# Teardown of the TP-Link Kasa Smart Wi-Fi Plug KP115

Benjamin Simmonds (5233344), UNSW Canberra

July 2021

**Abstract**

As the Internet of Things (IoT) continues to take the world by storm with an estimated deployment footprint of 20 billion globally, security of these devices is a major area of concern, with direct impacts to economic and citizen safety. To better understand the severity of the situation, this paper presents the findings of a security analysis of a contemporary Smart Plug device purchased only weeks ago from Bunnings Warehouse. The results were alarming, a useless encryption scheme combined with no authentication, makes it a trivial exercise to gain complete control over the device. Lack of regulations in this space, combined with strong economic drivers for manufacturers to push IoT devices to the mass market quickly may be contributing factors.

## Contents

## Introduction

This paper steps through a “tear down” (a methodical decomposition of a system) of the TP-Link (Kasa) Smart Wi-Fi Plug Slim model KP115, which was purchased from Bunnings for about $20 in July 2021.

Goals and questions:

- Evaluate firmware and software makeup
- Analyse the app-to-device network communication employed
  - Specific protocols and ports
  - The onboarding or discovery process
  - The use of any WAN (cloud) based services
- Assess any potential security mechanisms used, such as encryption
- If any are found to exist, practically demonstrate vulnerabilities

Figure 1: The TP-Link/Kasa Smart Plug under assessment (Co (2021a))

TP-Link is the umbrella company of Kasa, a consumer device manufacturer with a focus on IoT connected “smart devices”.

Some basic reconnaissance (TP-Link 2021) shows the “Kasa Smart Wi-Fi Plug Slim” is an actively supported product line and markets many consumer desirable qualities such as ease of use and setup, controllable from anywhere, voice control, an integrated timer for scheduling and even boasts “Multiple Safety Guarantees”.

## Aim

## Methodology

The security assessment was structured based on the lifecycle of an industry based offensive methodology, by MITRE Corporation (Corporation (2021)).

1. Reconnaissance: research on functionality, software eco-system available, documentation or patents on protocols, similar product lines, CVE databases
2. Resource development: create cloud account, install the android app, understand base functions, reverse engineer android app, design test bench
3. Initial access: establish connectivity to the smart plug via rogue access point (evil twin)
4. Discovery: network service scanning
5. Collection: MITM AP network sniffing
6. Command and control: reverse engineer the TP-Link Smart Home protocol (TSHP) and fabricate commands

The specifications (Co (2021c)) highlight the device operates over 2.4GHz 802.11 b/g/n. This eliminated use of Bluetooth related protocols, focusing the design for intercepting network traffic to Wi-Fi only. It turns out a majority of IoT devices deployed in the wild are Wi-Fi based. One theory is that this is driven by cost saving measures by manufacturers, due to the ubiquity and affordability of Wi-Fi (Junior et al. (2019)).

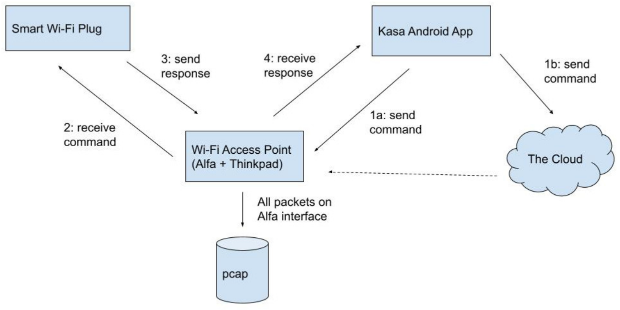


Figure 2: High level design to intercept all network activity

## Tools and Equipment

The security assessment made use of the following.

Equipment:

- A Kasa Smart Wi-Fi Plug Slim (KP115) – i.e., the device under evaluation
- Thinkpad X220 running Arch Linux
- Alfa AWUS036NHA USB Wi-Fi interface
- Google Pixel 4A Android Smartphone

Tools and software:

- The official *Kasa Smart* Android App
- A copy of the firmware `HS110(US)_V1_151016.zip` running on the device (Co (2021b))
- Wireshark and tcpdump
- binwalk
- `tplink-smarthome.lua`: a custom wireshark LUA dissector (Stroetmann and Esser (2018b))

- `tplink_smartplug.py`: a proof of concept that implements the weak XOR autokey encryption used by the TP-Link Smart Home protocol (TSHP) called (Stroetmann and Esser (2018c))
- Firmware for the Atheros family of network devices (for the Alfa)

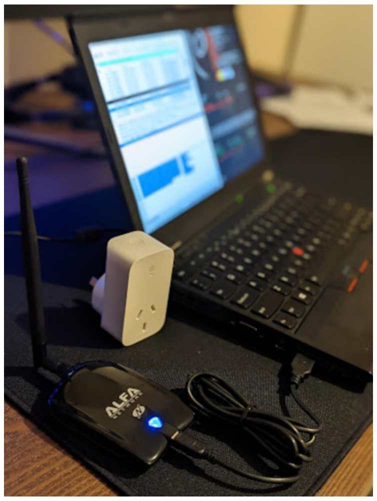


Figure 3: Equipment used (from left to right) an Alfa AWUS036NHA, the Kasa Smart Plug KP115 and Linux ThinkPad

After configuring the Atheros OS driver for the Alfa card, a Wi-Fi hotspot named "zeit8023" was setup:

```
$ nmcli dev wifi hotspot ifname wlx00c0ca991c64 ssid zeit8023 password 13371337
```

```
$ nmcli d
DEVICE                   TYPE      STATE                    CONNECTION
wlx00c0ca991c64          wifi      connected                Hotspot
enp0s25                  ethernet  unavailable              --
lo                       loopback  unmanaged                --
wlp3s0                   wifi      unmanaged                --
```

Having installed the "Kasa Smart" Android app from the Google play store on my Google Pixel 4A, the Google Pixel 4A Smartphone was connected to the hotspot, and TP-Link Smart Plug plugged into a power socket. The smart plug booted up automatically, and launched the pairing process.

On the ThinkPad, I configured Wireshark to profile all traffic flowing over the Alfa card. Using the "Kasa Smart" app on my Android I performed the following functions:

1. Switched the smart plug on.
2. Switched the smart plug off.
3. Switched the smart plug on.
4. Created a schedule to turn the switch off at 20:45
5. Switched the smart plug on.
6. Switched the smart plug off.

## Observations

Some high-level facts that were immediately drawn from the network capture:

- The MAC address of the smart plug is `c0:06:c3:09:e0:61`
- The IP address assigned to the smart plug is `10.42.0.126`
- The IP address assigned to the Android Pixel Smartphone is `10.42.0.202`

## Port scan

After capturing the network trace, a port scan was conducted against the smart plug device, which revealed port 9999 was open – see Annex: C nmap scan results.

Analysing the port 9999 related traffic in Wireshark revealed that many TCP conversations were taking place between the Android Smartphone (the app) and the TP-Link Smart Plug. The message data appeared to contain indecipherable binary data. Using CyberChef I attempted to decode the binary data using several common encoding and encryption methods, to no avail.

## TP-Link Device Firmware and Server Daemon

Running `binwalk` on a copy of the firmware running on the smart plug (Co (2021b)), showed it was running a Linux 2.6 based kernel, and several other popular embedded sub-systems:

- U-Boot Bootloader 1.1.4
- Linux Kernel 2.6.31
- A squashfs based filesystem

Examining the contents of the filesystem, discovered the following files of interest:

- /bin/busybox v1.01 (2015.10.16-03:17+0000)
- /usr/bin/shd

It turns out (Stroetmann and Esser (2018a)) that `/usr/bin/shd` is the daemon server process responsible for handling the port 9999 related traffic, and is a C binary:

```
shd: ELF 32-bit MSB executable, MIPS, MIPS32 rel2 version 1 (SYSV),
dynamically linked, interpreter /lib/ld-uClibc.so.0, corrupted section header size
```

At this point I decided to abandon reversing the smart plug shd server process any further.

## Decompiled Kasa Smart Android App

Given the app is the command center for all its connected smart devices, I conceptualised that it would need to encrypt the "command data" in whatever scheme was being used.

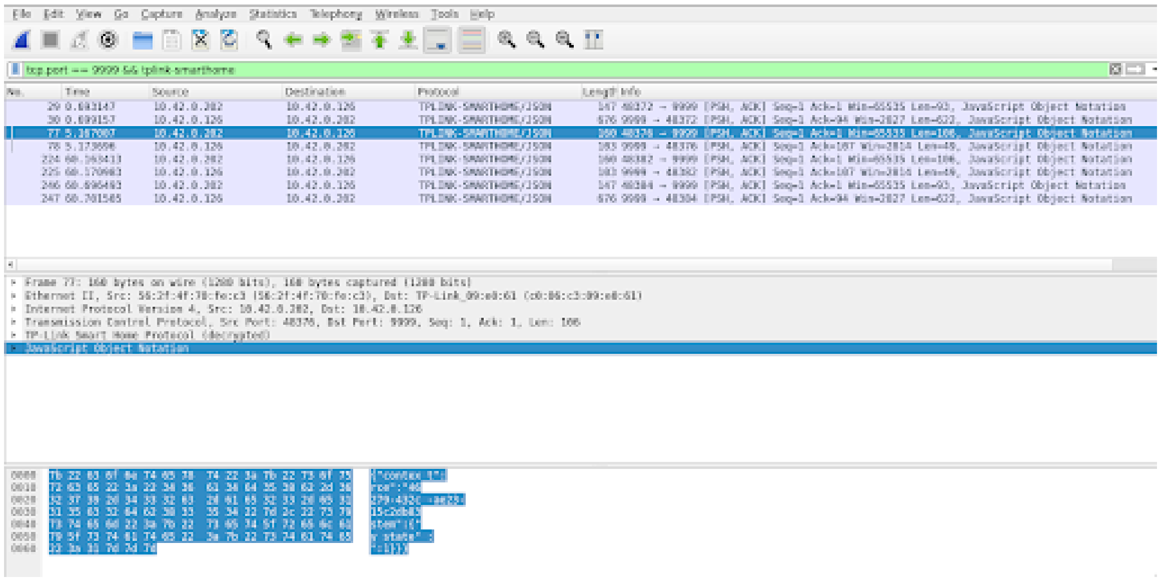
Using "Apk Extractor" on the Google Pixel Smartphone, I was able to extract the "Kasa Smart" app's apk package, and it turn it's Java bytecode. With some help of security researchers (Stroetmann and Esser (2018a)) I quickly discovered the key encryption function:

```
public static byte[] m7377b(byte[] bArr) {
    if (bArr != null && bArr.length > 0) {
        int i = -85;
        for (int i2 = 0; i2 < bArr.length; i2++) {
            byte b = (byte) (i ^ bArr[i2]);
            i = bArr[i2];
            bArr[i2] = b;
        }
    }
    return bArr;
}
```

This simple encryption function uses the XOR bitwise operator (^) and washes each byte of the "command data", starting with the hardcoded IV value of -85 (or 171). Each iteration, the next cleartext data byte is XOR'ed with the previous cleartext byte (i.e. i = bArr[i2]). Involving the actual data as the key is known as an autokey cipher (Foundation (2021)).

Given the use of a hardcoded nonce (or IV), this process can easily be reversed to decrypt the message data.

As a proof of concept, a Python implementation `tplink_smartplug.py` and custom Wireshark LUA dissector tplink-smarthome.lua (Stroetmann and Esser (2018a)) was used to conveniently decrypt and analyse the autokey encrypted conversations taking place between the app and the smart plug.



```
tcp.port == 9999 && tplink-smarthome

No.  Time        Source       Destination  Protocol              Length Info
  29 0.693147    10.42.0.202  10.42.0.126  TPLINK-SMARTHOME/JSON  147 48372 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=93, JavaScript Object Notation
  30 0.699157    10.42.0.126  10.42.0.202  TPLINK-SMARTHOME/JSON  676 9999 → 48372 [PSH, ACK] Seq=1 Ack=94 Win=2027 Len=622, JavaScript Object Notation
  77 5.167087    10.42.0.202  10.42.0.126  TPLINK-SMARTHOME/JSON  160 48376 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=106, JavaScript Object Notation
  78 5.173696    10.42.0.126  10.42.0.202  TPLINK-SMARTHOME/JSON  103 9999 → 48376 [PSH, ACK] Seq=1 Ack=107 Win=2814 Len=49, JavaScript Object Notation
 224 60.165413   10.42.0.202  10.42.0.126  TPLINK-SMARTHOME/JSON  160 48382 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=106, JavaScript Object Notation
 225 60.170983   10.42.0.126  10.42.0.202  TPLINK-SMARTHOME/JSON  103 9999 → 48382 [PSH, ACK] Seq=1 Ack=107 Win=2814 Len=49, JavaScript Object Notation
 246 60.696483   10.42.0.202  10.42.0.126  TPLINK-SMARTHOME/JSON  147 48384 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=93, JavaScript Object Notation
 247 60.701585   10.42.0.126  10.42.0.202  TPLINK-SMARTHOME/JSON  676 9999 → 48384 [PSH, ACK] Seq=1 Ack=94 Win=2027 Len=622, JavaScript Object Notation

▸ Frame 77: 160 bytes on wire (1280 bits), 160 bytes captured (1280 bits)
▸ Ethernet II, Src: 56:2f:4f:70:fe:c3 (56:2f:4f:70:fe:c3), Dst: TP-Link_09:e8:61 (c0:06:c3:09:e8:61)
▸ Internet Protocol Version 4, Src: 10.42.0.202, Dst: 10.42.0.126
▸ Transmission Control Protocol, Src Port: 48376, Dst Port: 9999, Seq: 1, Ack: 1, Len: 106
▸ TP-Link Smart Home Protocol (decrypted)
▸ JavaScript Object Notation

0000  7b 22 63 6f 6e 74 65 78  74 22 3a 7b 22 73 6f 75   {"contex t":
0010  72 63 65 22 3a 22 34 36  61 34 64 35 38 62 2d 36   rce":"46
0020  32 37 39 2d 34 33 32 63  2d 61 65 32 33 2d 65 31   279-432c-ae23-
0030  31 35 63 32 64 62 38 33  35 34 32 7d 2c 22 73 79   15c2db8
0040  73 74 65 6d 22 3a 7b 22  73 65 74 5f 72 65 6c 61   stem":{"
0050  79 5f 73 74 61 74 65 22  3a 7b 22 73 74 61 74 65   y_state":
0060  22 3a 31 7d 7d 7d                                  ":1}}}
```

Figure 4: Wireshark analysis using custom LUA dissector

Now able to decipher the commands, set out to enumerated each function that was invoked using the smartphone app, and the resulting cleartext JSON message. See Annex: A Enumeration of TP-Link Smart Home Protocol Commands.

Finally, an end-to-end demonstration of just how weak security (essentially non-existent) measures are, invoked each function of the smart plug using Python. See Annex: B Programmatic Control using Python Script.

## Conclusion

The TP-Link (Kasa) Smart Plug demonstrates the delicate balance the industry currently faces towards providing consumer convenient connected technology (i.e., IoT) while in a safe and secure manner. As this paper shows, the security concerns are real.

On one end of the spectrum IoT manufacturers are under aggressive pressure to push products to market with the hope of soaking up as much market share as possible, taking paths of least resistance to "just get the job done". On the other end, consumers, small and medium sized businesses are the ones left exposed due to under-baked products.

What deems an appropriate level of "safe and secure", what frameworks or standards could be applied without burdening technology companies with unrealistic overheads while providing consumer comprehendible advice. Mr Farrell linked (Home Affairs (2021)) to the thought-provoking initiative the Australian government is actively working through, to address these exact issues.

## Appendix A: Enumeration of TP-Link Smart Home Protocol Commands

Switch on request:

```
{
    "context": { "source": "46a4d58b-6279-432c-ae23-e115c2db8354" },
    "system": { "set_relay_state": { "state": 1 } }
}
```

Switch off request:

```
{
    "context": { "source": "46a4d58b-6279-432c-ae23-e115c2db8354" },
    "system": { "set_relay_state": { "state": 0 } }
}
```

Scheduled “turn off” event:

```
{
    "schedule": {
        "get_rules": {
            "rule_list": [
                {
                    "id": "689CB20F436CEDEE16F8752FDFD26EBC",
                    "name": "Schedule Rule",
                    "enable": 1,
                    "wday": [1, 1, 1, 1, 1, 1, 1],
                    "stime_opt": 0,
                    "smin": 1245,
                    "sact": 0,
                    "eact": -1,
                    "repeat": 1
                }
            ],
            "version": 2,
            "enable": 1,
            "err_code": 0
        }
    }
}
```

System status response:

```
{
    "system": {
        "get_sysinfo": {
            "sw_ver": "1.0.17 Build 210506 Rel.075231",
            "hw_ver": "1.0",
            "model": "KP115(AU)",
            "deviceId": "8006D8042A39366BDFDD6E65504D85FF1E64FC77",
            "oemId": "33AF6F102C6172F2E5BD6CEA656C19FF",
            "hwId": "8EDCE7F6CF8231678B90A6730D02BB41",
            "rssi": -34,
            "latitude_i": -354348,
            "longitude_i": 1491063,
            "alias": "Kasa smart plug",
            "status": "new",
            "obd_src": "tplink",
            "mic_type": "IOT.SMARTPLUGSWITCH",
            "feature": "TIM:ENE",
            "mac": "C0:06:C3:09:E0:61",
            "updating": 0,
            "led_off": 0,
            "relay_state": 1,
            "on_time": 31,
            "icon_hash": "",
            "dev_name": "Smart Wi-Fi Plug Mini",
            "active_mode": "schedule",
            "next_action": {
                "type": 1,
                "id": "689CB20F436CEDEE16F8752FDFD26EBC",
                "schd_sec": 74700,
                "action": 0
            },
            "ntc_state": 0,
            "err_code": 0
        }
    }
}
```

## Appendix B: Programmatic Control using Python Script

### Switch on

```
$ python3 ./tplink_smartplug.py -t 10.42.0.126 -c on
Sent:      {"system":{"set_relay_state":{"state":1}}}
Received:  {"system":{"set_relay_state":{"err_code":0}}}
```

### Switch off

```
$ python3 ./tplink_smartplug.py -t 10.42.0.126 -c off
Sent:      {"system":{"set_relay_state":{"state":0}}}
Received:  {"system":{"set_relay_state":{"err_code":0}}}
```

### Report energy use metrics

```
$ python3 ./tplink_smartplug.py -t 10.42.0.126 -c energy
Sent:      {"emeter":{"get_realtime":{}}}
Received:  {"emeter":{"get_realtime":{"current_ma":0,"voltage_mv":242327,"power_mw":0,"total_wh":0,"
```

### Show cloud details

```
$ python3 ./tplink_smartplug.py -t 10.42.0.126 -c cloudinfo
Sent:      {"cnCloud":{"get_info":{}}}
Received:  {"cnCloud":{"get_info":{"username":"ben@bencode.net","server":"n-devs.tplinkcloud.com","b
```

### Run an access point (AP) scan

```
$ python3 ./tplink_smartplug.py -t 10.42.0.126 -c wlanscan
Sent:      {"netif":{"get_scaninfo":{"refresh":0}}}
Received:  {"netif":{"get_scaninfo":{"ap_list":[],"err_code":0}}}
```

### Show the time

```
$ python3 ./tplink_smartplug.py -t 10.42.0.126 -c time
Sent:      {"time":{"get_time":{}}}
Received:  {"time":{"get_time":{"year":2021,"month":9,"mday":9,"hour":21,"min":9,"sec":42,"err_code"
```

### List general information (including firmware version)

```
$ python3 ./tplink_smartplug.py -t 10.42.0.126 -c info
Sent:      {"system":{"get_sysinfo":{}}}
Received:  {"system":{"get_sysinfo":{"sw_ver":"1.0.17 Build 210506 Rel.075231","hw_ver":"1.0","model
```

### List active schedules

```
$ python3 ./tplink_smartplug.py -t 10.42.0.126 -c schedule
Sent:      {"schedule":{"get_rules":{}}}
Received:  {"schedule":{"get_rules":{"rule_list":[{"id":"689CB20F436CEDEE16F8752FDFD26EBC","name":"S
```

## Appendix C: nmap scan results

```
nmap -v -sV 10.42.0.126
Starting Nmap 7.80 ( https://nmap.org ) at 2021-09-07 21:04 AEST
NSE: Loaded 45 scripts for scanning.
Initiating Ping Scan at 21:04
Scanning 10.42.0.126 [2 ports]
Completed Ping Scan at 21:04, 0.00s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 21:04
Completed Parallel DNS resolution of 1 host. at 21:04, 0.01s elapsed
Initiating Connect Scan at 21:04
Scanning 10.42.0.126 [1000 ports]
Discovered open port 9999/tcp on 10.42.0.126
Completed Connect Scan at 21:04, 1.84s elapsed (1000 total ports)
Initiating Service scan at 21:04
Scanning 1 service on 10.42.0.126
Completed Service scan at 21:04, 9.11s elapsed (1 service on 1 host)
NSE: Script scanning 10.42.0.126.
Initiating NSE at 21:04
Completed NSE at 21:04, 0.01s elapsed
Nmap scan report for 10.42.0.126
```

```
Host is up (0.054s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE VERSION
9999/tcp open  abyss?
```

## References

Co, TP-Link Technologies. 2021a. "Kasa Smart Wi-Fi Plug Slim Product Page." September 2021. https://www.tp-link.com/au/home-networking/smart-plug/kp105/.

———. 2021b. "TP-Link Smart Plug Firmware Hs110(US)_V1_151016.zip." September 2021. https://static.tp-link.com/res/down/soft/HS110(US)_V1_151016.zip.

———. 2021c. "TP-Link Smart Plugs Kp105 V2 Specifications." September 2021. https://www.tp-link.com/au/home-networking/smart-plug/kp105/#specifications.

Corporation, MITRE. 2021. "The MITRE ATT&CK Framework." January 2021. https://attack.mitre.org/.

Foundation, Wikimedia. 2021. "Autokey Cipher." September 2021. https://en.wikipedia.org/wiki/Autokey_cipher.

Home Affairs, The Department of. 2021. "Strengthening Australia's Cyber Security Regulations and Incentives." July 2021. https://www.homeaffairs.gov.au/reports-and-publications/submissions-and-discussion-papers/cyber-security-regulations-incentives.

Junior, Davino Mauro, Luis Melo, Hao Lu, Marcelo d'Amorim, and Atul Prakash. 2019. "A Study of Vulnerability Analysis of Popular Smart Devices Through Their Companion Apps." In *2019 IEEE Security and Privacy Workshops (SPW)*, 181–86. IEEE.

Stroetmann, L, and Esser T. 2018a. "Reverse Engineering the TP-Link Hs110." July 2018. https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/.

———. 2018b. "Tplink-Smarthome.lua." July 2018. https://github.com/softScheck/tplink-smartplug/blob/master/tplink-smarthome.lua.

———. 2018c. "Tplink_smartplug.py." July 2018. https://github.com/softScheck/tplink-smartplug/blob/master/tplink_smartplug.py.