

IoT Security Assessment Methodology

Sheridan | Get  
Creative

Honors Bachelor of Applied Information Sciences (Information Systems Security)

Revision 1 - Summer 2017

In Memory of Victor Ralevich

# I - PURPOSE

---

This methodology provides a standard process for assessing the security of Internet of Things (IoT) devices. The testing process has been split into categories that logically separate the phases of testing. Included are guidelines and specifications for setting up a testing environment. While this methodology aims to be comprehensive and of sufficient detail to be self-contained; it is assumed that the reader has sufficient Computer Science knowledge to understand and interpret the contents.

This document was produced in parallel to the testing of a small sample of best-selling IoT devices in order to produce and refine this process document. A course module has been created from this content for the eventual adoption into the Honours Bachelor of Applied Information Sciences (Information System Security) program. This module, intended for a fourth year audience, explores the practical aspects of this methodology in a more academic and open-ended format.

Finally the lab, as outlined, has been constructed for use and future development. Both the lab and this methodology are intended to be built upon and revised to accommodate changes to both the IoT landscape and the Sheridan infrastructure.

## II - AUTHORS

---

The Honours Bachelor of Applied Information Sciences (Information Systems Security) program at Sheridan College provides an eight month graduation project meant to encompass a substantive effort that embodies the knowledge gained by the culmination of the senior year. This methodology serves as a portion of the work effort undertaken by the authors during this time. We hope that the IoT testing lab, this methodology, and the course module we have developed serve as a foundation for continued research and advocacy for security of the Internet of Things.

*Nebojsa Bajagic*  
*Sheridan College, Oakville, Ontario, Canada*  
*bajagicn@sheridancollege.ca*

*David Sigmundson*  
*Sheridan College, Oakville, Ontario, Canada*  
*sigmunds@sheridancollege.ca*

### III - ACKNOWLEDGMENTS

---

We would like to thank Nicholas Johnston for his support and guidance during the implementation portion of this project. His involvement as a project and program advisor has been instrumental in the production of this document.

Sheridan College and the Faculty of Applied Science and Technology have made this project possible by providing our team with the infrastructure and encouragement necessary to make this project a success.

# IV - TABLE OF CONTENTS

---

I - Purpose .....	3
II - Authors.....	4
III - Acknowledgments.....	5
IV - Table of Contents.....	6
V - List of Abbreviations .....	8
VI - Legal Concerns & Ethics.....	9
On Disclosure .....	13
Disclosure Timelines .....	14
VII - Introduction .....	15
0 - The Testing Environment .....	17
0.1 - Hardware .....	17
0.2 - Software.....	20
0.3 - Lab Network Setup & Specifications .....	22
1 - Information Gathering .....	25
1.1 - Device Information .....	26
1.2 - Physical Design.....	29
1.3 - Internal Components .....	29
1.4 - Device Ecosystem .....	32
2 - Threat Modelling.....	37
2.1 - Qualitative Threat Modeling.....	38
2.2 Connectivity Threat Modeling .....	43
3 - Capability Discovery.....	49
3.1 - Network Association (Pairing) .....	49
3.2 - Passive Capture.....	51
3.3 - Scanning and Service Discovery .....	54
4 - Testing & Research .....	57
4.1 - Web Application Testing.....	57
4.2 - Mobile Application Testing .....	62
4.3 - Infrastructure & Device.....	65
4.4 - Hardware Testing.....	67

5 - Reporting .....	72
5.1 - Documentation .....	72
5.2 - Formatting .....	74
References .....	77
Appendix 1 - Signal Analysis.....	79
Our Process .....	80
Interpreting Modulation .....	84
Less Standard Modulation .....	87
Example Signal Analysis .....	87
Appendix 2 - Capturing Traffic .....	91

# V - LIST OF ABBREVIATIONS

---

Abbreviation	Meaning
<b>ADB</b>	Android Debugging Bridge
<b>ADB am</b>	Android Debugging Bridge Activity Manager
<b>ADB pm</b>	Android Debugging Bridge Package Manager
<b>AM</b>	Amplitude Modulation
<b>API</b>	Application Programmer Interface
<b>APK</b>	Application Package Kit
<b>ASK</b>	Amplitude Shift-Key
<b>CSRF</b>	Cross Site Request Forgery
<b>DOS</b>	Denial of Service
<b>FCC</b>	Federal Communications Commission
<b>FM</b>	Frequency Modulation
<b>FSK</b>	Frequency Shift-Key
<b>FTP</b>	File Transfer Protocol
<b>GHz</b>	Gigahertz
<b>GR</b>	Gnu Radio
<b>MFSK</b>	Multiple Frequency Shift-Key
<b>MHz</b>	Megahertz
<b>PCB</b>	Printed Circuit Board
<b>PSK</b>	Phase Shift Key
<b>RF</b>	Radio Frequency
<b>SDR</b>	Software Defined Radio
<b>SoC</b>	System on Chip
<b>SSID</b>	Service Set Identifier
<b>UART</b>	universal asynchronous receiver/transmitter
<b>URH</b>	Universal radio hacker
<b>XML</b>	eXtensible Markup Language
<b>XSS</b>	Cross Site Scripting



## VI - LEGAL CONCERNS & ETHICS

---

The authors of this methodology do not condone or encourage the breaking of any laws, by-laws or regulations. For the purposes of this methodology, the readers should be immediately concerned with those laws regarding reverse engineering and unauthorized access or testing of computer systems. Please understand the laws applicable to you within your jurisdiction before starting any testing processes. If you have any concerns about the process, consider seeking counsel or researching the limitations of your rights as a security tester. The Electronic Frontier Foundation (EFF) has a number of resources regarding your rights that we encourage you to review before moving forward.

We have included the information that we believe to be most pertinent below:

Simply put, ensure that you either own the devices you intend to test with, or have explicit written permission to perform such measures. [1]

Some devices may have specific regulations that prohibit testing, disclosure, or reverse engineering. Once again we recommend that you consult available literature or legal counsel to best protect your interests before proceeding.

The following excerpt is from the United States Code of Federal Regulations Title 37 - Patents, Trademarks, and Copyrights §201.40:

*(i) Computer programs, where the circumvention is undertaken on a lawfully acquired device or machine on which the computer program operates solely for the purpose of good-faith security research and does not violate any applicable law, including without limitation the Computer Fraud and Abuse Act of 1986, as amended and codified in title 18, United States Code; and provided, however, that, except as to voting machines, such circumvention is initiated no earlier than 12 months after the effective date of this regulation, and the device or machine is one of the following:*

*(A) A device or machine primarily designed for use by individual consumers (including voting machines);*

*(B) A motorized land vehicle; or*

*(C) A medical device designed for whole or partial implantation in patients or a corresponding personal monitoring system, that is not and will not be used by patients or for patient care.*

*(ii) For purposes of this exemption, “good-faith security research” means accessing a computer program solely for purposes of good-faith testing, investigation and/or correction of a security flaw or vulnerability, where such activity is carried out in a controlled environment designed to avoid any harm to individuals or the public, and where the information derived from the activity is used primarily to promote the security or safety of the class of devices or machines on which the computer program operates, or those who use such devices or machines, and is not used or maintained in a manner that facilitates copyright infringement. [2]*

The excerpt specifies exclusionary factors that would allow reverse engineering some IoT products. However, there are a large number of legal bodies that may limit your testing. We stress the importance of good-faith security testing and the limitations of risk to the public as mentioned in the Code of Regulations. These factors should be considered through all stages of testing. Testing the physical device is fairly straightforward from a legal standpoint. However, the lines may become blurred when assessing the other objects in the IoT ecosystem, mainly the mobile application and web services. Great care should be employed when performing web-based security testing. Ensure that your targets are well researched and you can prove a relationship between the manufacturer of the device and the ownership of target IPs. Be reasonable with your attack vectors. Do not engage in destructive testing and do not engage in acts that may be considered unlawful or unauthorized access to these remote assets. Any testing that would adversely affect normal server and client side operations should be considered strictly out of scope and highly undesirable.

Many devices and mobile applications will utilize third-party cloud services. We do not recommend in-depth security testing of third-party cloud providers. If the presence of third-party cloud providers is discovered, due to the shared nature of those services, we do not recommend testing to be performed against them. Avoid any interaction that may be considered or perceived as malicious or unlawful. Remember, these third-parties are likely unaware of your intentions and may assume the worst. If you have even the slightest level of concern about these interactions, halt your testing and perform your due diligence before proceeding.

Please try to avoid getting sued. We are not lawyers.

## ON DISCLOSURE

---

It is very likely that at some point in the testing process a meaningful security issue will be identified. This section outlines our recommendations for disclosure in such cases. We strongly advocate for responsible disclosure wherever possible. Responsible disclosure refers to ensuring that the vendor has adequate time to address the issues or vulnerabilities before this information is made public. In some cases you may find that the vendor is unreachable, does not care, or is already aware of such issues. There is also the possibility that the vulnerabilities have no method of being fixed or addressed by software updates. The ethics of disclosure in such cases should be considered. The benefit of disclosing to the public that the devices are vulnerable comes with the risk that this knowledge may be exploited by those with more nefarious goals.

Once identified, the findings should be reported as soon as possible upon the conclusion of the testing and verification of the findings. Indicating well in advance that the findings are to be released to the greater public may encourage remediation from the manufacturer. Disclosure timelines can be negotiated if remediation progress is certain and performed in good faith. However, unreasonable delay may necessitate public disclosure without coordination. This should be a last resort. The motivation for disclosing such information publicly should be based on concern for the security of users and nothing else. Non-coordinated disclosure may be a logical measure if no meaningful progress is made towards remediation within a reasonable amount of time. This timeframe will vary based on severity and difficulty of the changes required for remediation. Keep records of all of the important dates mentioned. If disclosure to

the public is the desired course of action, consider redacting or removing Proof of Concept (PoC) code and the details of the vulnerabilities. The choice will ultimately be yours and we hope that you take the time to make a decision with possible repercussions in mind.

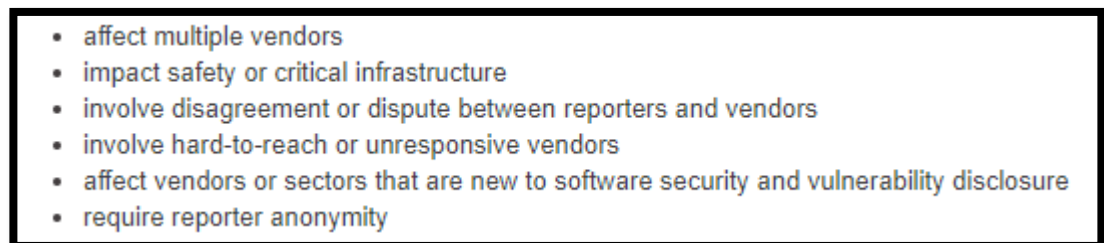
## DISCLOSURE TIMELINES

---

We encourage the use of CERT's vulnerability disclosure policy or that of a similarly respected body. The following recommendations for timelines of disclosure are put forward:

- ❖ Attempt to contact the vendor immediately following the completion of the testing.
- ❖ Public disclosure no less than 45 days following vendor contact.

CERT can also provide invaluable assistance during the disclosure process when some or all of the following criteria may apply. Figure 1 is an excerpt from CERT's Vulnerability Reporting Form as specifies why a tester may choose to disclose through CERT.

- 
- affect multiple vendors
  - impact safety or critical infrastructure
  - involve disagreement or dispute between reporters and vendors
  - involve hard-to-reach or unresponsive vendors
  - affect vendors or sectors that are new to software security and vulnerability disclosure
  - require reporter anonymity

*Figure 1: Excerpt from Certs Vulnerability Reporting Form [3]*

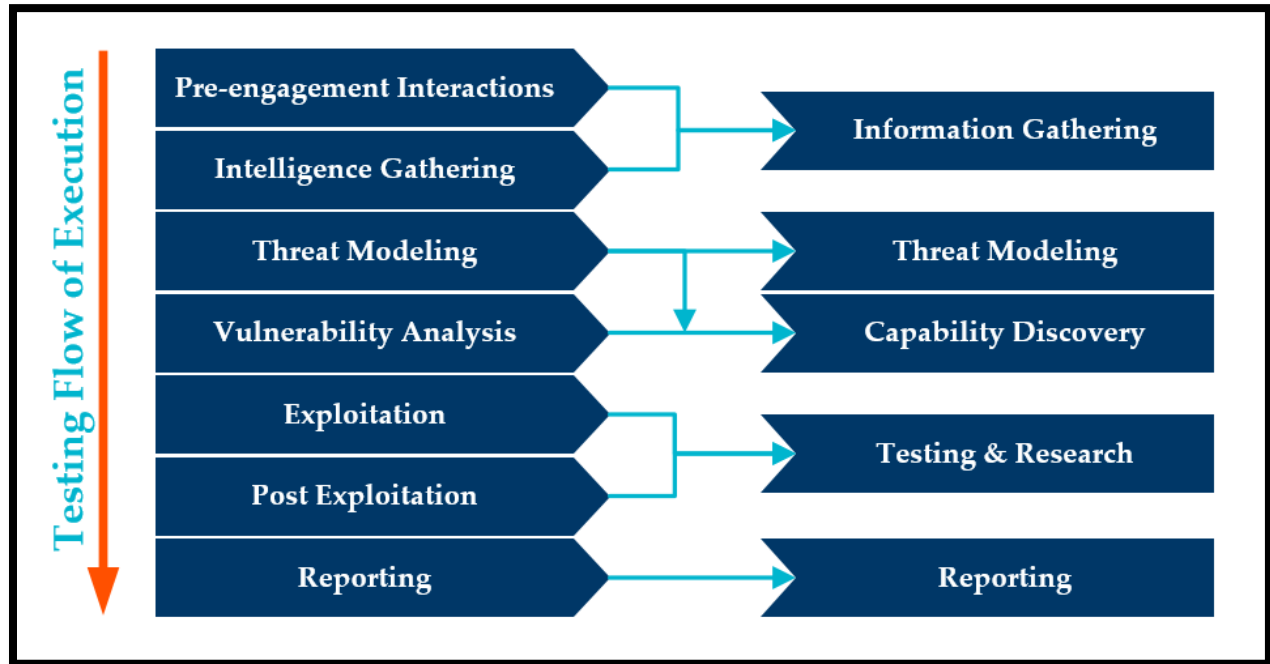
## VII - INTRODUCTION

---

The Internet of Things (IoT) devices referred to in this methodology are consumer devices. To meet our definition of IoT they must utilize network communications, either directly or through a proxy (hub device). IEEE defines the “Internet of Things” as “A network of items—each embedded with sensors—which are connected to the Internet.” [4] Though, based on testing performed in parallel to the creation of this document, the direct connections to the Internet are not always necessary. That being said, some form of networking connection is considered a minimum requirement regardless of protocol.

The goal of this testing methodology is to provide a framework with which to explore the security of these devices. Performing such testing may allow a tester to infer whether the manufacturer built and sold the product with consumer security and privacy as a priority. Therefore, we have established a process to identify attack surface, model potential threats, and attempt to exploit available channels in a manner similar to that of a malicious actor. This process is not entirely dissimilar to the classical high-level phases of penetration testing familiar to many industry professionals [5], but additionally considers core requirements within the context of IoT.

The high-level steps outlined by our methodology are as follows:



*Figure 2: IoT Testing Phases adapted from traditional engagement structure [5]*

This methodology, shown in Figure 2, requires a multidisciplinary set of skills, some of which are specializations within the field of Information Security. We have tried to make this methodology as accessible as possible knowing that many of these points are worthy of their own discussions, rather than subheadings alone. The thorough assessment of any IoT device, would require knowledge of network infrastructure, penetration testing, reverse engineering, cryptography, application security, computer engineering, and experience with an expanding library of special purpose tools. From this understanding, we hope to provide a refined process through which testing of IoT devices can be made more widely approachable to a broader demographic; ultimately helping the effort to secure the ever-growing Internet of Things.



# 0 - THE TESTING ENVIRONMENT

---

The setup of a suitable IoT lab and testing environment is the first logical step in the assessment process. As such, we aim to provide specifications and requirements necessary to perform thorough analysis using our proposed methodology. Below we have outlined the minimum and recommended equipment that may aid, expedite, and enhance the testing capabilities of the environment. Wherever possible this methodology will attempt to remain vendor and application agnostic. The optional requirements will be denoted by an asterisk (\*) in the list below.

## 0.1 - HARDWARE

---

**The Lab** - The testing area should be physically secure. Valuable electronics and equipment will need to be securely stored and safely deployed. Ensure sufficient accommodations are made for devices relevant to your own test cases. Keep in mind requirements may change during testing and accommodations will need to follow suit.

**Power Supplies** - Some IoT devices may not have their own power supplies provided with them. Some devices, like smart thermostats, may need to be wired into mains. Other devices may be designed to be run off of a larger system not provided with the testing device. This is common for automotive 'infotainment systems'. Common power supplies such as 5V, 9V, and 12V may be a benefit to have on hand.

**Basic Electrical kit** - Wire strippers, soldering iron, electrical tape, leads, various connectors, and multimeter may all prove to be useful. These tools are solely for the purposes of hardware testing. While these tools are an added expense, we stress the usefulness of the physical

vector. For example, most of these tools can be used for (re)attaching UART pins to compatible circuit boards; a common hardware-based system entry point.

**IoT Devices** - The proposed test devices related to carefully modeled test cases.

**Wireless Router** - This router will serve as a central node for test devices to connect to, communicate with, and facilitate the broader IoT ecosystem. The wireless network should be restricted for use by the security testers and the ecosystem itself. The network should utilize modern wireless security standards and ciphers where possible.

**Firewall\*** - This may be optional if the wireless router has sufficient on-board capability to filter traffic itself. Employing additional hardware and/or software network filtering will help mitigate unwanted traffic in and out of the testing network.

**Powerbar/UPS\*** - A powerbar or uninterruptible power supply should be used for all devices that require power for testing. These additions provide both safety and convenience in the event of electrical anomalies.

**Mobile Device** - An Android or iOS phone or tablet should be physically available for use in the IoT lab. A large number of IoT devices rely on their respective companion apps to communicate with the device ecosystem. Thus, they are an important part of the testing environment.

Virtualization of these devices is possible and should be considered when it makes sense for individual test-cases. Care should be taken to wipe any personal information from these devices if they are second hand or pre-used.

**Analysis Machine** - A relatively modern laptop/desktop should be adequate for use as an analysis machine. Detailed specifications will be determined by individual test-cases. While we believe that similar results can be accomplished with any operating system, a Linux distribution

is recommended based on availability of highly customizable and interoperable open-source tools. Bluetooth capability is recommended.

**Camera\*** - A camera is very helpful for documenting the devices, serial numbers, packaging, instructions, and internal components during physical disassembly. High resolution is recommended for reading small fonts commonly found on many embedded chips.

**Misc. Disassembly Tools\*** - A smartphone disassembly kit, screwdrivers, pry bars, and anti-static surfaces are recommended. These tools will assist in disassembly and provide marginal protection against damage incurred.

**JTAG\*** - Capability to explore functionality exposed by the JTAG pins is optional and will be determined by individual test-cases. The use and process for it is not detailed by this document.

**UART** - Universal asynchronous receiver/transmitter hardware allows for direct serial communications with the device. Harnessing such a connection may allow for low level access to the embedded filesystem as well as provide an alternative entry point for enabling remote persistence. We recommend having a variety of serial-to-USB connectors like PL2303 adapters or cp210x bridges. [6]

**External Modules\*** - Individual test-cases may require certain additional hardware components to enable an analysis machine to capture and interact with protocols not supported by default. For example Zigbee USB receivers.

**SDR\*** - SDR (Software Defined Radio) can be invaluable if the testing requires any blind signal analysis or testing of non-standard protocols and radio bands. Hardware such as the RTL-SDR or the HackRF can be acquired should the test-cases require it.

## 0.2 - SOFTWARE

---

This section outlines the various software tools that are utilized during the testing process. They are collected and described here for your convenience. The software listed here is largely free & open-source software, however paid or licensed alternatives can be equally or more capable.

**Nmap** - Network scanning tool that can identify hosts, open ports, running services, and perform device, OS, and service version fingerprinting. This tool is most commonly used during the Capability Discovery phase of testing. A graphical wrapper for nmap called Zenmap can be used when applicable as well.

**Dsniff** - Dsniff is a security tool designed to parse a variety of credentials from network captures and live traffic.

**Tcpdump** - Command line network protocol analyzer and capturing tool.

**Wireshark** - Graphical network protocol analyzer and packet capture suite. Wireshark is a wrapper for Tcpdump that includes extended filtering capabilities.

**Tcpreplay** - A tool that replays network traffic from packet capture files.

**Miranda** - A python utility for scanning and interacting with uPNP services.

**Netcat** - Netcat is typically described as the 'Swiss army knife' of networking tools. Netcat facilitates both inbound and outbound TCP and UDP connections for a large variety of network protocols.

**Telnet** - This basic remote administration protocol provides a terminal interface to the device. The service is typically left running by developers as an entry point for debugging and therefore has the potential to be misused.

**SSH** - The secure shell is a cryptographic remote login protocol. It is the preferred secure method of remote access.

**Audacity** - Audacity is useful for analyzing and inspecting audio during signal analysis. The spectrograph and waveform views are invaluable for analysis.

**Gnu Radio** - Gnu Radio is a massive suite of signal processing tools that can be used during signal analysis to decode signals. Be aware that tools like GNU Radio have a steep learning curve. Consider your test-cases before committing your testing time to these tools.

**RTL-SDR** - named after the hardware, provides a set of command line tools for capturing RF transmissions with RTL-SDR hardware.

**URH** - Universal Radio Hacker is a full featured suite of signal processing, capture, and analysis tools.

**Burp/ZAP/Fiddler/POSTMan** - Web proxy tools used to intercept and modify web requests.

These tools will route inbound and outbound web requests through their respective tools and provide a wide variety of testing methods that leverage HTTP request and response tampering.

**AppUse VM** - AppUse Dashboard includes a number of helpful tools for analysis, device interaction, emulation, application reverse engineering, APK decompilation and much more.

The Free version of this environment is limited in advanced offerings but provides a wealth of useful tools for most applications.

**Apktool** - Apktool can be used to decode and sign an APK. This allows a tester to create a patched version of the APK for various means. Apktool can be leveraged to sign an APK for cleaner re-installation on a device or to prove a more complex attack vector (Social Engineering, Phishing, etc.)

**MOBSF** - (Mobile Security Framework) is a static analysis framework that provides a highly digestible HTML report served locally.

**QARK** - Quick Android Review Toolkit is a tool designed and released by LinkedIn - also provides an HTML web report but also will create an 'exploit APK' which will allow a user to control the original app via direct action/intent injection on the device itself

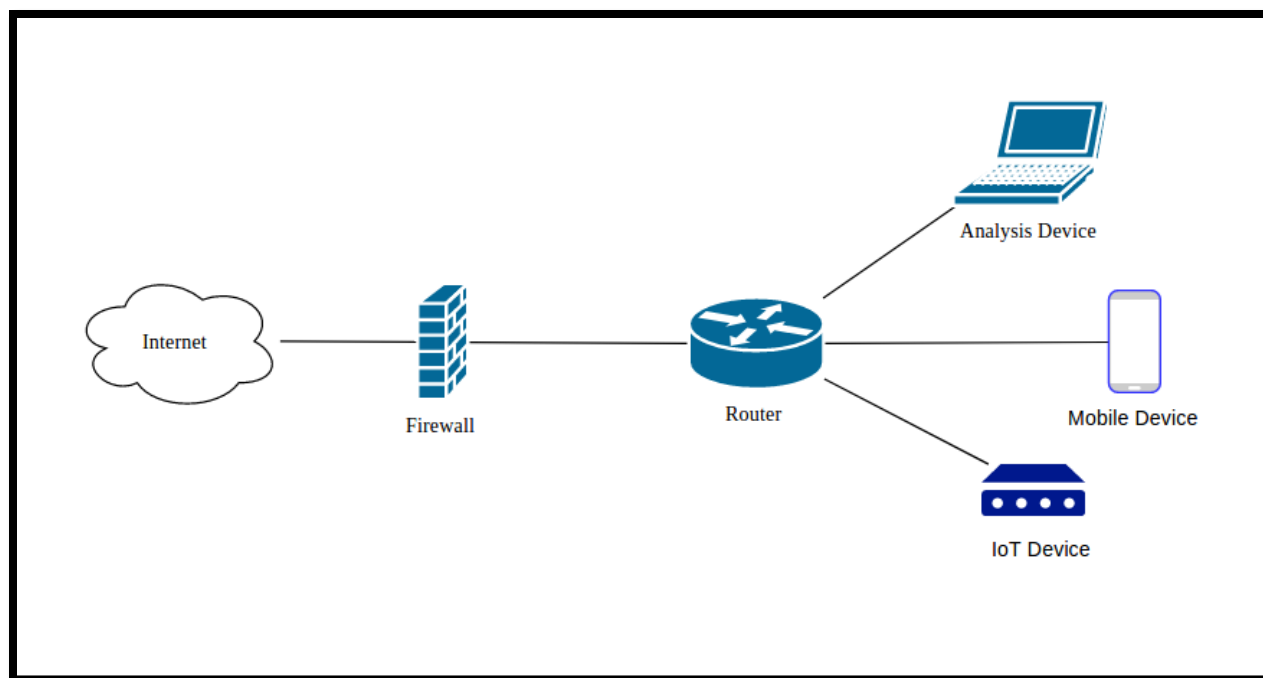
**Screen** - Screen has built-in functionality for facilitating USB-to-serial connection sessions and can be instrumental in UART testing.

**Drozer** - Drozer is a wrapper for ADB activity manager (am) and package manager (pm) and is useful for exploiting functions exported by android applications.

## 0.3 - LAB NETWORK SETUP & SPECIFICATIONS

---

Figure 3 shows the recommended setup of the testing environment (IoT Lab). Starting from the left, the lab should have a connection to the internet. A stand-alone firewall is recommended, however if cost is restrictive a router with built-in firewall rules may be used. Use of a firewall is highly encouraged. Generally, the firewall should be as restrictive as possible while still maintaining sufficient functionality pertinent to testing activities. Non-standard ports and remote terminal services can be enabled as needed. Most consumer routers will have network address translation (NAT) enabled by default. NAT is useful for avoiding the need to have every internal device publicly addressable.



*Figure 3: The basic IoT testing environment.*

Set up a local wireless 802.11 network with a robust security mechanism (WPA2-Personal) and a unique network SSID. Note that in our experience, many IoT devices do not support Wi-Fi Security beyond WPA2-Personal - i.e. a username and a PSK. Ensure that the router and analysis devices are capable of broadcasting on both the 2.4GHz and 5.0GHz networks. Make sure that the analysis device can connect to the wireless network and is configured to listen in promiscuous mode on its wireless interfaces. For more details on how to configure the analysis machine see Appendix 2 on capturing traffic.

It is recommended that a baseline of network activity is captured before any testing starts. The gateway and analysis machine will both generate noise on the network. Understanding this traffic will help when filtering for legitimate and interesting traffic later on. If required by the IoT device, the mobile device for use with companion applications should be added to the

network next. Whether a physical device or a bridged virtual machine is used, baseline traffic should be analyzed in a similar fashion to the analysis device. Connecting the IoT devices should be done only after baseline analysis of related analysis machines is performed and the Information Gathering phase for the IoT device has been completed.

This lab network should be deployed for the sole purpose of testing IoT devices. For the sake of privacy and soundness, it is imperative that access to this network is restricted to the security testers. The network traffic may be captured at any time for the sake of testing and it is crucial that those using the network understand and agree to be recorded. Capturing such traffic without permission may violate laws in certain jurisdictions.



# 1 - INFORMATION GATHERING

---

Before any testing begins, key characteristics of the device and unique identifiers should be thoroughly documented. It is quite likely that if the testing is done in an academic capacity, the information regarding the use of the device may be sparse. A priori knowledge of the device system is limited to that of what a typical consumer is privy to. While documentation in this phase can amass a verbose list of collected information, this section is designed to facilitate the collection of the bare minimum identification data proven to be instrumental in our own testing. Device markings, packaging, instructions, and data sheets can all provide useful information to security testers. It is at this point, that the importance of complete and accurate documentation be conveyed. Any findings should have enough information supplied to be comprehensive, repeatable, understandable, and meaningful to a third party with limited knowledge of the testing procedure.

## 1.1 - DEVICE INFORMATION

---

This information should be the easiest to obtain and comprises of the basic identifying information of the device.

**Manufacturer / Brand** - The manufacturer name or branding is the most obvious piece of identifying information. Such information can provide a baseline for inferences about the manufacturer's expected security posture based on industry expectations.

**Device name** - The device may have a name used to identify it, rather than just a description of what it does. This can help distinguish the specific test device from similar or, in some cases, identical devices branded by other manufacturers.

**Model Number** - This information is typically harder to identify but doing so will help link a specific product and/or revision to entries in technical specification documents.

**Photos** - It is highly recommended that photos of the device exterior and packaging be taken. These are useful for reporting on the device later on and is indicative of good testing and documentation habits.

**Cloud Services** - This information pertains to whether the device claims to use cloud services. Note that manufacturer claims and device functions do not always align.

**Device type / Subtype** - This helps classify the device being tested. It may also help researchers find similar products and implementations. Typically devices of the same family will have similarities in attack surface that can assist in performing a thorough assessment. This classification system will likely need to be expanded as new devices come to market. At the time of writing, our device type and subtype nomenclature has been grouped hierarchically and is shown in Figure 4.

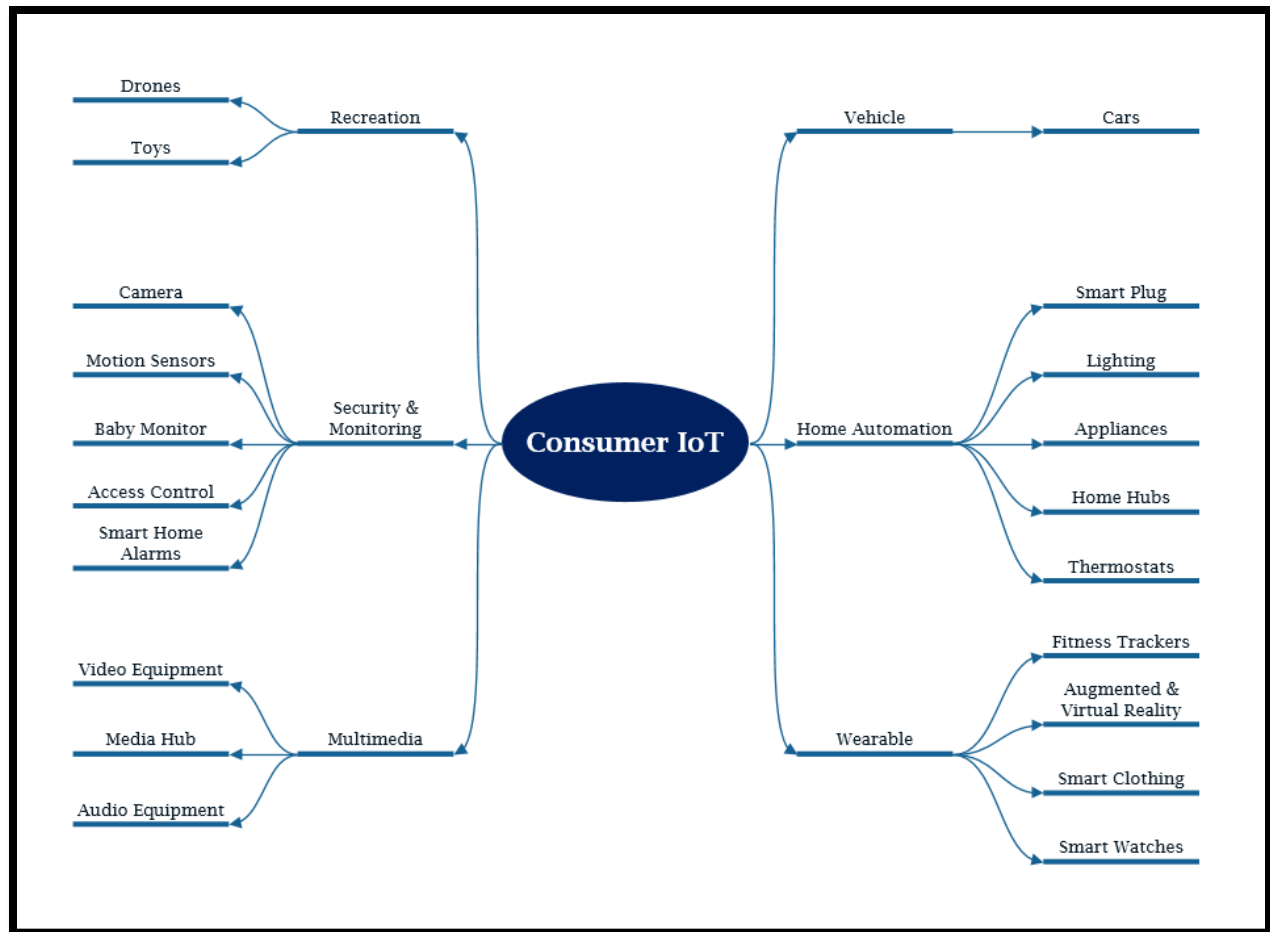


Figure 4: Consumer IoT heriarchical type chart

Device type and subtype can then be specified in a structured way. For example, an IP camera can be specified as Security & Monitoring/Camera. We believe a system of categorization will be beneficial in the long term as retroactive categorization is inefficient.

**Companion app** - If the device uses one or more companion apps to operate or augment the behavior of the device, record the names, platforms, and fully qualified names if possible. These applications will likely leverage web requests to special APIs on a remote web server.

**FCC ID** - If the device has a FCC ID present, record that information. Verify that the identification number is real by using one of the many online services. [7] If the device has an actual FCC entry, invaluable system specifications will be available for public access. This may include high-

and low-level operational information, internal photographs, data sheets, and even protocol descriptions.

**Serial Number** - The serial number is distinct from the model number and represents an individual device rather than the subset of devices. This can be used as a private key in some implementations.

**MAC Address** - In some cases, this information may be printed on the exterior of the device. If not found on the device itself, the MAC address can be obtained through network discovery activities once the device is powered on and/or connected to the network.

**Brief Operational Description** - Provide a small description of the intended function of the device. This can be useful later on when threat modeling by utilizing use case analysis.

**Default Credentials** - Instructions, packaging, or labelling may contain information about default credentials of the device. Record it where possible for use in post-exploitation phases.

## 1.2 - PHYSICAL DESIGN

---

**Exposed Ports** - Exposed ports may be intentionally left by the manufacturer for variety of reasons. Access to such ports may allow attackers to access internal systems, removable memory/storage or assume a more trusted connection to the device. USB, Micro USB, SD card, and Serial access ports are valuable external ports for attackers.

**Sensors** - Take note of any sensors that the device may possess. These impact the intended and unintended functions of the device. Hidden/undocumented functionality may be discovered during this step. Each physical sensor whether microphone, camera, GPS or otherwise each may pose a distinct privacy risk to the eventual consumer. The more complex the sensor data or equipment, the more complex the underlying system or systems will need to be. This may increase attack surface and consequently increase analysis time.

## 1.3 - INTERNAL COMPONENTS

---

Disassembly of the device may provide valuable information for testing if specifications about related behavior is sparse or nonexistent. Internal components can be identified fairly easily with practice and research. If the entire assembly has not been documented, useful information can still be obtained from the hardware of which it is comprised. Below is an example of a device that was disassembled during our testing. Details of what can be learned are listed below.

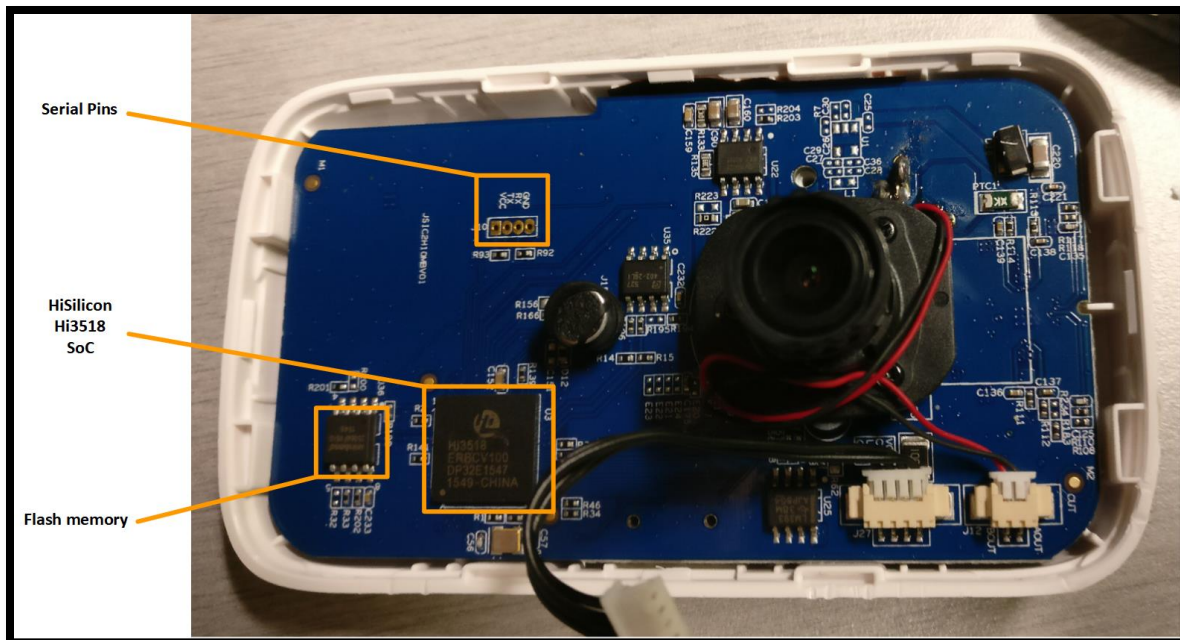


Figure 5: The front of a disassembled IP Camera.

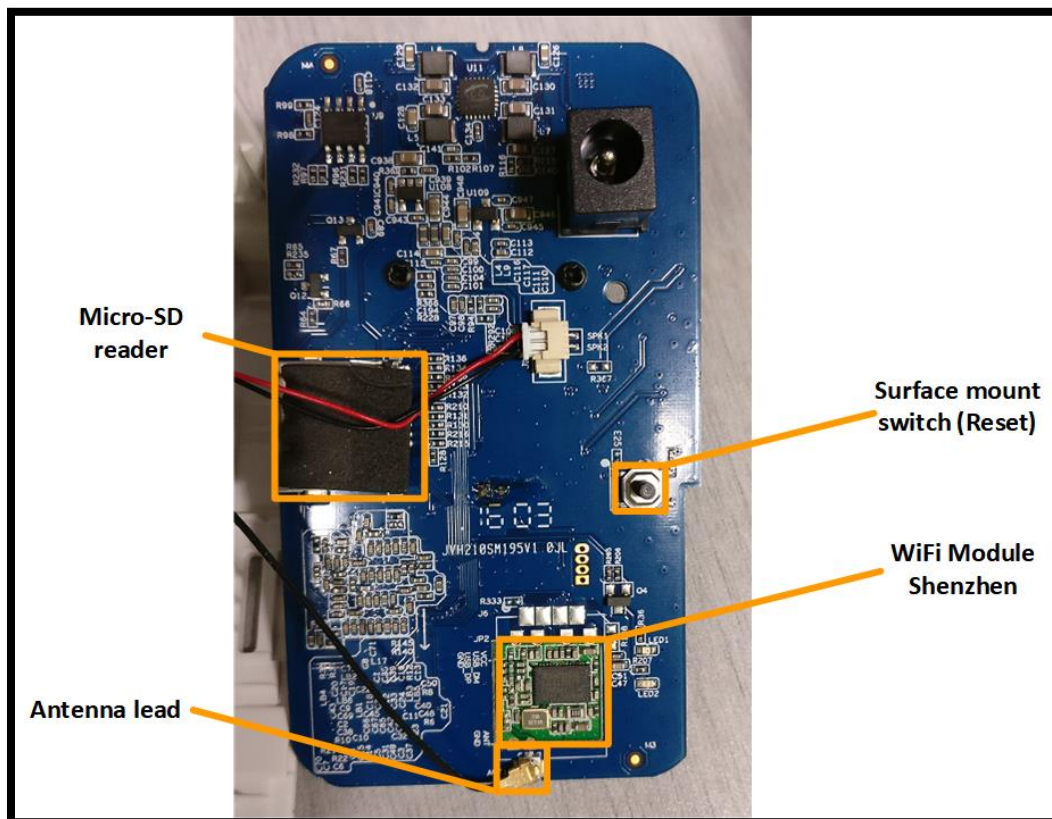


Figure 6: The reverse side of the IP Camera.

Pictured in Figure 5 are the serial communication ports labelled as RX & TX. These pins can be connected to for direct communication to the device. [6] The bottom-left component is the flash memory, from researching the chip we could guess that the camera could operate with a reasonably complex operating system. The bottom-right component is the System on Chip (SoC) manufactured by HiSilicon. The SoC identified has a detailed datasheet of over 1200 pages. [8] However the takeaways are that the chip supports multimedia encoding up to 720p and includes strong cryptographic capabilities including AES hardware support. This camera has the hardware capabilities to implement strong security and support reasonably complex operations. Out of frame, mounted to the surface of the interior housing are the microphone and speaker array. Figure 6 shows the reverse side of the PCB. From this view the externally accessed MicroSD reader, the add-on Wi-Fi module, and antenna lead are visible. Details on the Wi-Fi module would indicate which 802.11 standard is used, and in which band it operates. From the disassembly, we learn that the camera runs on an ARM architecture, and has the capability to run a complex operating system. Serial ports were present on the device, however they are not exposed and require modification to be leveraged. Further, the devices operational capabilities are identified before it was powered on. The device has a microphone, a Wi-Fi module, and, of course, a camera that is made accessible to the device.

Identifying wireless bands and sensors provides valuable information on the means by which out-of-band communication is used during the pairing or network association process. This process is often not documented but can introduce security issues. An overview for this process is described in 3.1 - Network Association (Pairing).

## 1.4 - DEVICE ECOSYSTEM

---

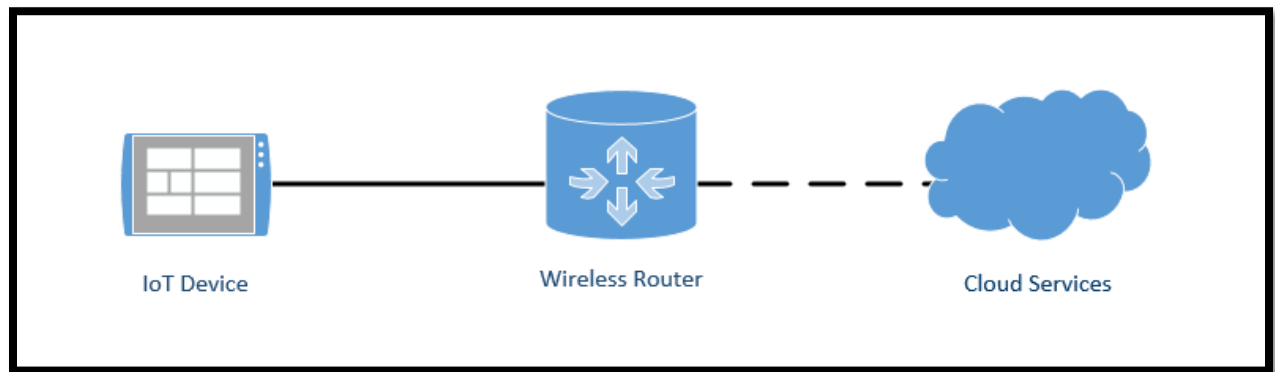
Before transitioning to the threat modeling stage, it is imperative to begin conceptually mapping the device and its services. Having a basic understanding of the device ecosystem will greatly assist in mapping attack surfaces. Devices may utilize a number of external entities to produce their own intended functions. At a minimum, an IoT ecosystem will rely on at least one additional node. The IoT device itself will require an access point through which it can connect to the greater Internet. In most cases this will be the consumer's wireless router. However, for smaller distributed devices, “hubs”, “bridges”, and/or “gateways” are used to communicate with numerous peripherals. For the purpose of clarity, we will refer to all subsequent instances of such devices and “bridges”. Setting up a simple diagram of the expected communication pathways is helpful for correctly mapping the device ecosystem. The following subsection suggests typical models that IoT devices may utilize. The naming conventions proposed within uses terminology that we have created that has been successful in conveying the structure of an IoT device and its related ecosystem.

### 1.4.1 - CONNECTIVITY MODELS

Figure 7 shows the Trivial Cloud Model for an IoT device ecosystem. These models have been abstracted to be only conceptual maps and do not necessarily reflect the complexity of a real-world system. An arbitrary device has been associated with a network, and cloud services are made accessible via remote access through the wireless router. For the purpose of testing the security of consumer devices, this represents the minimal attack surface for the tester as the

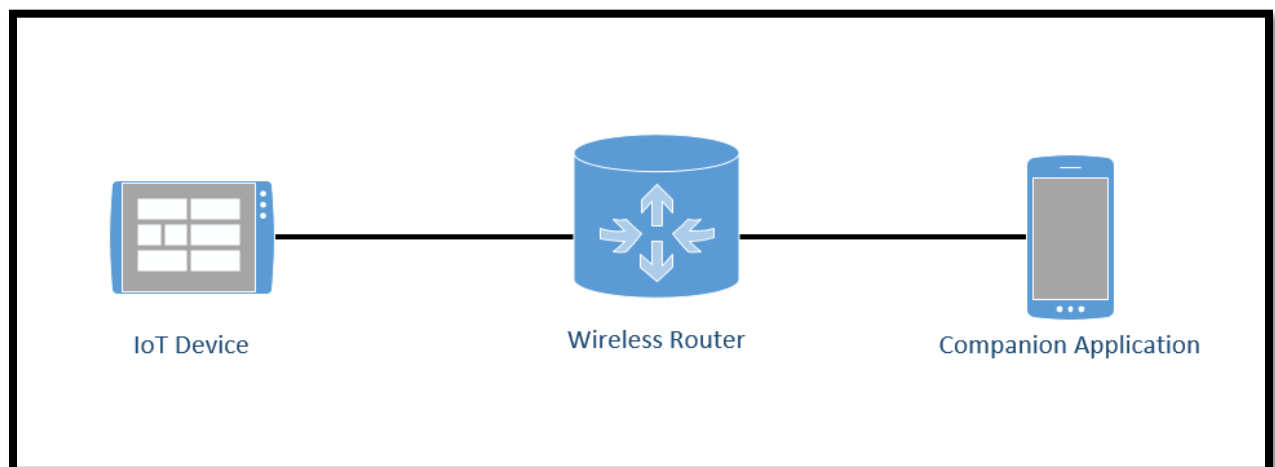


cloud services operate as a black box. Such architecture necessitates that data can and will be stored outside of the consumer's control.



*Figure 7: Trivial Cloud Model of connectivity*

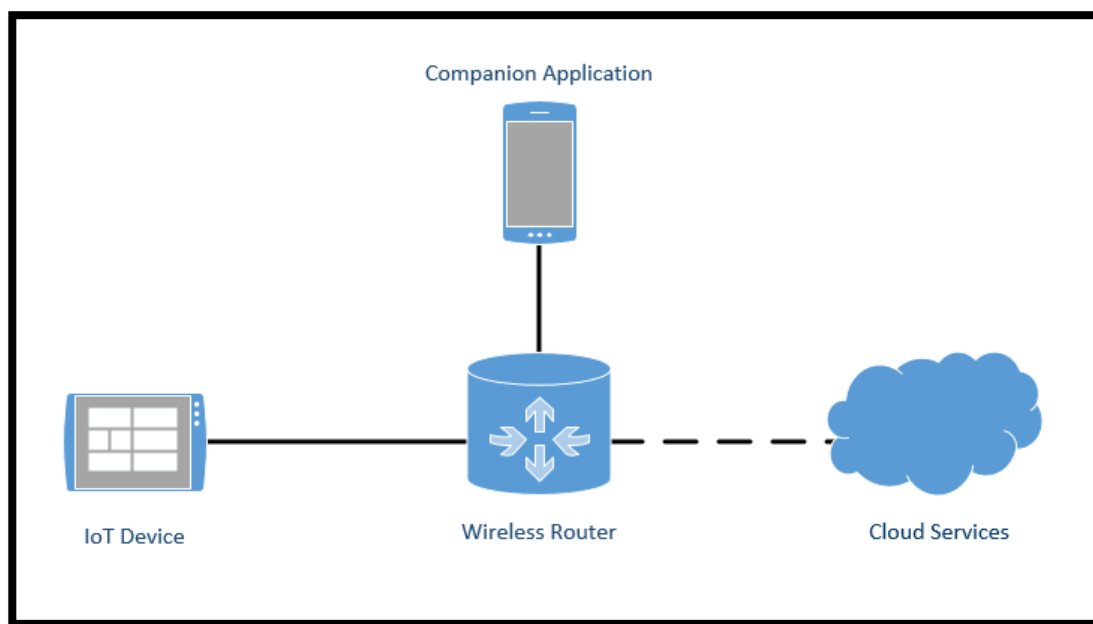
The Local Companion Model will incorporate a companion application that is intended to operate entirely on the local network. The cloud services are replaced by access through an application that interfaces with the IoT device locally. This connectivity model may represent an attack surface and privacy risk that stems from software directly installed on the consumer's hardware. The abstracted diagram is shown below in Figure 8.



*Figure 8: Local Companion Model of Connectivity*

Logically following, is the hybrid of the first two models - The Cloud Companion Model.

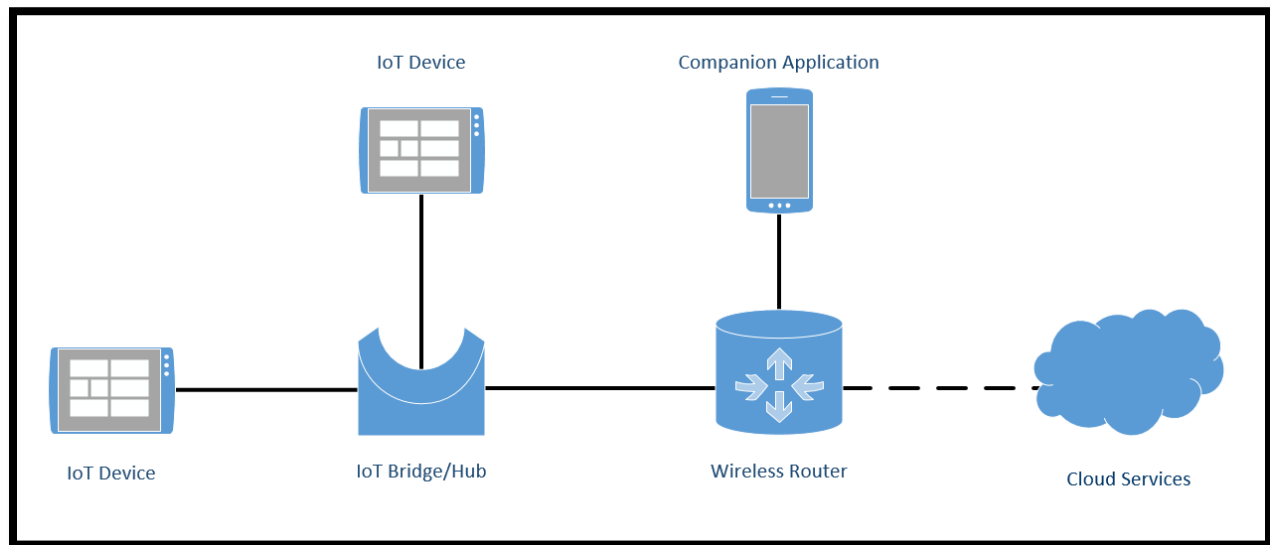
Companion applications are utilized alongside cloud services. The attack surface available is now the accumulation of their disjunct base models. The companion software must run on consumer hardware and the cloud services process data outside of consumer control. The model is shown in Figure 9.



*Figure 9: Cloud Companion Model of Connectivity*

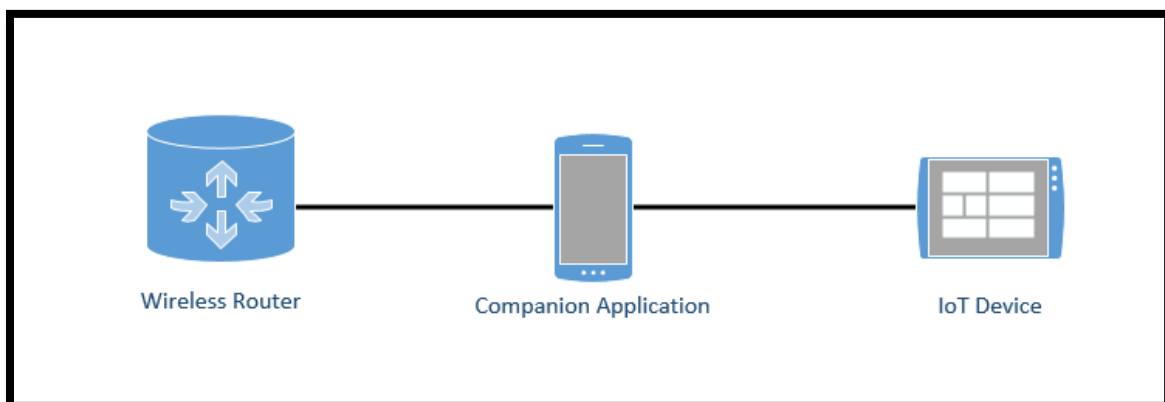
IoT systems that utilize “bridges” rely on separate networking devices intended to communicate with the individual IoT devices through separate networks, bands, or protocols. The addition of an IoT “bridge” to the model can greatly increase the complexity for the security tester. “Bridges” may operate on vastly different hardware and operating systems than the peripheral devices. Figure 10 shows a typical bridged IoT model combined with the Cloud Companion model - The Bridged Cloud Companion Model. This is not unlike the model describing by the Philips Hue smart bulbs. [9] The IoT peripheral devices of the same type can

be removed from the model if communications are assumed to be independent and bear no additional significance to the complexity of the model.



*Figure 10: Bridged Cloud Companion Model of Connectivity*

The next model of significance is shown in Figure 11 and is referred to as the Personal Area Network (PAN) Companion Model, is that which may describe wearable IoT device ecosystems. The IoT devices are paired via Bluetooth to mobile devices and communicate through a related companion application.



*Figure 11: PAN Model of Connectivity*

The ecosystem should be conceptually modeled to, at a minimum, the depth proposed above before proceeding to the threat modeling stage. There are a large number of variations and modifications to consider in these proposed ecosystems and as such, the connectivity models should reflect this diversity. Thankfully this modeling process is trivial as the transitivity of information is the only factor to consider.

The connectivity model naming scheme utilizes the following terms that can be combined and ordered as the ecosystem requires to expressly describe the system. The Trivial prefix is the only conflicting descriptor that should not be used with other descriptors.

- ❖ Trivial
- ❖ Local Companion
- ❖ Cloud Companion
- ❖ Bridged
- ❖ PAN
- ❖ Mesh (Not described here)

## 2 - THREAT MODELLING

---

This section will focus on threat modeling and drives a structured approach to how the remainder of the testing will proceed. This will, of course, narrow down the testing attack surface to a meaningful yet practical size for the scope of work. Keep in mind the relevant connectivity models of the IoT device ecosystem. The thought process should reflect vectors of attack for all elements, rather than just the physical device. In order to begin building a threat model, there are a number of questions that should be answered. These questions are designed to help the device (problem) owner build awareness about the devices and services being tested and evaluate the current security posture of the infrastructure. In order to provide a comprehensive perspective on threat categorization, we will be leveraging Microsoft's STRIDE Threat Modelling tool for use case development and enumeration. [10] In theory, many of the threat modeling techniques are used during development to pursue the creation of secure systems. Following this framework can quickly help designers identify unmitigated vectors of attack. For the purposes of this testing methodology, we will leverage this framework post design for a largely "unknown" system of devices. Further reading about the about STRIDE can be found [here](#).

The following subsections will demonstrate two complimentary threat modelling techniques. Qualitative Threat Modelling will leverage a questionnaire-style technique to help build awareness while Connectivity Threat Modelling will leverage relevant Connectivity Models to enumerate threat categories. Both techniques are based on STRIDE and can be used together or separately to achieve the desired results.

## 2.1 - QUALITATIVE THREAT MODELING

---

The following question sets are designed to spur a mode of cognitive awareness that will aid in developing a meaningful set of use cases for security testing. Note that not every question will apply to every testing engagement but this list is meant to provide an approachable entry point for execution of this phase.

### SAMPLE QUESTIONS

#### **Who is using the device?**

Are you a consumer or a business owner?

Who is and how many people are managing the device?

#### **How many assets are required?**

If more than one, do the devices need to communicate?

#### **Where will the device be located?**

Is the device part of the internal infrastructure?

Is the device segregated to a specially managed network segment?

Is the device located on the interior or exterior of your building?

#### **What is the purpose/goal of incorporating this device?**

Is the intention to increase productivity?

Perform Monitoring and Detection capabilities?

Provide Convenience and Comfort?

Perform critical business functions?

#### **What is the device designed to do?**

Do these functions align with the intentions?

**How will it be used to fulfil the purpose/achieve the intended goal?**

Is the device performing tasks locally or remotely?

Does the device require 3rd party tools to do so?

**What are the expected methods of access?**

Who has access?

Will access change over time?

**When is it expected to be running?**

Is the device expected to be active 24/7?

Does the device operate on a fixed schedule or a dynamic schedule?

How does this schedule affect the expected methods of access?

Finding answers to questions like these will allow security testers to more accurately and efficiently streamline the security testing process. Use cases can be built that will be used to provide meaningful testing scenarios that will best emulate real threats vectors. Such method provides testers with threats that are grounded to realistic goals for an attacker. These threats can range from privacy concerns to security vulnerabilities that allow for remote code execution. The following threat modelling scenario will aim to produce answers to a core subset of the above questions given a sample use-case for a consumer IoT device.

## **SAMPLE SCENARIO**

Steve is an average consumer-level user. He has two dogs but works full time with long hours and would like a way to keep an eye on them while away from his home. He also like to shop online but his office does not allow employees to ship packages to the office. Steve is worried about someone stealing his mail if he allows couriers to leave packages on the porch. Steve decides to purchase a pair of cloud-based IP Cameras to monitor his living room and front porch from his phone. Based on the scenario above, the following answers can be found:

### **Who is using it?**

Steve is a consumer. He will likely be managing the device himself and he needs 2 cameras in total.

### **Where will the device be located?**

The devices will be located both inside and outside of his home. One camera will be pointed at his front entrance providing a view of his front door which happens to have a numeric keypad locking system. The other camera will be pointed at his living room where the dogs spend most of their time. It is important to note that this view will include the rear sliding glass door and doggy door that leads to the backyard.

### **What is the purpose/goal of incorporating this device?**

The purpose of this device is to monitor the wellbeing of Steve's pets and inform him when a package has been left at his door and monitor its position.

### **How will it be used to fulfil the purpose/achieve the goal?**

The cameras are Wi-Fi enabled and are linked to a cloud service. This cloud service is accessible through a mobile application and/or web application that requires user authentication to view



the video streams. The camera records video and sounds and stored them locally to an SD card. The cloud service does, however, allow video snippets to be stored on the cloud for playback on Steve's devices via the mobile and web applications.

### **When is it expected to be running?**

Both cameras are always on and recording.

This information can now be used to determine some realistic threat vectors against Steve and his home surveillance set-up. Three main entry-points can be identified.

#### **1. Mobile Application**

- ❖ If Steve's phone is stolen, can the information stored locally by the application be accessed? Can authentication be bypassed?
- ❖ What communications between the device and the camera can be captured? Do these communications leak any data about the user or the network?

#### **2. Web Application**

- ❖ How secure is the login portal? Can sensitive data be accessed without authentication? What information about the webserver can be discovered?

#### **3. Home Network**

- ❖ Does the camera broadcast locally? Does that require authentication or are local users assumed to be given access?
- ❖ What information can be gathered/spoofed in regards to various sensors (Wireless, Bluetooth, IR, etc.)?

STRIDE Category	Realistic Threat Vectors
<b>Spoofing</b>	The web and mobile applications are prime targets for spoofing user credentials. Determining flaws in either may lead to access via default credentials or weak security configuration.
<b>Tampering</b>	Can the cameras be misconfigured remotely without user authentication? Cross-site request forgery is worth testing for. Remote configuration via SOAP may also be possible.
<b>Repudiation</b>	Check for logging capabilities.
<b>Information Disclosure</b>	Discover if the web server leaks information about its users or itself. Unauthorized access to the camera feeds may reveal to an attacker the access code to Steve's front door as well as information about access via rear sliding door.
<b>Denial of Service</b>	Can the cameras be disabled remotely? This would disrupt Steve's ability to monitor his home.
<b>Elevation of Privilege</b>	If breached, the camera may act as a pivot point for entry into Steve's home network.

Now we have a solid direction for our overall testing plan. We will:

1. Attempt to gain access to the mobile and web application accounts to expose an ability to exfiltrate stored recordings.
2. Attempt to gain access to the video stream remotely and expose the ability to collect information about physical security attributes and/or other private information about Steve.

3. Attempt to compromise the cameras in order to expose the ability to pivot into Steve's home network.
4. Attempt to discover alternative ways to tamper with the cameras to expose physical or short- and long-range communication weaknesses for denial of service.

## 2.2 CONNECTIVITY THREAT MODELING

---

This threat modeling technique utilizes the abstracted connectivity models first outlined in the Information Gathering phase (1.4 - Device Ecosystem). Successful execution will result in a simplified representation of the connectivity inherent to the ecosystem at hand. The Connectivity Model shows devices and paths of information transitivity. These elements are represented by nodes and edges respectively. The nodes should be considered objects that store, process, and transmit data. The edges represent connectivity and should be considered as information pathways with data that can be captured, altered, blocked, reordered or otherwise interfered with.

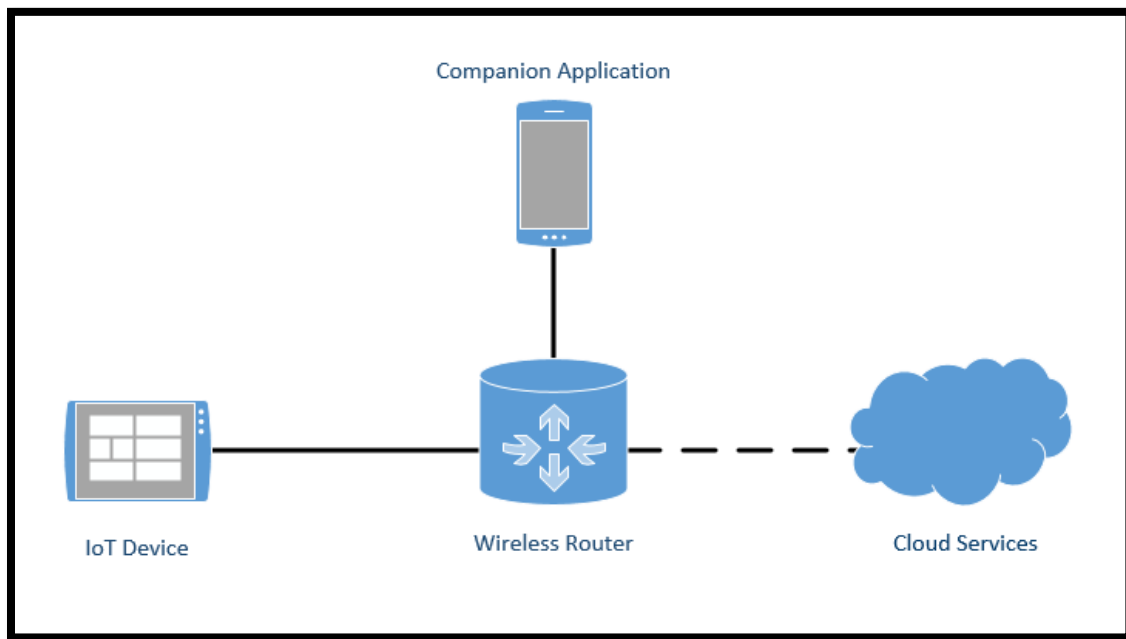
Microsoft's IoT Security Architecture framework provides a sound technical approach for how to proceed with enumerating threats during development. [11] The Security Architecture framework breaks the Threat Modeling process into four steps. This framework has been adopted and modified for the purpose of IoT security testing with minimal knowledge of the system. The original approach has been altered to better suit the challenge at hand. This model will now only use three separate steps, the first of which has already been completed if the Information Gathering stage has been successfully completed.

From a high level, the steps are as follows:

1. Model the application
2. Identify core elements
3. Enumerate STRIDE threat categories

## 1. MODEL THE APPLICATION

Consider the IP camera from the sample scenario in the previous section. The simplified connectivity model is shown again in Figure 12. The IoT device, local network, companion application, and the cloud services from the use case are all considered. This connectivity model will be the foundation for this method of threat modelling.



*Figure 12: IP Camera Connectivity Model (Cloud Companion Model)*

## 2. IDENTIFY CORE ELEMENTS

As simple as this process now sounds, we encourage the deliberate and thorough use of formal steps for threat modeling. The identification process utilizes the four distinct core elements from Microsoft's Stride model [10]. These elements, listed below, have been modified in definition to fit our testing methodology:

1. Processes: Data creation, modification, utilization. This encompasses code in execution.
2. Data Stores: Data at rest. Local storage, databases, RAM, logs.
3. Data Flow: Data in motion. Connectivity and networking. Transit on local device, such as system bus communications are included here.
4. External Entities: Services determined to be out of scope for testing.

With a slight alteration to the Connectivity Model, we have a rough estimate of what each node will likely do based on the four elements defined above. For quick modeling and unambiguous representation basic symbols can be used to represent the core elements. These should be fairly intuitive at a glance. Gears indicate processes, files represent data stores, arrows indicate data flow, and the prohibition symbol represents cloud services that you likely do not have permission to test. In this use-case, it is extremely likely that both the camera and the companion application process and store data. The updated diagram is shown in Figure 13.

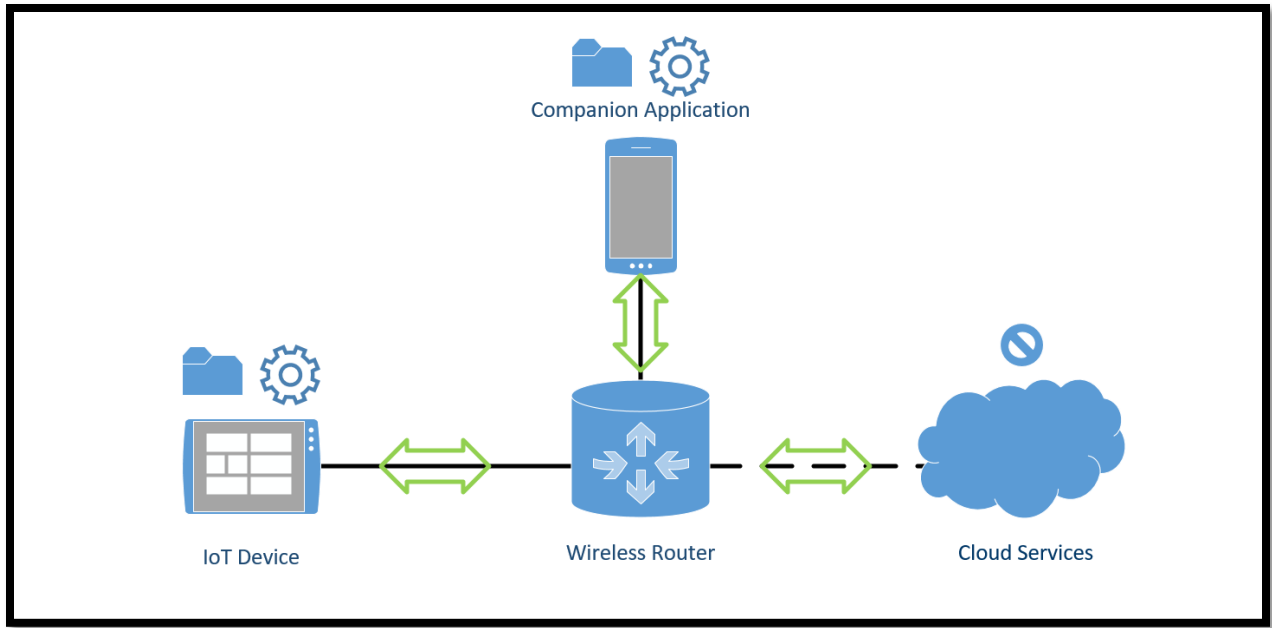


Figure 13: Connectivity Model with core elements indicated.

Once a basic identification of elements is complete, the list can be written, as follows, in a tabular format. *Table 1* reflects the core elements of each node in an organized way.

Connectivity Model Segment	Core Element
IoT Device ( IP Camera )	Processes
IoT Device ( IP Camera )	Data Stores
IoT Device ( IP Camera )	Data Flow
Companion Application	Processes
Companion Application	Data Stores
Companion Application	Data Flow
Cloud Services	Data Flow

Table 1: Tabular representation of core elements

### 3. ENUMERATE STRIDE THREAT CATEGORIES

Finally, we incorporate the STRIDE recommendations, as outlined by Microsoft, for each of the core elements. In summary, the following Threat Categories apply [10]:

**Processes:** Are to be tested for STRIDE threats.

**Data flows & Data stores:** Are to be tested for TID threats.

**External entities:** These are unlikely to be within the scope of the test, however communication contained to the testing network are fair game. These entities should be tested for SRD.

These threats can be added to the earlier table to provide a more comprehensive listing of threats to the device and its ecosystem. The completed table is shown in *Table 2*. These threats abstractly represent the categorical attacks that the testers can attempt to exploit in the testing process.

Connectivity Model Segment	Core Element	STRIDE Categories
IoT Device ( IP Camera )	Processes	STRIDE
IoT Device ( IP Camera )	Data Stores	TID
IoT Device ( IP Camera )	Data Flow	TID
Companion Application	Processes	STRIDE
Companion Application	Data Stores	TID
Companion Application	Data Flow	TID
Cloud Services	Data Flow	TID
Cloud Services	External Entities	SRD

*Table 2: Expanded tabular representation of threats.*

The STRIDE threat categories represent potential threats to each of the four core elements (Processes, Data Flows, Data Stores, and External Entities) that have been attributed to the

nodes in the Connectivity Model. This listing represents the full abstract attack surface of the Connectivity Model. As new processes, data flows, and data stores are encountered during testing activities, refer back to these categorical findings to ensure that a complete assessment of the attack surface is performed. Moving forward as specific instances of the core elements are identified, these threat categories will guide the security testing.

Using the techniques above, we can apply our proposed use cases to our next phases of testing. In the Capability Discovery stage, active discovery techniques will be performed to determine what the test devices offer in terms of functionality and communication.



## 3 - CAPABILITY DISCOVERY

---

Following the successful development of practical, threat-emulating use-cases through threat modelling techniques, it is important to begin performing active collection of device information to discover both hidden and advertised device capabilities. In so doing, the list of use cases should grow and shrink based on the capabilities, or lack thereof, discovered. It is important to note that, while the Threat Modelling stage is designed to produce a streamlined plan for testing, the plan should be treated as a dynamic strategy that requires thoughtful revision as testing continues.

### 3.1 - NETWORK ASSOCIATION (PAIRING)

---

The first stage of Capability Discovery includes capturing network association traffic across the IoT device ecosystem. The inherent challenges of passing the network SSID and secret key to a device with no a priori information about the network is often an issue that is poorly addressed. The process is made more difficult with devices that have no display or interface with which to input such information. As such, many IoT manufacturers have devised clever and cost effective ways of associating to a network. This process, if not designed securely, can disclose network and personal credentials to attackers.

From the details collected in Information Gathering phase, there should be a reasonable idea of how the device associates to a network. In most cases, the mobile device will use some method of out-of-band communication to associate the device to a network. Whichever method is used, we must validate three key requirements:

1. The network authentication information is transferred using strong encryption.
2. Global static keys were not used as ad-hoc passwords.
3. Keys used for authentication are following a minimum standard for complexity.

Failure to adhere to the above requirements would indicate that the IoT device does not employ strong security methods for network association. The following categories have been identified through our research as common network association methods for IoT devices.

## **AD-HOC NETWORKS**

The transfer of network information may be facilitated through an ad hoc network. This may be done through a standard 802.11 networks or through other common protocols like ZigBee.

Credentials can be passed through a proprietary client or a local web server exposed over the ad-hoc network. Alternatively, a Bluetooth pairing process could be initiated between network nodes.

## **AUDIO ASSOCIATION**

Mobile applications may utilize encoded audio streams in both audible and non-audible ranges to transfer the network information to the microphone-equipped IoT devices. A microphone can be used to capture this information. Proprietary and open-source libraries may be leveraged for this method. Services like Chirp.io are making this process more accessible to IoT manufactures. [12] Signal analysis may be performed to determine if the audio information was transferred as plaintext. Information about decoding such methods of transmission is shown in Appendix 1.

## QR CODE

Some mobile devices create QR codes from supplied network credentials. These may be held up to the camera or video sensors to transfer the network information. QR readers can easily decode these messages. As with other methods, care should be taken to verify that the message is not plaintext or static.

## PHYSICAL TRANSFER

The method of physical transfer utilizes a physical medium to transfer the information required to pair to the network. This may include connecting a device to a general purpose computer through an Ethernet cable. Alternatively, USB variants can be used. Physical transfer can also be achieved by writing configurations to removable media such as MicroSD cards that are then physically connected to the device. Physical transfer can be done in a reasonably secure manner, however, this is often at the cost of leaving exposed ports on the device. It is fairly simple to monitor or examine traffic on physical interfaces. Wireshark, for instance can capture traffic along USB and Ethernet interfaces.

## 3.2 - PASSIVE CAPTURE

---

Useful information may be obtained by observing traffic within the IoT ecosystem before the active testing is started. This section will outline key information to look for by utilizing passive capture techniques.

## **TRAFFIC ANALYSIS**

Once the device is successfully associated to a network, traffic may be properly observed and examined. Detailed recommendations on capturing network traffic can be found in Appendix 2. During the initial stages of the testing, it is recommended that access to the Internet is disabled at the gateway in order to better understand remote service dependencies.

In contrast to the baseline network activity gleaned from the lab setup stages, the traffic related to the newly associated IoT device should be apparent. If the MAC (Physical) address of the IoT device was not identified from the Information Gathering stage, it should be easily acquired and recorded at this point. This is useful for filtering communications during later stages of testing. The Organizationally Unique Identifier (OUI) can also be used at this point to determine the make of the WI-FI module if not previously discovered during physical inspection. Identification and inspection of the traffic should indicate which active services are available and used. While the ports and services that can be used are numerous, there are a few of specific importance at this phase. Common services of importance are listed below.

### **DNS**

Inspection will quickly indicate if the device attempts to resolve domain names upon access to the network. Wireshark captures can be quickly and easily filtered to find traffic directed to the gateway on port 53. This can indicate connectivity to cloud services, remote authentication, device registration, beaconing, or other communication initialization activities.

## **uPNP**

Miranda is a security tool that can sniff or actively scan for uPNP traffic as well as submit arbitrary uPNP requests based on parsed schema. Miranda can quickly identify uPNP services and report on their exposed functionality. uPNP services provide attack surface that is often vulnerable to overflows and command injection attacks.

## **MDNS**

MDNS can actively broadcast useful information in plaintext, as it is intended for a local broadcast audience. Hostnames, device names, and various identifiers can be retrieved. Device names can also disclose chipsets through naming conventions.

## **Plaintext Traffic**

Many devices will use a variety of plaintext communications for service discovery, peer-to-peer communications, announcements or local network activities. A brief examination of passive traffic should be conducted. As with most of the web vulnerabilities, it may be possible to perform Man-in-the-Middle (MITM) attacks, replay attacks, or spoof these communications. If the protocol is non-standard, it is likely a good candidate for deeper testing. After a reasonable amount of traffic has been captured, DSNIFF can be run against the packet capture. Dsniff may quickly parse through packet captures for credentials belonging to a variety of service communications. This is an effective way to catch “low-hanging fruit” early on in the testing process.

### 3.3 - SCANNING AND SERVICE DISCOVERY

---

This section directly focuses on how to effectively scan IoT devices for exposed services. While a lot of this methodology will be very familiar to those who have experience performing penetration tests, there are a few key differences that should be acknowledged within the context of testing IoT devices. Actively scanning or mapping a device on an internal lab network can be different from performing the same activities at a large scale during an enterprise-level test. The testing lab environment should be entirely owned or operated by the testers. This alleviates most, if not all, concerns about unauthorized access or scope boundary issues associated with typical penetration testing. Additionally, rather than testing within a large range of IP address blocks, the IoT ecosystem should be comprised of a small number of fixed hosts. Scanning these devices is only limited by the relative computing power of embedded hardware. It is highly likely that the analysis machines have the potential computing power to effectively perform Denial of Service (DoS) attacks by scanning less capable IoT hardware. Scans should be rate-limited if high accuracy of results is desired, as small System on Chip (SoC) processors can be easily overloaded. Unlike traditional penetration testing activities, the assessment of IoT devices can be far less adversarial. There is typically no active party attempting to defend in real-time against the testing or to identify the intentions of your operations. The usual tradecraft of covert scanning has no real advantage in the confines of the testing network. There is no real need to be clandestine when testing in a privately owned and controlled environment. Be loud, be thorough, and get results.

## NMAP

Nmap is the go-to choice for most service discovery activities and, as such, the following recommendations for its use may prove useful:

1. Scan the entire port range. It is not uncommon for services to be exposed on the ephemeral ports. For the sake of completeness, ports 1-65535 should be examined. The only tradeoff here is time. [-p1-65535]
2. Scan UDP ports as well. Small embedded devices often leverage UDP services to perform core functions. Connectionless protocols require less overhead, which can be favorable when dealing with tight memory requirements. Scanning the entire port range is once again recommended.
3. Go slow. IoT devices are often low power and only capable of so much throughput. A modern laptop may be multiple orders of magnitude faster and more than capable of bogging down the device with requests. Scaling Nmap's speed with the -T<0-5> flag is almost mandatory for accurate results.
4. Keep it simple. Nmap comes bundled with a large number of wonderfully versatile scripts allowing for modes of operation that are meant to evade, circumvent, and trick observing parties or firewalls in an adversarial setting. In general, these scripts won't be necessary for the bulk of scanning needs but can, on occasion, provide helpful results for specific use-cases. Start with simple operation modes first and revise the scanning method to be more complex if needed.
5. Document your process. Save both scanning input and output for a complete picture of what has been done so far. Record, where, what, when, why, and how you are

performing the scans as well as the nature of the output. If you scan again later, compare the results. Looking back on this info at a later time can prove useful if sufficiently detailed.

Nmap is fairly reliable when identifying open services. The output of Nmap will report on what service the port is likely to be providing. Manual verification of the services running on the ports found will generally be performed during the Testing phase.



## 4 - TESTING & RESEARCH

---

To reach this point, it was necessary to gather information, model the system, enumerate vectors of attack, and identify specific services and potential points of entry that may be exploited. Armed with this information, we can now begin transitioning to active testing techniques within a few choice domains. Within the context of IoT, in order to support the ubiquitous connectivity of these devices, the user workflow will likely leverage both mobile- and web-based applications alongside network infrastructure and physical capabilities. The following sections will walk through critical testing categories along with suggesting key areas of focus within. It is critical to note that the IoT device, the companion software, and web interfaces have varied testing methodologies and tools. The end goal of applying the threat model to exploit discovered services still stands.

### 4.1 - WEB APPLICATION TESTING

---

Web applications can range from a handful of simple web pages that serve only to display static content to complex multi-page, multi-flow web applications that handle user input and data management. This variety can make finding a decent starting point for testing a feat in and of itself. The good news is that big or small, all web applications can be broken down into having the presence or absence of the following components: Session Handling and User Account Management, Data Input and Output Management, and Security Controls.

In an effort to keep the focus of this methodology on IoT specific testing, the following components will be discussed from a theme of high-level guidance. The principles behind these testing areas are useful for testing the relationships between IoT devices and their web-based companion applications.

## **SESSION HANDLING AND USER ACCOUNT MANAGEMENT**

Web applications that support user accounts should provide a way to log into and out of the application securely. To manage the sessions themselves, securely configured sessions cookies should be implemented. [13]

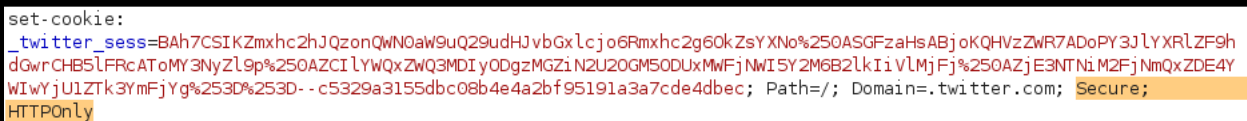
### **LOGIN/LOGOUT**

In the event that the web application is not served over a SSL/TLS connection, check to see if the login functionality is allowing for cleartext submission of passwords. Such a finding is important as cleartext credentials can be intercepted by attackers and used to access a user's account. Password reuse is still a large issue among users and therefore interception of cleartext credentials can lead to breach of user accounts on other services as well.

A good logout scheme will destroy and invalidate session cookies to ensure that the browser will no longer accept them for access to the application and, instead, prompt a user to reauthenticate. Testing can be as simple as logging out and then attempting to browse to an authenticated page with the previously used cookies. Using a proxy tool like PortSwigger's BurpSuite or OWASP's ZAP can be helpful in replaying requests with expired cookies to observe application behavior. [14] [15]

## SESSION COOKIES

Session cookies should be implemented securely. While there are many web development frameworks that provide secure cookie implementations by default, many applications use manually (mis)configured cookies attributes. The **HTTPOnly** flag should be set for any session management cookies as it prevents access to those cookies by client-side scripts like those used in Cross-Site Scripting (XSS) attacks. [4] The **Secure** flag will prevent the browser from transmitting the cookie over an unencrypted HTTP connection. Setting this flag will help mitigate the risk of an attacker intercepting session cookies on the wire with which to hijack a user session. [16] Both of these flags are shown in Figure 14.



```
set-cookie:
_twitter_sess=BAh7CSIKZmxhc2hJQzonQWNoaW9uQ29udHJvbGxlcjo6Rmxhc2g60kZsYXNo%250ASGFzaHsABjoKQHVzZWR7ADoPY3JlYXRlZF9h
dGwrCHBSLFRcAToMY3NyZl9p%250AZCIlYwQxZWQ3MDIyODgzMGZiN2U2OGM5ODUxMWFjNWl5Y2M6B2lkIiVlMjFj%250AZjE3NTNiM2FjNmQxZDE4Y
WIwYjU1ZTk3YmFjYg%253D%253D- - c5329a3155dbc08b4e4a2bf95191a3a7cde4dbec; Path=/; Domain=.twitter.com; Secure;
HTTPOnly
```

*Figure 14: Twitter.com session cookie set using HTTPOnly and Secure flags*

**Note:** Not all cookies need these two flags set. Many cookies are part of third-party frameworks used for ad services and other meta-data tracking. When testing, make sure to narrow down the minimum cookies necessary to facilitate a user session.

## SESSION TERMINATION

User sessions should, in general, provide mechanisms for automatic session termination via time-out in the event that a user has left an inactive session open. This style of session

termination will help mitigate session hijacking attacks, if the application is vulnerable to them, by creating a fixed window in which such an attack may occur.

Whatever the time-out window, the main interest of a tester is to ensure that once the timeout threshold has been reached, the relevant session cookies have been destroyed and invalidated. Using a proxy tool like PortSwigger's BurpSuite or OWASP's ZAP can be helpful in replaying requests with expired cookies to observe the application behavior.

## DATA INPUT AND OUTPUT MANAGEMENT

Few of the applications you discover will solely serve static pages. As such, take note of any input field or file upload functionality. When testing web applications, consider vulnerabilities based on poor data input and output validation and sanitization. Many applications will provide user creation and profile update/account management flows. These are prime targets for stored XSS and Cross-site Request forgery (CSRF) attacks. HTTP GET requests are also good targets for reflected XSS, via parameter tampering, for use in phishing attacks.

**Note:** Take extra caution when testing for these types of attacks. Stored XSS attacks leave behind persistent payloads that may affect both server side and client side operations.

Mobile applications may store sensitive information in a local SQLite database or in XML configuration files. Make sure to verify first that the application allows for backup by inspecting the AndroidManifest.xml file for the line shown in Figure 15.

**<application android:allowBackup="true"**

Figure 15: XML segment found in an AndroidManifest.xml file

This setting indicates that a full backup of the application can be made which includes the local database and XML configuration files. With your device attached via ADB, the following command sequence will help you create a backup and convert it to the tar format using python.

[17] There are alternative ways to view backups, but this method is quick and straightforward and does not require any special purpose software utilities.

```
$ adb shell pm list packages
$ adb backup -f <backup_name> <package-name>
$ dd if=<backup_name> bs=1 skip=24 | python -c "import
zlib,sys;sys.stdout.write(zlib.decompress(sys.stdin.read()))" >
<backup_name>.tar
$ tar xvf <backup_name>.tar
```

Using your preferred SQLite browser, comb through the information in the backed up .db files for any plaintext credentials you may find. It is also a good idea at this point to take note of what kind of Personally Identifiable Information (PII) is being stored on the device. Figure 16 shows a redacted sample of such information. Sensitive information coupled with backup functionality is a great attack vector harvesting information from a stolen device.

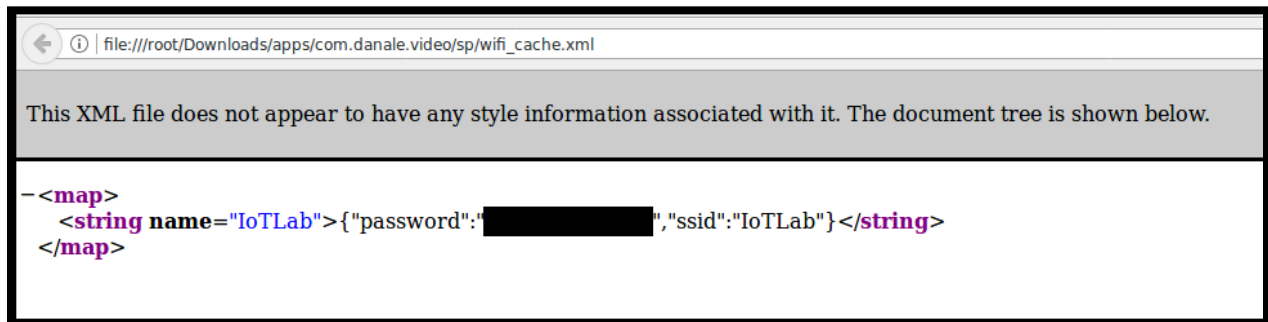


The screenshot shows the SQLite Browser interface. At the top, a dropdown menu is set to 'user' with a search icon to its right. Below this is a table with the following columns: id, alias, photo, password, sign, accountName, token, last login time, is login, and user id. The first row of data is visible, with several cells redacted with black boxes. The 'id' column contains the value '1'. The 'user id' column contains the value '13b4ea484'.

	id	alias	photo	password	sign	accountName	token	last login time	is login	user id
1	1	h[REDACTED]@	[REDACTED]	[REDACTED]	[REDACTED]	ha[REDACTED]mail.com	3.e4789144e37e078b4ec52e3365f8e7d7	1498658226360		13b4ea484

Figure 16: Redacted contents of an Android backup as seen in SQLite Browser

If available, the sp directory in many android backup archives will contain XML configuration files in which plaintext information may be found. An example of these contents can be found in Figure 17 below.



*Figure 17: Backup data from Android application containing sensitive data.*

## 4.2 - MOBILE APPLICATION TESTING

---

This stage of testing can go in many directions. Many applications will simply wrap web requests will purpose built APIs designed to allow the mobile application to communicate with a web application server. This testing will require set-up of a web proxy on the mobile device. This proxy will allow for the interception of web requests and responses between the mobile device and the web application server.

Other applications will have extended device functionality that will require a deeper look at the application binary. This can be achieved using various testing frameworks but essentially requires unpacking the APK files as a simple archive and inspecting the AndroidManifest.xml file for exported functionality and permission requirements. Beyond this, one can use additional utilities like **dex2jar** and **JD-Gui** to browse the java source files. More details about gathering this information can be found below.

## EXPLORING ANDROID BINARIES

### 1. Permission sets

The application will require certain permissions to be granted in order to perform core duties. Do the permission requests coincide with the expected application functionality? For example, a flashlight app should, in no way, shape, or form, be requesting location information, SMS data, call logs, or anything besides the LED flash functionality from the device camera.

### 2. AndroidManifest.xml

After unpacking the APK file with a tool like **Apktool**, you will find a file called `AndroidManifest.xml` which will contain all the information about the application regarding its functionality in terms of activities, services, and broadcast receivers. For the sake of simplicity, we will look at this from the perspective of an unrooted device since with root, you have access to every aspect of the application which can be unrealistic in some situations. That being said, it is worthwhile testing with a rooted device if the individual threat model allows it.

- a. **ADB Activity Manager** (am) can be used to manipulate these exported functions from the command line. This is useful for testing bypass vectors since execution of the activities in this manner does not require them to be visible/ available by touch to a user. For example, can information stored on the device be accessed without logging in by directly calling a read function by name?
- b. **Drozer** is a local client-server application that provides helpful wrappers for ADB Activity Manager. [18] Using Drozer to manipulate function calls can make this

method of testing much easier since it can scan the `AndroidManifest.xml` file to narrow down exportable functions. Exportable functions with null permissions are especially interesting since they can be called independently of others.

- c. **QARK** can be used to create an entirely new version of the APK designed to concentrate all the app functions to the raw activity manager requests seen above. [19]

### 3. Java source files

After decompiling the APK with your chosen set of tools, you can now browse the Java class files that make up the APK itself. It is recommended to first do a search for any hardcoded credentials, IPs, database connection strings, or anything else that may be pertinent given the related threat model. This information may be useful if passwords have been reused in other aspects of the device's connectivity. Moving forward, testing options include reverse engineering the application and recompiling it with custom behavior. Though useful when done correctly, this is slightly outside of the desired scope for this methodology.

The findings discovered from the above testing will range from typical web application vulnerabilities to secure coding issues to general privacy concerns. It is imperative to approach the web and mobile testing from the perspective outlined by a related threat model. It can be tempting to explore certain paths of interest while testing but beware of heading down too many 'rabbit holes'.



## 4.3 - INFRASTRUCTURE & DEVICE

---

Web and mobile application testing is performed when the device ecosystem and related threat models incorporate them. The testing of an IoT device itself will be a constant factor but not every device is guaranteed to have a web or mobile application associated with it.

Nevertheless, if the Threat Modeling & Capability Discovery phases were performed thoroughly, an actionable list of exposed services and their respective risks should have been generated. Leveraging the targets found in this list, the security testers should then be equipped to perform active testing techniques with the goal of verifying and exploiting these vulnerabilities. The following section will refer back to the threat model for the IP Camera scenario put forth in section 2 - Threat Modelling. Leveraging the threat model will help us in assessing the attack surface of the example device.

### EXAMPLE

Recall that we determined that the IP camera from the sample scenario in section 2.2 contained the three core elements of Processes, Data Stores, and Data Flow. For the purposes of this example, consider that the only exposed service on the camera was determined to be a File Transfer Protocol (FTP) server. Fingerprinting the FTP service would likely return version information. Simple Common Vulnerability Enumeration (CVE) database checks would indicate if the identified service is susceptible to any previously enumerated vulnerabilities. These checks may generate 'simple' findings but they are still valid vulnerabilities that should be reported. Once the basics are out of the way, we can attempt to apply our threat model to search for misconfigurations in the identified service. After that exposure is identified, it is

imperative to move through STRIDE threat categories to thoroughly assess the service in full. Security misconfigurations, implementation errors, and use of implicit trust boundaries can exacerbate the vulnerabilities present in a target system. Utilizing the FTP service, the goal will be to identify issues that fall directly into any of the STRIDE threat categories attributed to a Process. The results below reflect actual findings from a device of the same device type.

**Spoofing:** The FTP server runs without enforcement of password authentication and, as such, does not require a login. The privilege level achieved is root.

**Tampering:** The FTP service can be used to push new Shadow and Passwd files to the filesystem, as well as overwriting the configuration files used during system and factory resets. Successful exploitation allows for persistence across system reboots.

**Repudiation:** Further inspection would indicate that logging is not enabled on the FTP service due to storage limitations.

**Information Disclosure:** Various credentials are stored in readable segments of the disk open to FTP transfer.

**Denial of service:** FTP can be used to replace core system files, rendering the device inoperable.

**Elevation of Privilege:** The FTP service runs as root. The user logged into the FTP service has full control over writable sections of disk. Additional users can be added to the system with arbitrary UIDs.

## 4.4 - HARDWARE TESTING

---

Recall the physical inspection of the device from the information gathering phase. The goal of this subsection is to determine the security of the device should an attacker have physical access. In our experience it is very common for consumer IoT devices to be vulnerable to physical or hardware attacks. This section will explore a number of vectors for testing the physical device. Note that when physical access to a device has been gained, it is assumed that it can and will be compromised. The vectors described in this section aim to explore what functionality is available, and to what extent, should physical access be granted.

### TECHNICAL DATA

The technical specifications of a device can help drive your testing. Knowing key information about the locations of physical connectors, chip architectures, unadvertised sensors, and component serial numbers will allow you to tailor your attack vectors to be most efficient for the device at hand. We stress the importance of identifying components during the early stages of testing as the derived information can be paramount in exploiting the physical hardware.

Information retrieved from technical specification and data sheets will be the most useful data going forward. Most of the documents can be found by searching the web for results related to identifiers found on the components. A data sheet on the printed circuit and SoC could produce the following information:

## **1. Pins, Connectors, and Ports**

Connectors or pins are often left on the board for debugging, flashing memory, and factory configuration. These are generally intended for the OEM and service personnel. Luckily, you do not need to belong to either group to leverage their functionality. Connectors such as UART, JTAG, USB, and serial connections may be available. In some cases, these pins may be unmarked. As such, referencing the technical specifications of the Printed Circuit Board (PCB) will likely identify the connections.

## **2. System Architecture**

Documentation on the chipset may disclose the specific architecture used by the device. Should serial ports be present, the baud rate (speed of bit transmission) is likely specified in this documentation. This information is invaluable when attempting to access serial communication ports. Architecture notes will also help you make key assumptions about the capabilities of the chips being testing. Some architectures are very commonly paired with specially designed operating systems/sub-systems and filesystems that will inherently provide certain services while sacrificing others. These sacrifices are important to consider since the limited space and file system privileges available on embedded devices can force you to perform common tasks through unconventional means.

### **3. Endianness**

Endianness is a piece of information that is fairly unlikely to affect the testing process.

However, the tester may encounter situations where this knowledge can be useful for niche testing vectors. This will be most important if attempting to run shellcode through built in HTTP services, debugging binaries on the device, or using hexdump as a makeshift FTP service. Being aware of byte patterns in raw data will help you make sense of what you discover.

### **4. Hidden Components**

Technical documentation about the systems components may reveal that the hardware being used is capable of hidden functions not advertised by product descriptions or disclosed by the OEM. Hidden networking modules such as Bluetooth modules, dedicated AES hardware, and even sensor devices may be attached, functional, yet unused. These 'hidden components' may have unmitigated attack surfaces that are of interest to a security tester.

## **LOCAL STORAGE**

If the IoT device has removable storage (intentional or otherwise), it is worthwhile to verify that some protection has been placed on its contents. If the storage on which the main file system resides is accessible and readable (unencrypted) this exposes the device to an increased number of attacks. Firstly, dumping and cracking the password hashes can be extremely trivial on devices with limited computing power and on devices using deprecated algorithms. Plaintext credentials, configurations, and file contents will be available for analysis and modification. Further, dumping of binary files can allow for fuzzing, debugging, and exploit development. Presence of unencrypted, removable, main filesystems should be considered a

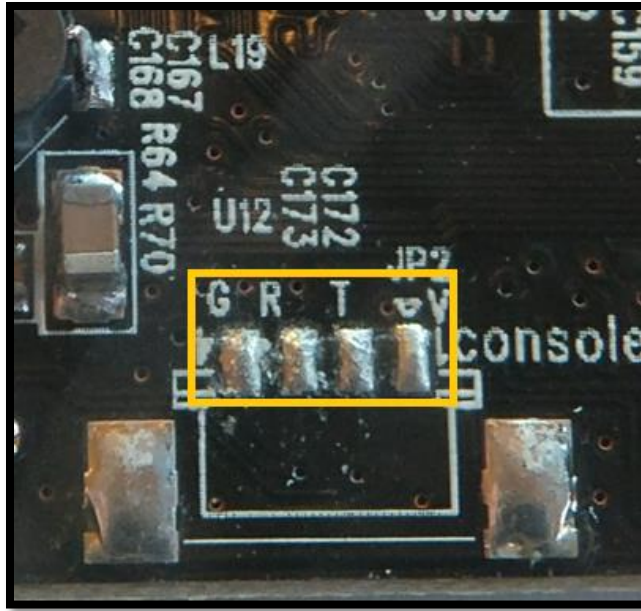
large vulnerability. The network association method may rely on physical transfer of network information to the device. For example, loading a configuration file onto a removable SD card. This may contain sensitive information that could be quickly retrieved and read by an attacker.

## **SERIAL CONNECTIONS**

While other connection methods may exist, prior testing was able to successfully leverage an exposed set of UART pins through which low-level access to a test device was. Reflection upon this experience may provide insight and guidance for replicating such an attack vector.

Utilizing this hardware can give an attacker direct terminal access to the device through a few different means. Primarily, successful connection to the UART pins will allow for a serial connection through which an attacker may execute commands locally on the device. This vector opens up opportunities to enable remote connections services like Secure Shell (SSH), Telnet, and FTP. Once, enabled, these services can provide a much more stable command environment with which to perform further testing. Serial connections often leave root login enabled.

Further, sensitive debugging information can be written screen by the device while connected to the serial connection in this way. This debugging information alone, however, can provide meaningful real time log data about processes running on the device. This is especially helpful when on-device logging functionality is misconfigured or absent altogether. An example of a UART connector can be seen in Figure 18. These pins would require soldering to maintain a reliable serial connection. It should be noted that UART connectors can come in many forms and are not always clearly marked on the PCB.



*Figure 18: UART pads exposed on an IoT device.*

Additional work may be required to locate and configure serial communications such as UART and JTAG, but other ports may provide more immediate access. Various Ethernet, USB and other connectors may be available for use.

## **HARDWARE SWITCHES**

Many devices tote physical buttons that can perform various functions. Reset buttons, for example, will perform different levels of resets ranging from simple reboots to factory resets. Some devices may also include hidden debugging functionality behind standard switches based on duration or frequency of activation. Manipulation of physical hardware switches can revert the device to less secure states. Regardless of specific function, being aware of these exposed switches and understanding the implications of their use will help you define practical physical attack scenarios with which to perform more complex vulnerability analysis.

# 5 - REPORTING

---

There are a few ways that security testers can begin organizing their findings and arriving at conclusions about security flaws. The key to doing this effectively is understanding the audience for which the findings will be relevant. Mainly, we will differentiate between academic findings, vendor disclosure, and those intended for the greater public.

## 5.1 - DOCUMENTATION

---

Regardless of whether you are performing your testing as part of a professionally contracted project or simply exploring the security posture of your own personal network, there are core documentation practices that should be followed.

### 1) Keep good notes in a segregated location

- ❖ Jot down versions, URLs, IPs, device names, service names, etc. either by hand or in a digital format.
- ❖ Maintain a timeline of anything you find interesting and worthy of revisiting.

### 2) Verbosity can be your friend

- ❖ Detail will help jog your memory weeks or months in the future. Expect to rely on this information at a later time.

### 3) Multimedia Evidence

- ❖ Take screenshots, pictures, or video captures of things you find as you go. Notetaking can be time consuming and disrupt a testing flow if done too frequently. Screenshots are a shortcut away and can help you verbalize your testing session when complete.



- ❖ Video captures are especially useful when attempting to log a complex flow of actions.

4) Log your daily actions and next steps

- ❖ While it is important to log what you find, it is also important to log what you did. You don't want to repeat testing you've already done.
- ❖ Based on what you've done so far, make notes outlining next steps, new use-cases and testing directions. Keep yourself moving forward.

5) If testing with a team, ensure that there is a note-sharing platform in use that works for your team. It is important to keep team members informed of testing that has been completed and vulnerabilities that have been found. This will help avoid testing overlap and discontinuity of testing narratives.

**Note:** Maintain an accurate log of device and app versions. Applications may receive automatic updates through the mobile app store/web-server admins during your testing so be sure to remain consistent. Keeping track of the relationship between your findings and developer patch notes will also save you from exploring futile test cases. If knowingly working with older versions, ensure that your findings have not already been patched.

The benefit to following these points is that you will likely have all the information you need to generate a report if or when the time comes. In the event that you need to escalate your research to the point of disclosure, please refer to our section on responsible disclosure. Keep in mind that during testing, you will likely hit 'walls' that will prevent you from progressing with your current attack path. This is okay since every successful test vector will reward you with more information, some of which may resolve the previously encountered blockers. Having

comprehensive notes will allow you to clean up your current testing with the ability to return at a later date without having to re-discover your original footholds.

## 5.2 - FORMATTING

---

When reporting findings, ensure that this minimum identifying information about the device, software, or services is included. An example is shown in Table 3. Most if not all of this information should have already been recorded if the recommendations outlined in this methodology have been followed to this point.

Field	Example
Manufacturer	Camera co
Manufacturer Site	CameraCo.com
Device Name	CU247
Model #	IPCCU2447
Device Type / Subtype	Security & Monitoring > Camera
FCC Id	CU100007
Firmware Version	1.02
Mobile App	<a href="https://play.google.com/store/apps/details?id=net.Camera.Co">https://play.google.com/store/apps/details?id=net.Camera.Co</a>
Photos Of Product	N/A
Description Of Product	IP Camera with Android app client. Supports ONVIF.

*Table 3: Example of the minimum identifying information of the device.*

When documenting specific vulnerabilities, consider the format demonstrated in Table 4. This is a fairly common structure for vulnerability findings as it quickly and simply enumerates results in an easy to read format. Each row provides a different level of technical information which allows for the core impact of the finding to be understood by readers at various levels of technical expertise.

Finding # - Finding Title e.g. Remote Code Execution via Unsecured API	
<b>Vulnerability Category</b>	Include a reference to a well-known framework for vulnerability enumeration/categorization like CVS, CWE, and OWASP Top 10. Doing so will help the reader contextualize the finding before reviewing the technical details.
<b>Target</b>	Include the targets affected by this vulnerability. This can be a list of IPs, web application API endpoints, etc.
<b>Description</b>	
Describe in detail the impact and likelihood of the finding. Try to quantify the level of knowledge and effort required to exploit the vulnerability along with proposed implications of what an attacker may be able to do after successful exploitation. The description should provide a reasonable justification for pointing out the vulnerability.	
<b>Exploit Info / Payload / Etc.</b>	
This section should include all relevant information about specific exploit code and supporting steps required to replicate the exploit. The reader of this section should be able to recreate the conditions necessary to perform the exploit themselves. This is critical for audiences tasked with remediating the finding. Remember: if the reader cannot verify that the finding is legitimate, it probably won't get fixed.	

*Table 4: Template for vulnerability findings*

Finally, when reporting on vulnerabilities take the necessary precautions when submitting, transmitting, or sharing said information. Strong encryption, and confirmed information about the recipient's identity is imperative. If the security tester is weary of trusting the vendor or other disclosure contact, consider leveraging CERTs vulnerability reporting service as mentioned in the section on disclosure.

# REFERENCES

---

- [1] Electronic Frontier Foundation, "Coders' Rights Project Reverse Engineering FAQ," [Online]. Available: <https://www.eff.org/issues/coders/reverse-engineering-faq>. [Accessed 23 May 2017].
- [2] Cornell Law School, "37 CFR 201.40 - Exemptions to prohibition against circumvention," [Online]. Available: <https://www.law.cornell.edu/cfr/text/37/201.40>. [Accessed 8 July 2017].
- [3] C. M. U. -. CERT, "Vulnerability Reporting Form," [Online]. Available: <https://vulcoord.cert.org/VulReport/>.
- [4] IEEE, "Towards a definition of the Internet of Things (IoT)," 27 May 2015. [Online]. Available: [http://iot.ieee.org/images/files/pdf/IEEE\\_IoT\\_Towards\\_Definition\\_Internet\\_of\\_Things\\_Revision\\_1\\_27MAY15.pdf](http://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision_1_27MAY15.pdf).
- [5] Pentest-Standard.org, "The Penetration Testing Execution Standard," [Online]. Available: [http://www.pentest-standard.org/index.php/Main\\_Page](http://www.pentest-standard.org/index.php/Main_Page).
- [6] F. Astroza, "Hacking HI3518 based IP camera," [Online]. Available: <https://felipe.astroza.cl/hacking-hi3518-based-ip-camera/>.
- [7] FCCID.io, "Searchable FCC ID Database," [Online]. Available: <https://fccid.io/>.
- [8] HiSilicon Technologies Co., "Hi3518 HD IP Camera SoC Data Sheet," [Online]. Available: <https://cdn.hackaday.io/files/19356828127104/Hi3518%20DataSheet.pdf>.
- [9] Philips Lighting B.V., "Philips Hue Bridge 2.0," [Online]. Available: <http://www2.meethue.com/en-us/productdetail/philips-hue-bridge>.
- [10] Microsoft Corporation, "The STRIDE Threat Model," [Online]. Available: [https://msdn.microsoft.com/en-us/library/ee823878\(v=cs.20\).aspx](https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx).
- [11] Microsoft Corporation, "Internet of Things security architecture," 3 July 2017. [Online]. Available: <https://docs.microsoft.com/en-us/azure/iot-suite/iot-security-architecture>.
- [12] Asio Ltd. Chirp, "Chirp," [Online]. Available: <https://www.chirp.io/>.
- [13] OWASP, "Session Management Cheat Sheet," [Online]. Available: [https://www.owasp.org/index.php/Session\\_Management\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Session_Management_Cheat_Sheet).
- [14] OWASP, "OWASP Zed Attack Proxy Project," [Online]. Available: [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project).
- [15] PortSwigger, "Burp Suite Editions," [Online]. Available: <https://portswigger.net/burp/>.

- [16] OWASP, "SecureFlag," [Online]. Available: <https://www.owasp.org/index.php/SecureFlag>.
- [17] "How to extract or unpack an .ab file (Android Backup file)," [Online]. Available: <https://stackoverflow.com/questions/18533567/how-to-extract-or-unpack-an-ab-file-android-backup-file>.
- [18] MWR InfoSecurity, "Drozer User Guide," [Online]. Available: <https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-drozer-user-guide-2015-03-23.pdf>.
- [19] LinkedIn, "QARK," [Online]. Available: <https://github.com/linkedin/qark>.
- [20] RTL-SDR.COM, "ABOUT RTL-SDR," [Online]. Available: <http://www.rtl-sdr.com/about-rtl-sdr/>.
- [21] Great Scott Gadgets, "HackRF One," [Online]. Available: <https://greatscottgadgets.com/hackrf/>.
- [22] Laws.Justice.gc.ca, "Radiocommunication Act," [Online]. Available: <http://laws.justice.gc.ca/eng/acts/R-2/FullText.html>.
- [23] GNU Radio, "Learn About GNU Radio," [Online]. Available: <https://www.gnuradio.org/>.
- [24] Watkins-Johnson Company, "FSK: Signals and Demodulation," [Online]. Available: <http://edge.rit.edu/edge/P09141/public/FSK.pdfv>.
- [25] SigIDWiki.com, "Signal Identification Guide," [Online]. Available: [http://www.sigidwiki.com/wiki/Signal\\_Identification\\_Guide](http://www.sigidwiki.com/wiki/Signal_Identification_Guide).
- [26] M. Greenman, "An Introduction to Multi-Frequency Shift Keying," [Online]. Available: <http://www.qsl.net/zl1bpu/MFSK/>.
- [27] C. M. U. -. CERT, "Vulnerability Disclosure Policy," [Online]. Available: <https://www.cert.org/vulnerability-analysis/vul-disclosure.cfm>.
- [28] A. Das, "The Tangled Web of Password Reuse," [Online]. Available: [http://www.jbonneau.com/doc/DBCW14-NDSS-tangled\\_web.pdf](http://www.jbonneau.com/doc/DBCW14-NDSS-tangled_web.pdf).
- [29] OWASP, "HttpOnly," [Online]. Available: <https://www.owasp.org/index.php/HttpOnly>.
- [30] A. Labs, "AppUse," [Online]. Available: <https://appsec-labs.com/appuse/>.

# APPENDIX 1 - SIGNAL ANALYSIS

---

A thorough and complete methodology on capturing and reverse engineering signals would be well outside the scope for this document and is worthy its own project. That being said, capture and analysis of out-of-band traffic is a thoroughly neglected aspect of IoT testing. It is not uncommon for devices to use of these out-of-band communications while associating to a network. Occasionally, you may encounter a device communicating with other devices using a radio frequency distinct from the one(s) used to associate to the network itself. Assessing the communications on such channels can be a difficult task for those not familiar with the process. Sensitive network or personal information can be transmitted through these methods. This appendix outlines our process for basic signal analysis.

The absolute basics of reverse engineering signals can be done with affordable hardware and Open Source Software. For recording and analyzing signals in other RF (radio frequency) bands, a SDR (Software Defined Radio) is recommended. The RTL-SDR is an inexpensive SDR that is capable of capturing RF signals between 25MHz-1700MHz. [20] It is, however, not capable of transmitting but this is not an issue since, in most cases, the legality of transmitting on some radio frequencies is not always in the favor of the security testers. The more expensive option, the HackRF provides a much more comprehensive range of operation, from 1MHz to 6GHz. [21] The HackRF also supports both receive and transmit functionality. Keep in mind that local laws may prohibit the civilian transmission of data on certain radio bands. [22] Most of these devices are sold with a variety of antennas for use depending on the desired signal range and band. These two devices are quite popular and are supported by all the software tools outlined in this

appendix. The software suites that are needed are freely available and open source. GQRX, RTL-SDR, URH, and Audacity are all that are needed to perform the basics listed here. For those willing to learn more about signal decoding, Gnu Radio provides an unmatched set of features for processing complex digital signals. [23]

## OUR PROCESS

---

Successful execution of the Information Gathering phase should provide you with an understanding of test device and hardware capability. Primarily, the tester should be concerned with identifying the operational radio bands of the device being tested. To quickly reiterate the following are ways to identify such information:

- ❖ Read the user manual, if it exists
- ❖ Identify the FCC ID and use a service like <https://fccid.io> to retrieve the specifications
- ❖ Physically examine the hardware, search for markings and locate datasheets

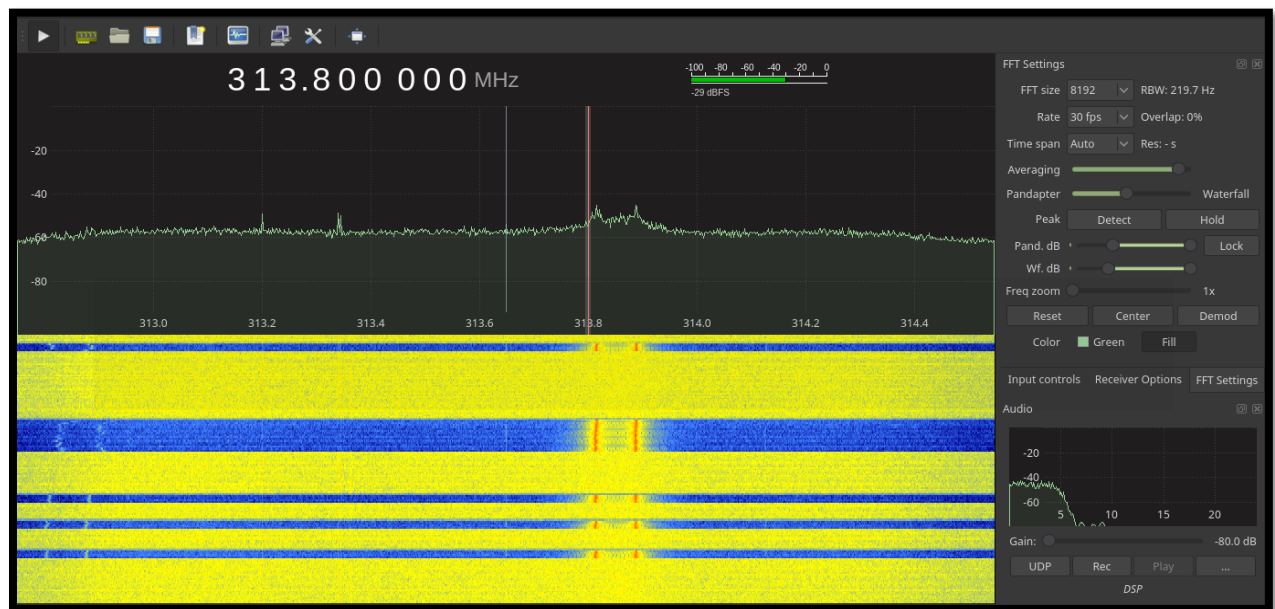
It is not recommended to blindly search common RF bands for device traffic. This literal signal to noise ratio is not in your favor. Once one or more RF bands have been identified record that information somewhere convenient. For example, the datasheet for a sample car key or ‘Keyless Entry Transmitter’ was listed to operate to 313.8 MHz as shown in Figure 19.

Operating Frequencies		
Frequency Range	Rule Parts	Line Entry
313.8-313.8 MHz	15.231	1

*Figure 19: Sample output from fccid.io indicating operational frequencies. [7]*

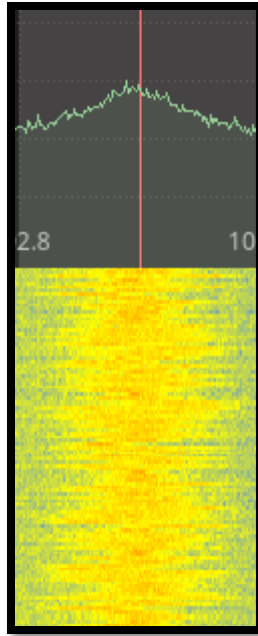


Once a band has been identified, the next step is to verify that it is transmitting on a RF we can capture. For the 313.8 MHz band, that would fall within the range for a RTL-SDR device to capture. Using GQRX, the channel can be set to 313.8 MHz and capture can begin. The activity on the radio band is easily visualized by GQRX. Figure 20 below shows a number of key presses performed with the key fob causing intermittent bursts of transmissions.



*Figure 20: Visualizing RF transmission using GQRX in waterfall view.*

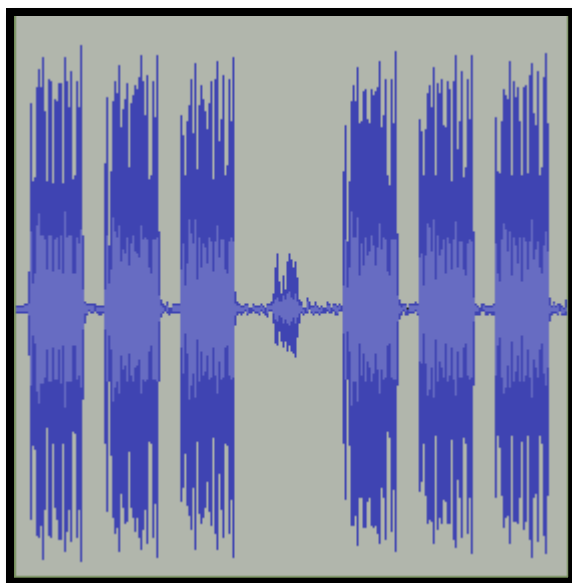
Once the band(s) have been identified, confirm that the band range and sampling rate are sufficient to catch the transmissions. By zooming in on the demodulation band, pictured on the top half in red, we can observe the transmissions by sending signals from the key-fob. Adjust the sampling rate until the signal is distinct. The higher the sampling rate the higher the resolution will be later in the process. The tradeoff of high sample rates is high CPU rates and higher storage requirements. A close up of the waterfall view as shown in Figure 21 shows the captured sampling in greater detail.



*Figure 21: A close up of the waveform and waterfall view in GQRX.*

Once the settings for the frequency, sampling, and gain have been tuned, begin recording the signals. Start the capture, then proceed to start the devices transmission. In this case, that meant pressing the lock button on the key fob. The capture can be stopped by pressing ctrl+c at any time. This raw file can be used to later replay the traffic if the testing hardware permits. For analysis, it is recommended to convert this raw file into an audio format, .wav being the most common. Tools like sox can convert them on the fly with little practice.

Once the signal is in an audio format it can then be analyzed using an audio processing tool. For this application, Audacity provides a substantial set of useful features. Open the file in Audacity and follow the recommendations for creating a working copy. The waveform should be visible in the main window.



*Figure 22: A waveform of a transmission as view in Audacity.*

In Figure 22 the transmission shown starts and stops, seemingly repetitious sequences are also apparent. The similarity between the first and second burst can be assumed to be a result of retransmission. Retransmission is very common for unreliable protocols as it gives the receiving target a better chance of receiving an uninterrupted signal. Through observing the spectrograph and the waveform, the next step is to identify the modulation used in this transmission.

## INTERPRETING MODULATION

---

The following subsection outlines some of the more common modulation methods used in RF transmissions. Before actual data can be extracted from the captured signal, the type of modulation used will have to be determined. For the experienced, the modulation can be recognized almost instantly by the waveform alone. The following three methods of modulation are described and illustrated using the waveform generator supplied with the Universal Radio Hacker suite. As these are digital signals and not analog, the terms FM (Frequency Modulation) and AM (Amplitude Modulation) are not used. Those terms are typically used for analog data, while their digital counterparts are referred to as Amplitude-Shift keying (ASK) and Frequency-Shift Keying (FSK).

### AMPLITUDE-SHIFT KEYING

ASK is the likely the easiest form of modulation to identify. As named, the amplitude of the signal is modulated on & off (~1 & ~0) to represent the digital binary sequence. The Universal Radio Hacker output in Figure 23 does a wonderful job of visually representing this process. The topmost waveform is the *carrier signal*. This refers to the constant signal generated that will be used to carry the information once modulated. The middle waveform, is the digital signal. This square-wave is the information to be transmitted; in this case the binary sequence 1010. Finally, the bottom waveform is the modulated signal. The amplitude of the carrier signal is increased (turned on) to represent the binary 1, and inversely lowered (turned off) for binary 0.

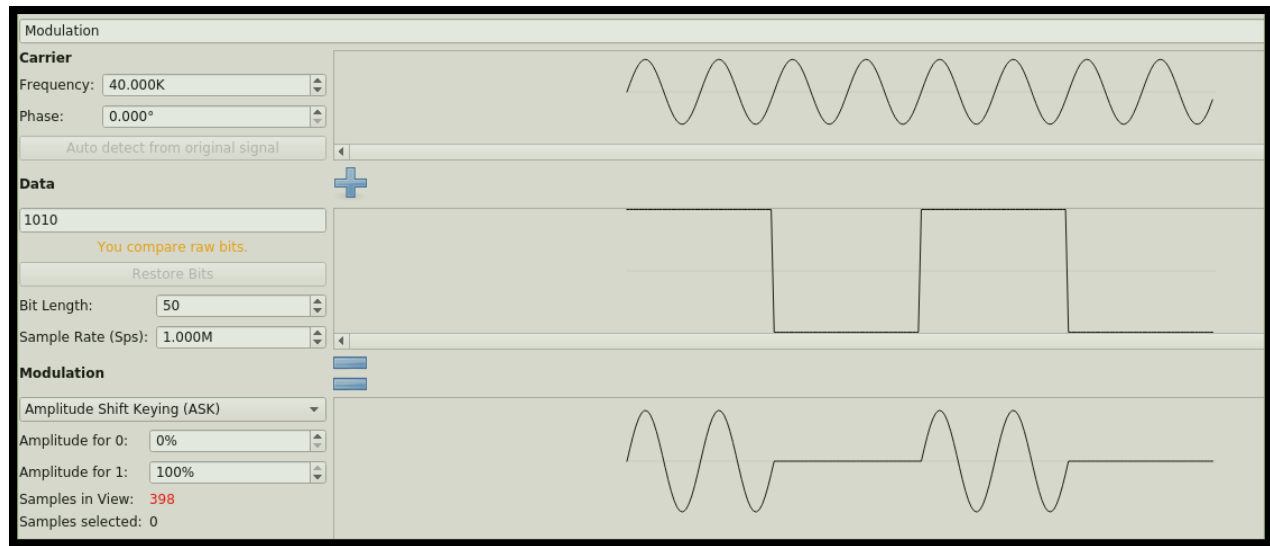


Figure 23: Universal Radio Hacker showing ASK modulation.

Amplitude-Shift keying is fairly easy to decode by hand or with software such as GR and URH.

## FREQUENCY-SHIFT KEYING

FSK is another common form of modulation and will use two distinct frequencies to transmit data. These are conventionally referred to as the *mark* and the *space*, representing 1 and 0 respectively. [24] These binary values are transmitted at an expected constant interval. Once again, a generated signal from Universal Radio Hacker is shown below in Figure 24 to demonstrate the modulation. The carrier signal and digital signal are effectively unchanged from the previous example. In the modulated signal, we can see the 1 (mark) values represented by a burst of the higher frequency signal, and the 0 value (space) by the low frequency.

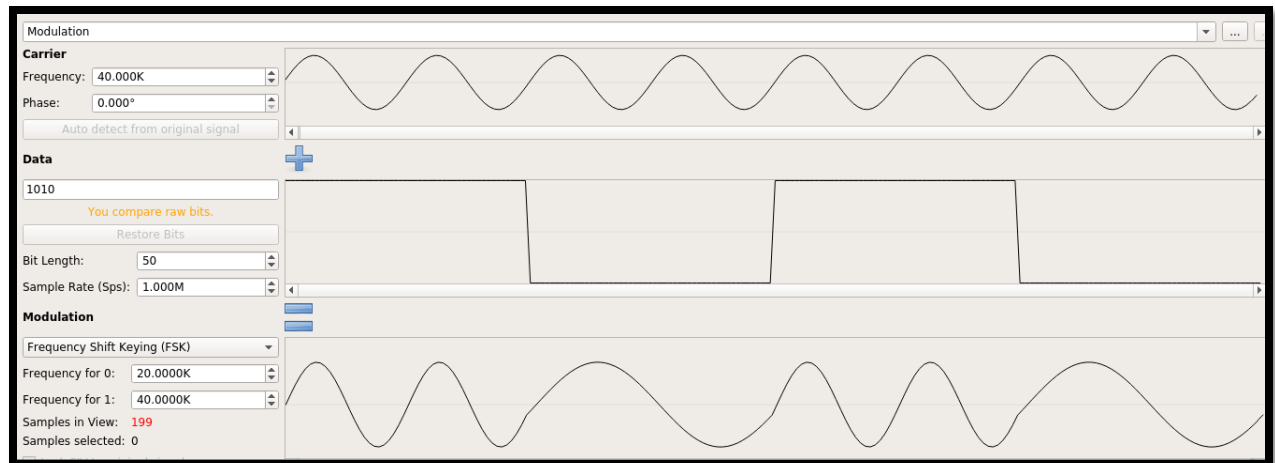


Figure 24: Universal Radio Hacker showing FSK modulation

## PHASE-SHIFT KEYING

Phase-shift keying (PSK) relies on altering the phase of the carrier signal to transmit data. As with the above examples, the carrier wave and digital signal are shown in Figure 25 below. The high and low values are demarcated by the abrupt transition of phase. It may be difficult to determine the originating phase if decoding by hand. This may result in the demodulated digital signal being inverted. Thankfully this is a simple operation to correct; invert every bit. It is recommended to use a software suites like URH for PSK demodulation if possible.

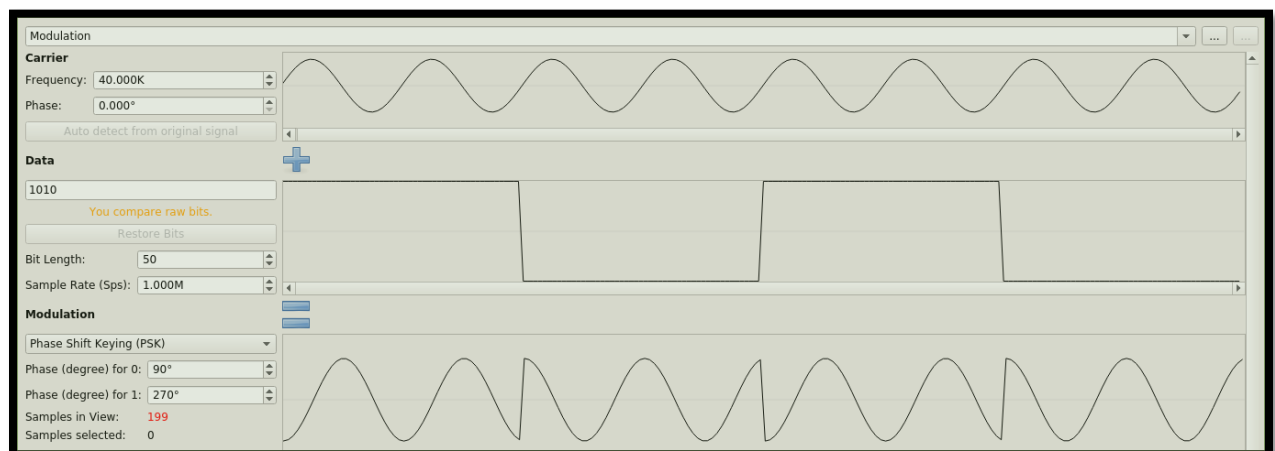


Figure 25: Universal Radio Hacker showing PSK modulation

## LESS STANDARD MODULATION

---

The three types of modulation, ASK, PSK, and FSK are the three major archetypal variants of digital modulation. However, there are countless variants and modifications to these schemes.

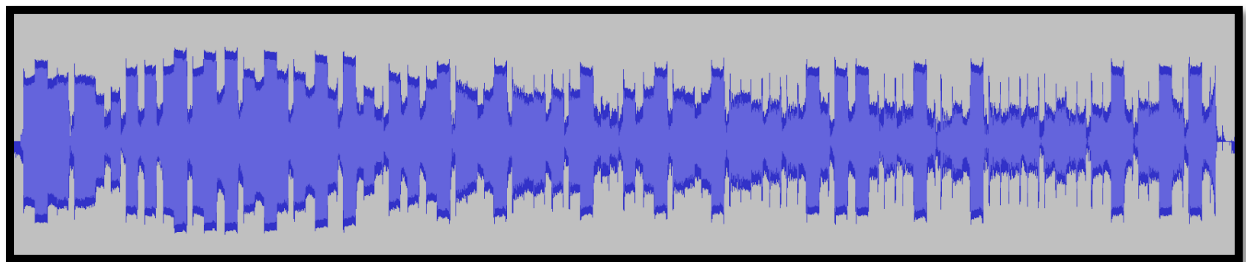
If the signal does not represent any of the types listed above, the [SigIDWiki](#) provides a large number of samples and specifications to assist with identifying less standard transmissions. [25]

More complex transmission schemes will likely require significant processing with suites like GNU Radio or the acquisition of specialized equipment. Time and monetary investment will have to be factored in if this is to be pursued.

## EXAMPLE SIGNAL ANALYSIS

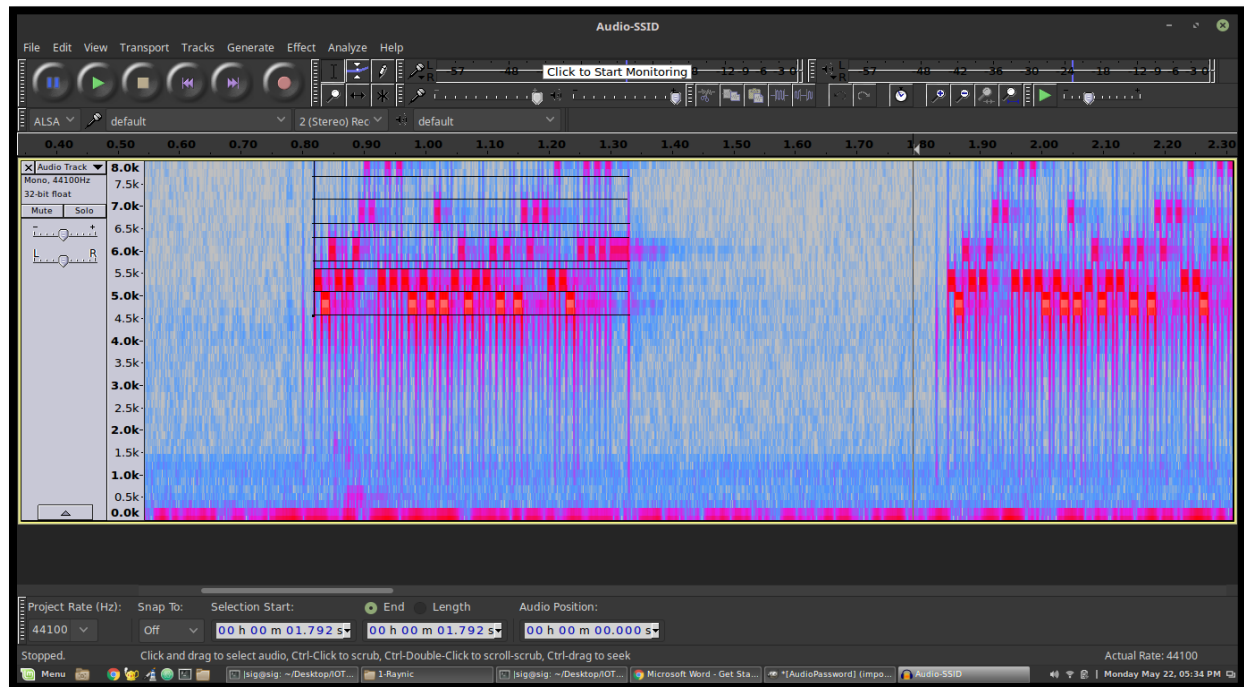
---

The following is an excerpt from one of the many IoT devices that were tested during the creation of this methodology. The signal below was transmitted from a smartphone through a device companion app installed during the pairing process. This transmission utilized the smartphone speakers to transmit this data at a nearly inaudible frequency. A laptop speaker array was, however, able to capture this from reasonable distance. Figure 26 shows the unmodified waveform of the captured pairing process audio as viewed in Audacity.



*Figure 26: The waveform of the captured pairing process audio as viewed in Audacity.*

The transmission shown in Audacity is roughly 1.2 seconds in length. The waveform visualization does not immediately resemble any of the fundamental methods of modulation. After those methods are ruled out, deeper research into modulation identification is required.



*Figure 27: The spectrograph of the captured pairing process as viewed in Audacity.*

When viewed as a spectrograph as shown in Figure 27, the signal appears to be much less chaotic. Five distinct tones that are non-uniform in distribution are visible. Horizontal lines were added to the first transmission to make the tone boundary clearer for analysis. The non-uniform distribution would indicate that weak or no encryption was used. Figure 28 shows a close up of the waveform where the distinction between the tones is clear.



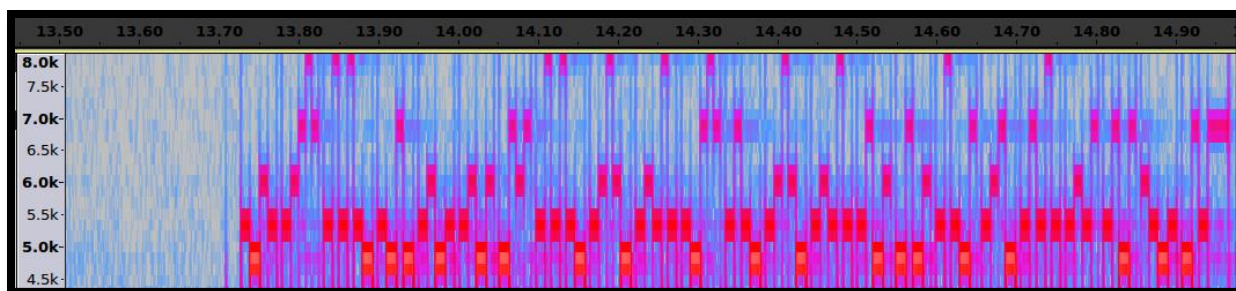


Figure 28: Close up view of the spectrograph tones.

After some research, the signal was identified as being some form of non-standard multiple frequency-shift keying (MFSK). Most MFSK formats will use  $2^n$  distinct tones that each represent  $n$  bits. [26] The cardinality of the tone set not being a power of 2 in this case is atypical. It was assumed at the time that the 5th tone was used as an interrupt tone to separate repeating tones. This assumption turned out to be correct. Therefore, the logical conclusion was that the lowest tone frequency would represent the lowest bit sequence (00). The remaining tones would represent the 01, 10, and 11 bit sequences respectively. Figure 29 shows the work in progress when running a trial demodulation by hand.



Figure 29: Whiteboard decoding of the captured signal

With each tone representing two bits, a byte would be represented by 4 non-interrupt tones.

Following this the tone sequence is demodulated with the following binary sequence being the result:

```
01001001 01101111 01010100 01001100 01100001  
01100010 01111011 01010100 01101000 01100101
```

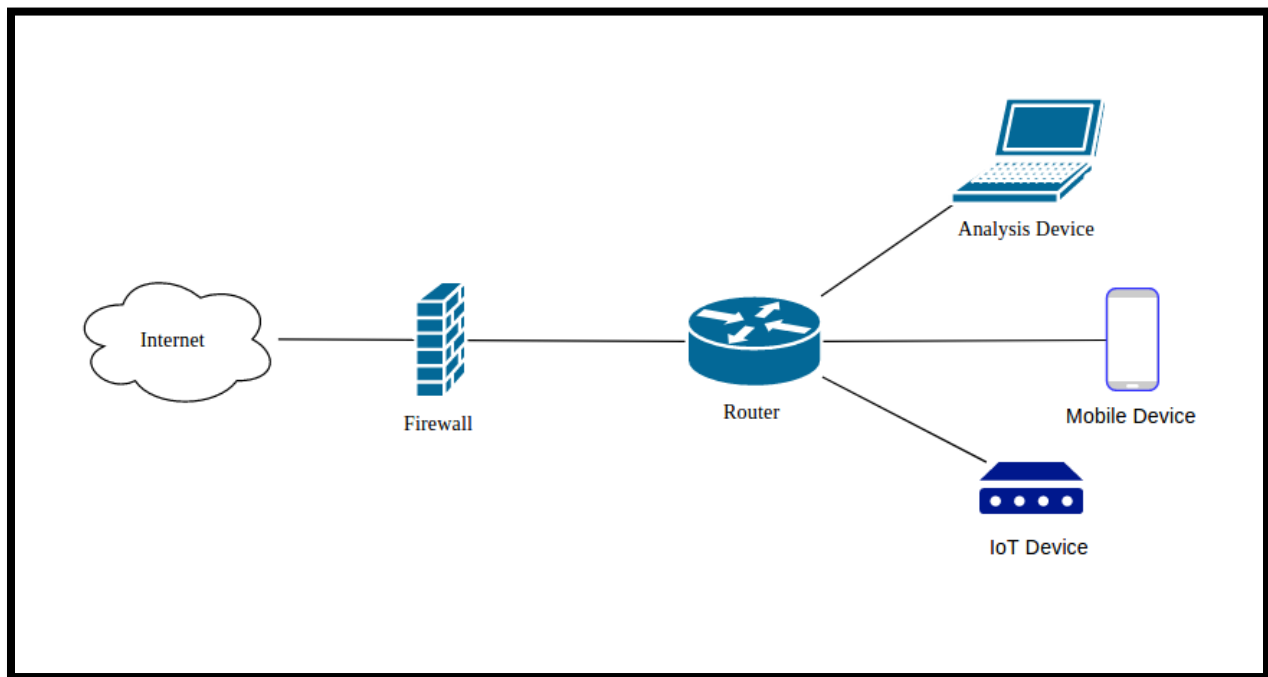
After the conversion to ASCII, the contents of the transmission are quite clear. *"IoTLab{The"*

We can confirm that this is the plaintext SSID four testing network and the start of the correct passphrase at the time of testing. Therefore, the lack of encryption used for the pairing broadcast mechanism is considered insecure.

## APPENDIX 2 - CAPTURING TRAFFIC

---

Conducting analysis on network traffic requires the security testers to capture communications. There are many methods, tools, and opportunities to capture traffic within the testing environment. In this appendix, the details and strategies of doing so will be discussed. Before continuing, recall the recommended setup of the lab. This setup has an internal wireless network in which the IoT device, the companion application, and the analysis device will be contained. The routers second interface is the connection to the larger Internet. For reference, see the diagram in Figure 30 below.



*Figure 30: The reference diagram for suggested lab setup.*

Some planning and understanding of the device to be tested will be needed before arriving at a solution for traffic capture. There are only a few major points of consideration before proceeding.

## **COMPANION APPLICATIONS**

If the device being tested is reliant upon a companion application, consider how that application will be running in your lab. Setting up a mobile proxy on the device to route traffic through a test machine will be instrumental in capturing and altering web requests made by the application. Note that rooted Android Devices will have greater proxy capabilities via apps like Proxy Droid.

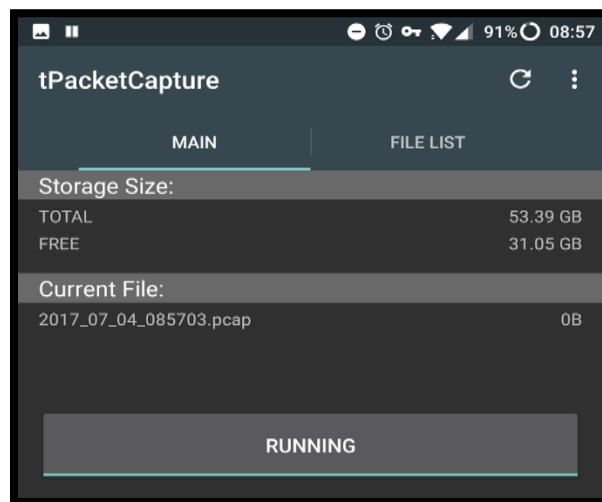
## **VIRTUALIZED DEVICES**

In environments where the companion device is being virtualized or bridged through the analysis device, you will have the option to capture traffic on the host environment. The analysis device should be well equipped to capture all traffic passing through its bridged interfaces. This setup should be suitable for capture and analysis of traffic to and from the companion device only. If greater scope of capture is required, such as traffic between the IoT device and the cloud service, consider changing or adding a method of packet capture.

## **PHYSICAL DEVICES**

For testing physical devices, e.g. Android phone or tablet, consider enabling pcap functionality to capture directly from a physical network interface. This alone will not sufficiently provide visibility of the entire network, however, it should be reliable for traffic passing through the device itself. Android devices will allow for local Tcpdump instances to run and directly capture traffic from the interface. Much like the previous method, this is an acceptable method for capturing traffic explicitly involving the device itself. Even when rooted, the capture method

may struggle with higher rates of traffic. Shown in Figure 31 is the tPacketCapture interface showing a running capture on a mobile device.



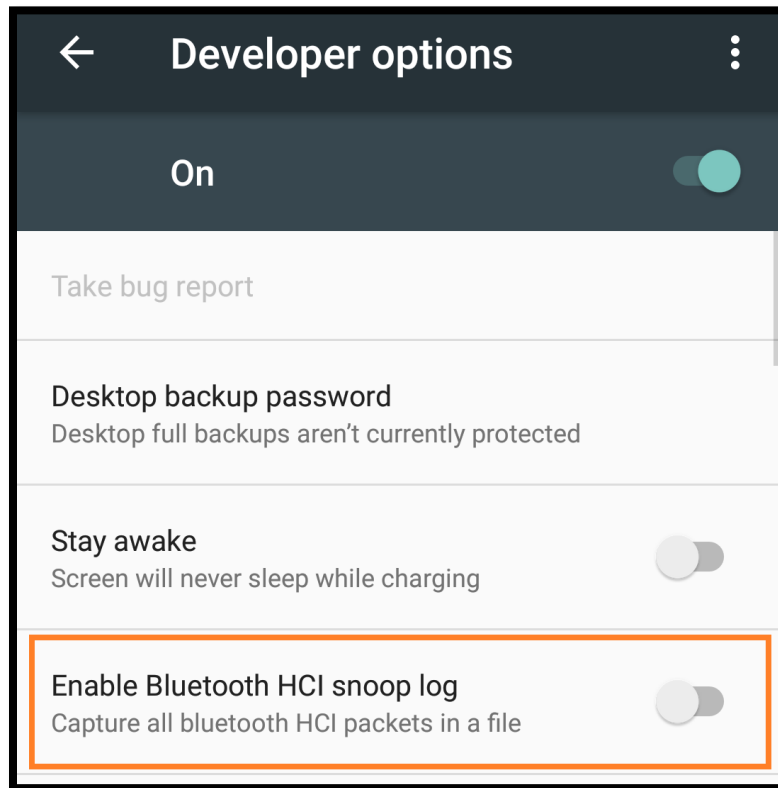
*Figure 31: - tPacketCapture Mobile application interface*

## **BLUETOOTH & ZIGBEE CAPTURE**

If the IoT device, or mobile application use any of the 802.15.x standards such as IEEE 802.15.1 “Bluetooth” or IEEE 802.15.4 “Zigbee”, some adjustments to the capture method may be required. For Bluetooth communications, Wireshark can directly capture traffic from the analysis device providing that it has a Bluetooth interface. Otherwise, investment in a Bluetooth dongle may be necessary. If the Bluetooth communication takes place between a physical mobile device and a paired peripheral, consider logging the Bluetooth activity from the device itself. This can be quickly configured by doing the following:

1. Enable the developer options by clicking Settings > About Phone > Build number 7 times

2. Navigate to the Developer options menu and select the “Enable Bluetooth HCI snoop log” option. This will capture Bluetooth traffic to a btsnoop\_hci.log file. This option is shown in Figure 32



*Figure 32: The Bluetooth logging option in the Developer options menu*

Zigbee communications may be more of a challenge to capture. It will be very likely that an additional dongle or module will be required to capture this traffic. While Zigbee operates in the 2.4GHz band, it will be dropped by most captures, or on occasion appear as malformed traffic. While out of scope for this document, Texas Instruments has a line of SoCs (TI CC2531) that are used to create Zigbee modules and can be leveraged to capture this traffic more reliably. There are a number of capable modules that utilize this chip for Zigbee sniffing. For active RX (receive) and TX (transmit) capability you may need to consider more expensive

modules. Carefully examining chipsets and documentation will indicate if pursuing these peripherals will be required for related test-cases.

**Note:** Do not expect your network card to capture these protocols by default. Wireshark will typically only capture 802.11 traffic without explicit configuration and compatible hardware.

## **PROMISCUOUS CAPTURES**

Capturing in promiscuous mode is ideal for capturing traffic that is not explicitly addressed to the listening interface. Some interfaces, however, cannot operate on promiscuous mode or require elevated privileges to do so. On Linux systems, this can be quickly configured by entering “`ifconfig <interface> promisc`” on the command line.

## **RADIO BAND CONSIDERATIONS**

Some built-in cards may not support 2.4 & 5 GHz but an ideal environment should leverage both frequencies. This can be accomplished with additional NICs (USB, PCIe, etc).<sup>1</sup> - Information Gathering on the devices being tested should reveal which bands will be relevant to your test-cases.

## **CAPTURING DIRECTLY FROM THE ROUTER**

OpenWRT/DD-WRT/Tomato firmware may allow you to run instances of `tcpdumpmini` directly on a router. This will provide the greatest level of visibility into your network’s traffic. This is a fantastic option for capture in the 802.11 band, however, this will be highly dependent on

firmware used. There may also be difficulty in exporting the captures from the router to an analysis machine.

## **CAPTURING OUTBOUND TRAFFIC**

Networks taps are another viable option for capturing wired communications. If using a network tap, location is important. Depending on where your tap is located, you will logically capture more and less data. Keep in mind, though, that while placing a network tap in a high traffic area like at the gateway is tempting, you are likely to drop packets when nearing your peak bandwidth loads. We recommend using more than one tap if your network becomes sufficiently complex. Keep in mind the cost of additional hardware. Network taps are comprised of multiple LAN ports and, when placed between two points on a segment, are configured to clone the captured data to another machine. These can be very helpful when trying to get a precise replication of outbound data on a predominantly wireless network.

## **TCPDUMP (MINI)**

Tcpdump and its variants are lightweight, command line programs for performing packet captures. Often the best and only choice for devices on which terminal/non-GUI interfaces are the primary means of communicating with the device.







[github.com/SiggyD/Project-Hateful-Ferret](https://github.com/SiggyD/Project-Hateful-Ferret)