030" packet to the authentication server every 10 seconds if the program cannot obtain a response packet within 20 seconds. The total number of transmissions is limited to 5 times. From our experimental results, we can obtain the correct response from the two plugs using the default password and obtain the "5000" packet when scanning the offline plug. It is also observed that we do not get response packets from the plugs that do not use the default password. These results verify the findings in Section IV-A.

We then perform the brute force attack against the two plugs not using the default password. Since the default password "1234" includes only numbers and the length is 4, it may mislead users to set a 4-digit passcode. Therefore, we give the two plugs a password of four random numbers. Our Python controller program performs the brute force attack and we can get the right response within several minutes. This demonstrates the feasibility of the brute force attack. Recall that as a matter of fact, the maximum length of the password is 20. The brute force attack would be ineffective if such long passwords are used for plugs.

Finally, we perform the device spoofing attack using our own plugs in case that long passwords are used for plugs and the brute force attack does not work practically. We evaluate the feasibility of this attack in two scenarios: (i) the plug and the iPad located in the same network and (ii) the plug and the iPad located in different networks. The experimental results show that, in both scenarios, our Python program that works as a spoofed plug, denoted as Python plug, can obtain the credential from the controller as described in Section IV-C whatever the password is.

Figure 6 illustrates the relationship between the success rate and the time interval of sending the "1010" packets by our Python plug. As shown in Figure 6, the faster the transmission frequency, the higher the success rate. For instance, if the time interval of sending the "1010" packets is 3 minutes, the chance of successfully obtaining the credential from the controller is above 90%.

We are able to perform the firmware attack as introduced in Section IV. The firmware can be customized with various applications including netcat. With such a malicious firmware, an attacker can create a reverse tunnel from the plug to a desired server and gain the root access on the plug system. However, this firmware attack has to be deployed locally at this time.

VI. Defense Strategies

In this section, we present guidelines of the defense strategies to mitigate the risks from the Edimax plug vulnerabilities exposed in this paper.

A. Secure Communication Protocol

Cryptography has to be employed to encrypt communication. Encoding and obfuscation are not enough to provide secret communication. In this paper, we can see that an attacker can crack the obfuscation algorithm by analyzing the network traffic. With an eavesdropping attack, she can observe all the plaintext transmitted between the plugs and the controller. To mitigate these threats, secure communication protocols should be adopted, e.g., DTLS, TLS/SSL, and HTTPS, to encrypt the content transmitted between the plug, the controller, the authentication server, and the command relay server.

B. Mutual Authentication between Plugs and Servers

The spoofing attack stems from the fact that the authentication server does not authenticate the genuine plug. The attacker only needs to send a legitimate datagram using a command type of "1010" and the MAC address of the victim's plug in order to fool the authentication server. The device authentication mechanism should be adopted at both the server side and the plug side. For example, the device vendor can assign a public/private key pair to a device before it leaves the factory. The authentication server hosts a database of public keys of all the plugs. Therefore, the authentication server can adopt the public-key authentication to authenticate the genuine devices.

There is a possibility of spoofed server attacks against the authentication server and relay server. An attacker may employ DNS poisoning or man-in-the-middle attacks and pretend to be the two servers. To counter the attack, plugs and control apps should be pre-installed with public keys of the two servers and verity certificates of the two servers before transmitting authentication credentials and data.

C. Intrusion Detection System

To thwart the scanning attack, an intrusion detection system should be employed at the server side. The intrusion detection system should be able to identify extensive scanning attacks. For example, it should detect the continuous and rapid password attempts. Moreover, the intrusion detection system can be used to detect abnormal behaviors. For instance, during the spoofing attack, the authentication server can identify the attack by simply tracking the geolocation of the registered plugs. If the geolocation information shows that two consecutive physical locations of a registered plug is far away in a short time period, the spoofing attack may be underway.

D. Anti-bot Mechanisms

To prevent the brute force attack, the authentication server should adopt methods to determine if the login is performed by a human or a bot. For instance, the CAPTCHA can be used to mitigate the brute force attack conducted with bots. Limiting the number of login attempts can be an effective way to prevent this type of attack.

E. Data Integrity

According to our experiments, we can change the IP address of the rendezvous server, i.e., the command relay server. Recall that the authentication server generates the IP address of the rendezvous server and the relay ID in **STEP 6a**. If the message is received by our spoofed plug, we can modify the IP address of the rendezvous server and send the message back to the authentication server. The server does not check the data integrity. As a result, the controller receives the IP address of our desired server and then establish a TCP connection to our server. This attack is possible since an attacker can tamper with the data from the authentication server at the spoofed plug side. Message authentication codes should be adopted.

VII. Related Work

IoT systems are similar to traditional information systems that consist of software, hardware, data, communication, and end users. Therefore, IoT systems are subject to the similar set of attacks against traditional information systems. In a typical smart home automation system [4], the components such as the smart device, house gateway, cloud server, API, mobile device, and application, may all cause security and privacy problems. Existing work relevant to our study roughly fall into the following categories: software-related attacks, hardware-level attacks, data-related attacks, communication-related attacks, and end users-related attacks.

Software-related security issues [7], [8] are similar to the traditional computer systems. For example, a buffer overflow exploit is found by analyzing Home Network Administration Protocol (HNAP) [7] so that it can be used to execute any code on the device. A stack-based buffer overflow of the general library, glibc [9], is exploited to attack several home hubs [8].

Hardware-level attacks [10]–[12] concentrate on compromising the hardware, e.g., tampering external flash memories and glitching address lines. For instance, Hernandez *et al.* [10] use a USB stick connecting to a home automation device, the Nest Thermostat, so that the device will load the code stored on the stick without any check. Thus, an attacker could exploit the security hole to install a malware on the device. Ly *et al.* [11] study the Itron Centron smart meter. They found that the device ID is stored in an external EEPROM that does not provide read/write protection. By rewriting the ID on the EEPROM, they can use one meter to forge another meter.

Data-related security problems are investigated as well [13]–[17]. For example, Ronen *et al.* [16] exploit the smart lights to establish a covert light channel to leak data from a secure place. The receiver could be deployed at a long distance.

Communication-related security vulnerabilities have also been substantially studied [3], [18]–[24]. For instance, Rouf *et al.* [18] reverse-engineer the unsecured wireless communication protocol of automatic meter reading and discovered the lack of security mechanisms to protect user security and privacy. Dhanjani [19] hacks the Phillips Hue lightbulb system and finds that the authentication mechanisms are not strong. Molina [20] exploits the KNX, a standardized home automation communication protocol, and finds that the lack of authentication and encryption allows an attacker to remotely control the appliances in a hotel. Rahman *et al.* [21] finds the communication protocol vulnerabilities of the wearable device (Fitbit). Various attacks, e.g., eavesdropping and injection, could be performed to impair the security and privacy of the victim. By automatically analyzing the applications and forging the authentication messages, Zuo *et al.* [24] design an authentication message generator to perform brute force attacks against the corresponding remote application server. Obermaier and Hutle [22] investigate the vulnerabilities of communication protocols of four surveillance camera systems.

End users-related security threats often come from various side channel attacks, e.g., vision-based attacks [25], [26] and residues-based attacks [27], [28], to obtain users' passwords. For example, Yue *et al.* [26] investigate attacks that can capture

a victim's password without observing the text on the screen of a smart device. Zhang *et al.* [28] utilize the fingerprint powder to derive the tapped keys on the surface of a mobile device and infer the victim's password.

To mitigate these threats, researchers also propose various guidelines [29]–[32]. For example, an overview of security and privacy of cyber physical systems can be found in [32]. A security architecture [33]–[35] for IoT systems can be used to provide comprehensive security protection. Secure hardware [36] and trusted [37], [38] software can be applied to provide data integrity verification to defend against malware. The security of end users can be significantly improved by educating users to employ a secure input method [26] to enhance the interface security between human and IoT systems. To address the device authentication issues, the device fingerprinting [39], [40] from different layers could be leveraged to identify the genuine smart device for fraud protection.

VIII. Conclusion

In this paper, we send out a strong warning message on the security problems of Edimax plug system and hope that Edimax plug and other IoT device manufacturers enhance the security of their systems. We study the vulnerabilities of a smart plug system by reverse engineering its communication protocols. After we obtain the details of its communication protocols, we are able to identify several security vulnerabilities, including insecure communication protocols, lack of device authentication, and a weak password policy. We propose four attacks, device scanning attack, brute force attack, device spoofing attack, and firmware attack, to demonstrate the severity of these security risks. We have implemented these attacks and performed real-world experiments. Our analysis and experimental results show that an attacker is able to control these smart plugs completely. The device scanning attack can find all online plugs. The brute force attack and device spoofing attack can obtain the device password whatever it is. The firmware attack can obtain the root access on the plug system. To thwart these serious threats, we present the guidelines for corresponding countermeasures.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China under grants 61502100, 61402104, 61572130, 61632008, 61602111, 61532013, and 61320106007, by US NSF grants 1461060, 1642124, and 1547428, by the Natural Sciences and Engineering Research Council of Canada under the grants 261409-2013, by Jiangsu Provincial Natural Science Foundation of China under Grant BK20150637 and BK20140648, by Jiangsu Provincial Key Laboratory of Network and Information Security under grants BM2003201, by Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under grants 93K-9 and by Collaborative Innovation Center of Novel Software Technology and Industrialization. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Gartner, "8.4 billion connected 'things' will be in use in 2017, up 31 percent from 2016," http://www.gartner.com/newsroom/id/3598917, February 2017.

[2] P. Siano, "Demand response and smart grids - A survey," *Renewable and Sustainable Energy Reviews*, vol. 30, pp. 461–478, 2014.

[3] Mario Ballano Barcena and Candid Wueest, "Insecurity in the internet of things," https://www.symantec.com/content/dam/symantec/docs/white-papers/insecurity-in-the-internet-of-things-en.pdf, 2015.

[4] A. Jacobsson, M. Boldt, and B. Carlsson, "A risk analysis of a smart home automation system," *Future Generation Computer Systems*, 2016.

[5] Google, "Known disruptions of traffic to google products and services," https://www.google.com/transparencyreport/traffic/disruptions/#group=REGION, August 2016.

[6] A. John, "Mac address and oui lookup," http://aruljohn.com/mac.pl, August 2016.

[7] /DEV/TTYS0, "Hacking the d-link dsp-w215 smart plug," http://www.devttys0.com/2014/05/hacking-the-d-link-dsp-w215-smart-plug/, 2014.

[8] M. Smith, "Security holes in the 3 most popular smart home hubs and honeywell tuxedo touch," http://www.networkworld.com/article/2952718/microsoftsubnet/security-holes-in-the-3-most-popular-smart-home-hubsand-honeywell-tuxedo-touch.html, 2015.

[9] "Critical security flaw: glibc stack-based buffer overflow in getaddrinfo() (cve-2015-7547)," https://access.redhat.com/articles/2161461, 2015.

[10] G. Hernandez, O. Arias, D. Buentello, and Y. Jin, "Smart nest thermostat: A smart spy in your home," in *Proceedings of the Black Hat USA*, 2014.

[11] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin, "Security analysis on consumer and industrial iot devices," in *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC*, 2016.

[12] O. Arias, J. Wurm, K. Hoang, and Y. Jin, "Privacy and security in internet of things and wearable devices," *IEEE Transactions on Multi-Scale Computing Systems*, 2015.

[13] J. Lin, W. Yu, X. Yang, G. Xu, and W. Zhao, "On false data injection attacks against distributed energy routing in smart grid," in *Proceedings of the 3rd ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2012.

[14] Q. Yang, J. Yang, D. A. W. Yu, N. Zhang, and W. Zhao, "On false data-injection attacks against power system state estimation: Modeling and countermeasures," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2014.

[15] J. Lin, W. Yu, and X. Yang, "On false data injection attack against multistep electricity price in electricity market in smart grid," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2016.

[16] E. Ronen and A. Shamir, "Extended functionality attacks on iot devices: The case of smart lights," in *Proceedings of IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.

[17] Q. Yang, J. Yang, D. A. W. Yu, N. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, 2017.

[18] I. Rouf, H. Mustafa, M. Xu, W. Xu, R. Miller, and M. Gruteser, "Neighborhood watch: Security and privacy analysis of automatic meter reading systems," in *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS)*, 2012.

[19] N. Dhanjani, "Security evaluation of the philips hue personal wireless lighting system," http://www.dhanjani.com/docs/Hacking%20Lighbulbs%20Hue%20Dhanjani%202013.pdf, 2013.

[20] J. Molina, "Learn how to control every room at a luxury hotel remotely," https://www.defcon.org/images/defcon-22/dc-22-presentations/Molina/DEFCON-22-Jesus-Molina-Learn-how-to-control-every-room-WP.pdf, 2014.

[21] M. Rahman, B. Carbunar, and M. Banik, "Fit and vulnerable: Attacks and defenses for a health monitoring device," in *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)*, 2013.

[22] J. Obermaier and M. Hutle, "Analyzing the security and privacy of cloud-based video surveillance systems," in *Proceedings of the 2nd*

ACM International Workshop on IoT Privacy, Trust, and Security (IoTPTS), 2016.

[23] H. Li, Z. Xu, H. Zhu, D. Ma, S. Li, and K. Xing, "Demographics inference through wi-fi network traffic analysis," in *Proceedings of the 35th IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[24] C. Zuo, W. Wang, R. Wang, and Z. Lin, "Automatic forgery of cryptographically consistent messages to identify security vulnerabilities in mobile services," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[25] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero, "A fast eavesdropping attack against touchscreens," in *Proceedings of the 7th International Conference Information Assurance and Security (IAS)*, 2011.

[26] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao, "Blind recognition of touched keys on mobile devices," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[27] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge attacks on smartphone touch screens," in *Proceedings of Workshop on Offensive Technology WOOT*, 2010.

[28] Y. Zhang, P. Xia, J. Luo, Z. Ling, B. Liu, and X. Fu, "Fingerprint attack against touch-enabled devices," in *Proceedings of the 2nd Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2012.

[29] R. Romana, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Computer Networks*, 2013.

[30] J. S. Kumar and D. R. Patel, "A survey on internet of things: Security and privacy issues," *International Journal of Computer Applications*, 2014.

[31] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in internet of things: The road ahead," *Computer Networks*, 2015.

[32] H. Song, G. A. Fink, and S. Jeschke, "Security and privacy in cyber-physical systems: Foundations, principles and applications," *Chichester, UK: Wiley-IEEE Press*, 2017.

[33] J. Noorman and P. Agten and W. Daniels and R. Strackx and A. Van Herrewege and C. Huygens and B. Preneel and I. Verbauwhede and F. Piessens, "ancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base," in *Proceedings of USENIX Conference on Security (Security)*, 2013.

[34] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of European Conference on Computer Systems (EuroSys)*, 2014.

[35] F. Brasser and P. Koeberl and B. E. Mahjoub and A.-R. Sadeghi and C. Wachsmann, "Tytan: Tiny trust anchor for tiny devices," in *Proceedings of Design Automation Conference (DAC)*, 2015.

[36] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "Smart: Secure and minimal architecture for (establishing a dynamic) root of trust," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2012.

[37] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

[38] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, "Cuba: Secure code update by attestation in sensor networks," in *Proceedings of ACM Workshop on Wireless Security (WiSec)*, 2006.

[39] A. Bates and R. Leonard and H. Pruse and D. Lowd and K. Butler, "Leveraging usb to establish host identity using commodity devices," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.

[40] D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. Beyah, "Who's in control of your control system? device fingerprinting for cyber-physical systems," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

Zhen Ling is an assistant professor in the School of Computer Science and Engineering at the Southeast University, Nanjing, China. He received the B.S. degree (2005) and Ph.D. degree (2014) in Computer Science from Nanjing Institute of Technology, China and Southeast University, China, respectively. He joined Department of Computer Science at the City University of Hong Kong from 2008 to 2009 as a research associate, and then joined Department of Computer Science at the University of Victoria from 2011 to 2013 as a visiting scholar. His research interests include network security, privacy, and Internet of Things.

Junzhou Luo is a full Professor in the School of Computer Science and Engineering, Southeast University, Nanjing, China. He received his B.S. degree in applied mathematics from Southeast University in 1982, and then got his M.S. and Ph.D. degree in computer network both from Southeast University in 1992 and in 2000 respectively. His research interests are next generation network, protocol engineering, network security and management, grid and cloud computing, and wireless LAN. He is a member of the IEEE Computer Society and co-chair of IEEE SMC Technical Committee on Computer Supported Cooperative Work in Design.

Yiling Xu received the B.S. degree in digital media technology from Jiangnan University, Wuxi, China, in 2016. Currently, she is working toward the master degree in computer science and engineering at Southeast University, Nanjing, China. Her current research interests include Internet of Things, privacy, and security.

Chao Gao received the B.S. degree in electrical engineering from Xian Jiaotong University, Xian, China, in 2011 and the M.S. degree in electrical and computer engineering from Northeastern University, Boston, MA, USA in 2015, respectively. She is currently working toward the Ph.D. degree in computer science at University of Massachusetts Lowell, Lowell, MA, USA. Her current research interests include Internet of Things and network security and privacy.

Kui Wu received the B.Sc. and the M.Sc. degrees in Computer Science from Wuhan University, China in 1990 and 1993, respectively, and the Ph.D. degree in Computing Science from the University of Alberta, Canada, in 2002. He joined the Department of Computer Science at the University of Victoria, Canada in 2002 and is currently a Professor there. His research interests cover network performance analysis, online social networks, Internet of Things, and parallel and distributed algorithms. He is a senior member of IEEE.

Xinwen Fu received the B.S. and M.S. degrees in electrical engineering from Xian Jiaotong University, China and University of Science and Technology of China, in 1995 and 1998, respectively. He obtained Ph.D. degree in computer engineering from Texas A&M University, College Station, in 2005. He is an associate professor in the Department of Computer Science, University of Massachusetts Lowell. His current research interests include network security and privacy, digital forensics, wireless networks, and network QoS. His research was reported by various media such as Wired and aired on CNN and CCTV 10.