SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING

Date: June 5, 2021

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Soren Madsen
Jack Schoen

ENTITLED

Jamming Attack Workaround Study

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREES OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING
BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING

Thesis Advisor: Dr. Behnam Dezfouli

Thesis Advisor: Dr. Andrew Wolfe

CSE Department Chair

EE Department Chair

Jamming Attack Workaround Study

by

Soren Madsen
Jack Schoen

Submitted in partial fulfillment of the requirements
for the degrees of
Bachelor of Science in Computer Science and Engineering
Bachelor of Science in Electrical Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 5, 2021

Jamming Attack Workaround Study

Soren Madsen
Jack Schoen

Department of Computer Science and Engineering
Department of Electrical Engineering
Santa Clara University
June 5, 2021

ABSTRACT

The Internet of Things (IoT) is a fast growing industry with strong footholds in the smart home market featuring devices such as the Amazon Echo, Ring security cameras, smart TVs, and much more. However, it doesn't stop there; the industrial sector has begun using smart devices for measurement, automated tasks, and time sensitive communication. Many of these devices have become reliant on WiFi technology and are vulnerable to attacks on the security of the protocols involved.

In this paper, we discuss the details of the deauthentication attack on WPA and WPA2 systems and propose a solution for detection and recovery in IoT networks all included in a single, easy-install device. We evaluated our solution in a smart home environment against variations in packet reception rate, and concluded that our system is effective for small smart home networks. In addition, this solution can be modified in a variety of ways to further the field of IoT security and provide manufactureable, easily accessible smart system security solutions.

Acknowledgements

Thank you to Professors Behnam Dezfouli and Andrew Wolfe for their extensive advise and generous resources. Their commitment, time, and relentless effort were fundamental to the success of this project. We would also like the thank the Santa Clara University Undergraduate School of Engineering for providing us the funds to complete this project. Lastly, thank you to the IEEE and IoT communities for giving us the means to work off past research and developments in the field.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

As devices within the Internet of Things (IoT) have become more popular in different environments, the variety of products available to consumers and industries has increased with the growth of IoT. With the increase in variety, there comes a diversity in quality of software and network design. As of 2018, the average US household had 8 connected devices and that number is projected to rise to 13.6 by 2022 [7]. As growth continues, the design of products tends to emphasize the convenience and efficiency provided to the user. This means that while great solutions are being developed to automate homes and other environments, there has been a lack of foresight into the security risks that an abundance of connected devices and an over reliance on wireless technology pose.

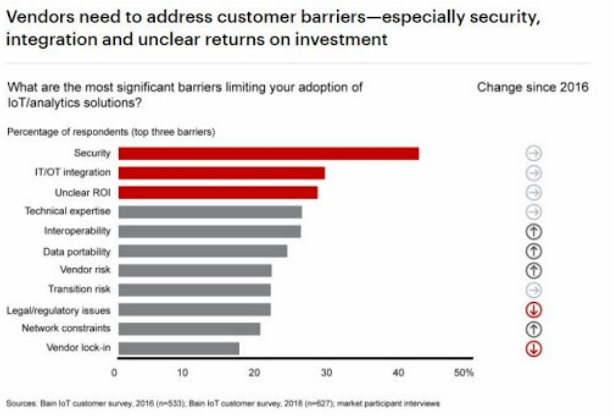

Figure 1.1: Top Consumer Concerns with IoT Growth

Because computer and network security is such a broad area of expertise, this paper will be concentrating on security within the IoT network. In addition, the integrity of a single IoT network is an issue for all users of IoT devices from smart homes to smart cities to automated production and more. The figure above is a graph depicting the change in responses to a customer survey designed to address user concerns for further IoT growth. Security is clearly shown as top growing concern among many. Both companies and individuals with valuable assets intend to maintain

privacy and integrity of their systems.

The largest and most notable presence in the IoT sector has been based in WiFi connected devices. With over 12 billion devices now connected, WiFi connected devices account for 5 billion of the total IoT industry[8]. Yet, with this expansive growth, security technology has not caught up. Many devices today still rely on old forms of WiFi Protected Access: WPA and WPA2. The current management of wireless connects has a number of flaws that we will dive deeper into in this paper. The key takeaway is that many devices still utilize this outdated, and unprotected technology.

1.2 Problem

Despite its release in 2018, WPA3 technology, the next generation of Wifi Protected Access, has still failed to dominate the market. Its predecessor, WPA2, was launched in 2004 and still has control over much of the market including industrial and smart home IoT devices. The IoT typically involves a multitude of devices in a single, making it costly to upgrade the system as technology advances. With the release of the WPA3 standard, protected management frames will become implemented across the board, but many devices used in the smart home and industrial IoT will remain vulnerable to deauthentication attacks.

In industrial IoT environments, we found that despite the availability of WPA3, many devices continue to use WPA2 technology. Because an industrial IoT network typically features many wireless devices, upgrading exclusively the WiFi capabilities on a device is not only generally impossible based on the hardware design, but for networks of hundreds of nodes it simply is worth the time to take apart each device to do so. Thus, many are left with the only option for network protection to be device replacement. This is still costly in both time and money.

In smart home environments, we see a strong parallel in key issues but on an individual consumer level. With the release of popular home security devices like cameras and door sensors from major corporations like Amazon, Ring, and Next, smart home security systems have become more prevalent than ever and the number of connected devices per are expected to rise 13.6 smart devices per home as mentioned earlier. For many families that desire home security systems, replacement is not affordable, but why should security be exclusively for those that can afford it? Through an alternative form of network protection, both the smart home and industrial IoT can be secured in cost effective manner.

1.3 Solution

Our proposed system is a plug-and-play Raspberry Pi network monitoring system designed to detect deauthentication attacks in smart networks. Our goals for this project revolve around three key objectives. First, we wanted to provide a means for deauthentication attack detection in a smart home environment. Second, this device was to provide a simple user interface such that no experience is required to operate the system. Third, we wanted to investigate possible

recovery techniques for working around the attack and restoring the network. The resulting implementation abstracts complex network management and security into a single, easy-install device. Our real-time, efficient design enables fast response time to an attack through an intuitive web interface that makes network management possible for the everyday consumer. With this device, we provide the means for an easily accessible, low cost security solution that reduces the need for device upgrades while simultaneously opening the door for future research in this area relevant to the IoT industry.

Chapter 2

Background Information

2.1 Management Frames in WPA/WPA2 Systems

Before the introduction of WPA3, wireless access points utilized what are called Management Frames for establishing and maintaining connections. In addition, these Management Frames provide functionality for seamless device hand off in larger area networks with multiple access points. There are seven main types to consider: Probe, Beacon, Action, Association, Disassociation, Authentication, and Deauthentication. Via these seven different types, wireless devices are capable of attempting to connect to an access point, receiving information from the access point, disconnecting from an access point, and disassociating entirely from an access point.

While Management Frames initially provided an elegant solution for device management, there are several key drawbacks to consider from a security perspective. All of these frame types are not only unencrypted, but they are also unauthenticated and extremely easy to spoof with just about any form of packet injection tool. Therefore, the functionality of the Management Frames can be leveraged for malicious intent by an adversary through what is called packet spoofing.

2.2 Deauthentication Attacks

In large industrial systems, office buildings, smart homes, and even smart farms, it is not uncommon to find a plethora of WiFi connected devices. Because many of these systems are designed to be low cost and mass produced, the designs feature hardware that may be of an older variety. For most purposes, an older WiFi protocol called 802.11n will get the job done. However, because this is older technology and 802.11w never took off, devices are not implemented with Protected Management Frames (PMFs). As such, an adversary can take down these systems utilizing packet injection.

For example, a porch pirate attempting to steal an Amazon package from a front door equipped with a Ring camera could deauthenticate the wirelessly connected camera from the network and steal the package while camera was inoperable. In another example, an adversary could wreak havoc in an office building by taking down Nest thermostats and sensors. While this does not necessarily warrant the most dire consequences, it violates a fundamental

right to privacy and protection. Home consumers and industrial workplaces alike demand that purchased technology function as intended and care about the security of that technology.

A deauthentication attack in practice utilizes the deauthentication Management Frame which is marked via two parameters in the 802.11 MAC frame header shown below. The field labelled Type, when set to a value of all zeroes, 00 in binary, describes a Management Frame. A Subtype field value of 12, or 1100 in binary, describes a Deauthentication frame. By using these MAC header fields, we will demonstrate how to combine this with other techniques to perform detection of an attack.

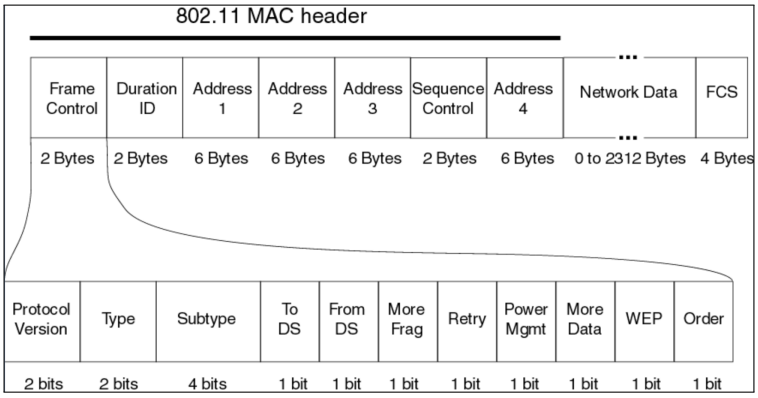

Figure 2.1: MAC Frame Header Information

Chapter 3

Related Work

In this section, we present some related works within the field of deauthentication attack research.

3.1 Session Management System

One proposed solution in [1] to the problem of deauthentication attacks suggests using a session management system which serves to verify deauthentication frames. Since currently management frames are encrypted, they do not require an authenticated user and thus are able to be manufactured by an attacker to seem as though they originated from the client. This causes the client to disconnect from the access point, which leaves them vulnerable as they attempt to reconnect.

Ananay Arora proposes a new solution to this problem in using a secure hashing algorithm, or SHA, which is then used to hash a unique identifier during the association phase of connection between a client and an access point. When deauthentication frames are sent, there is then a check for this unique ID along with the frame as a means to verify the frame. In their paper, Arora proposes making use of Universally Unique Identifiers, also known as UUID, which have 6 predetermined bits and then leave 122 bits for the device's identifier. This leaves a total of 5.3 x 10^{36} combinations of UUID with those bits for clients to utilize. The secure hashing algorithm used is SHA-512 which generates a 512 bit hash. This algorithm is used to hash the UUID that was generated during authentication and the hash is then stored to be used later as needed.

Arora goes on to claim that after creating benchmark results on two widely commercially available devices, they have demonstrated that it is fit for implementation. This application of cryptographic functions combined with the generation of unique tokens allows for a secure association protocol that would prevent deauthentication attacks orchestrated against 802.11 wireless networks. Furthermore, they claim that this protocol is fit for implementation across various access points and clients by simply performing a firmware upgrade, as it was developed as an extension for 802.11 networks.

3.2 Detection of Deauthentication and Disassociation Attacks with a Test Device

This project [9], similarly to our own, designed a network device that detects deauthentication and disassociation attacks against devices within a local area network. This uses a test device on the network that serves to bait an attacker into sending a deauthentication request from an unused MAC address chosen by the test device. Upon receiving a deauthentication request from that specific address, the sender of that management frame is confirmed as attacking the network and steps are able to be taken to counteract the attack.

The way this is set up involves the test device sending first a probe request from the selected previously unused MAC address. Then, the test device replies with a probe response using the test device's original MAC address. Next, the test device sends out an authentication request, once again using the chosen MAC address. It then replies with an authentication response using the test device's initial MAC address. Lastly the test device sends out an association request with its faked MAC address, and then replies to this with its own MAC address sending an association response.

This serves as a way to bait an attacking device sniffing the network into thinking the selected MAC address is an access point, and therefore that MAC address should be used as the source when deauthenticating the test device. If the test device is to receive any deauthentication or disassociation requests from that MAC address, then whichever device used this address is the attacker. Another suggested approach, which incorporates an element of our own design, is to have the test device connected to a wired network as well.

This way, when any sort of disconnection request is sent to the established test device, it can confirm over the wired connection whether or not it received a valid management frame. Our device flips this, using the wired connection to interface with the network, however instead of acting as a bait target it monitors the devices connected to the network to see how the deauthentication packets are being sent. One problem with having a device acting as a test connection to be deauthenticated is that any attack targeting only select devices, not every device within range, may not end up targeting the test device and thus the attack would go unnoticed.

The main difference between this work and our own is the ability to detect targeted attacks. Our device functions as a sniffer on a network, monitoring packets being sent within its range. This device functions as a bobber on a fishing line, so that you know when a fish is on that specific hook. If an attacker decides not to attack the test device, it will continue to presume that no attack is underway on the network. Our sniffer looks for various hallmarks of a deauthentication attack, and uses these to tell if any of the devices on a network are being attacked.

3.3 Mitigation of Deauthentication and Disassociation Attacks

One proposed solution from [11] to mitigate deauthentication and disassociation attacks uses a number of random bits that are generated using the current system time, as well as a counter, to create a pseudo random number which can

be used to authenticate management frames. This solution uses an algorithm, called the MAX algorithm, to combine the seconds, milliseconds, and microseconds of the current system time with the factorial of the counter, but limits the counter's factorial to the remainder when divided by 1048575, the maximum value of a twenty bit number, and then adds a decimal number derived from the counter value. This creates the pseudo random twenty bit number which is then used for authentication between the client and access point.

This authentication is handled by first generating the twenty bit number when the client first connects. Twenty bits were chosen because the last twenty bits of the WLAN MAC layer frame scheme are currently used by the four bit fragment number which is claimed to be unused in all cases, and the sixteen bit reason code which is a fixed value in the case of deauthentication and disassociation attacks. It is stated that, as a result, those bits can all be randomized and used for the access point to issue each client a twenty bit pseudo random number during the authentication response step.

In order for a client to disconnect or deauthenticate, it must send back the twenty bits issued in the authentication step. The purpose of this is so that any malicious third party is unable to send the proper deauthentication frame including these last twenty authentication bits. If the last bits are not a match between the deauthentication request and the access point, then the client stays connected and deauthentication fails. In their testing, this system generated one thousand distributed numbers, without any duplicates. They claim that this system will also slow down an attack by more than an hour, which is a substantial amount of time to slow an attacker, especially if combined with a detection system to allow for a response to the attack.

This project is differentiated from ours because in this work the group sought to delay and ideally mitigate attacks without attempting to identify when an attack occurs. By delaying an attack by approximately one hour, they buy time to detect it, but without any system in place to do that detection an attack could still continue on. Additionally, while this system for mitigation requires reallocation of bits from the WLAN MAC layer frame scheme, which would require changes across all the devices, our system works as an adjunct to the network. So while other solutions would require broader acceptance for implementation to be feasible, our intrusion detection system could simply be added without any other changes to network functionality. That is just one problem with this system.

Another problem that may need to be addressed is the storage of authentication bits. It declares that the pseudo random number of twenty bit length will be stored alongside the MAC address of the client it is allocated to, in "a specific table." However, not discussed is where and how the client-side storage will be orchestrated, which is problematic because this storage would need to be made universal across all clients. Additionally, it does not mention what may happen in the event of an error where one side forgets or loses their copy of the twenty bits, and how a client would be able to disconnect and if it would be able to re-authenticate in that case. Without being able to deauthenticate, a client would have routing and handoff issues when it comes time to connect to a new access point.

3.4 Defense Against Deauthentication Attacks on 802.11 Networks: The Letter-Envelope Protocol

This proposed solution [10] to deauthentication and disassociation attacks involves using a one way hard function which serves to verify the legitimacy of deauthentication frames. This solutions uses prime numbers randomly generated on both the client as well as the access point to enable a form of authentication for disassociation or deauthentication frames sent between the two nodes.

This is done by taking the generated primes on the client side, and multiplying them together to form N1. Likewise, the access point's generated primes are multiplied together to form N2. While in the authentication step, the client and access point exchange N1 and N2. For the client to disconnect from the access point, the client must send the first prime used to create N1. Similarly, for the access point to disconnect the client, it must send the first prime of N2. If there is a mismatch, then there is no disconnection.

Because the generated primes are large, this scheme makes it very difficult for fake primes to be created and simply dividing N1 by a number is easily detectable. These numbers are very hard to duplicate, as N1 and N2 can only be created from the randomly generated primes. This provides security, but is not without drawbacks. A. Arora goes so far as to claim that this system could even find itself vulnerable to DoS attacks if the attacker repeatedly sends association requests repeatedly using different prime numbers and spoofing different MAC Addresses. Another issue is that firmware upgrades were required in order to handle the cryptographic scheme when this protocol was originally announced. While newer systems can handle this burden, it was never widely implemented and has not made a resurgence.

Like some other proposed methods of mitigating deauthentication and disassociation attacks, this project covers a separate area of defense from our own work. While we seek to identify, this method looks to prevent and mitigate attacks by authenticating the packets that make them possible. Another commonality between the letter-envelope protocol and some other mitigation techniques, storing a sort of key is required, which would need to be implemented in clients and access points to enable this technology.

3.5 Queuing Deauthentication or Disassociation Requests

This work [3] aims to detect and mitigate deauthentication or disassociation attacks by queuing packets for a set amount of time, so that the following packets can be observed to determine the nature of the packet attempting to disconnect a client from an access point. By keeping deauthentication or disassociation requests in a brief queue of five to ten seconds, an access point can see what types of packets follow the queued requests. If data packets come after deauthentication packets, then the system knows that it can disregard the deauthentication frames. This is because no client would want to disconnect while still attempting to exchange information.

There are, however, some issues with this solution to deauthentication attacks. The first comes into play when transferring between access points. Because the packets responsible for disconnecting the device as it transfers from one access point to another will be delayed, there can be problems routing the packets through the new access points. Another issue is that an attacker could keep a client connected by spoofing packets to the access point appearing to be from the client. This issue could potentially be addressed by combining with solutions which check to see if a packet is from a legitimate source.

This project encompasses a larger range of functionality when compared to our own work. However as a trade off for being able to detect and eventually mitigate deauthentication attacks, there are routing and hand-off issues as well as the possibility of a malicious user spoofing packets to keep a client connected to an access point. These trade offs come from the attempts to mitigate deauthentication attacks. For this solution to be implemented both the client and access point need to have modified protocols for dealing with incoming packets, which is another large difference from a system like ours.

3.6 Reverse Address Resolution Protocol

This protocol, as referenced in [4], is used to map MAC Addresses to IP Addresses. This is useful in the case of deauthentication and disassociation attacks because they frequently use spoofed packets. These packets appear to come from one MAC Address but in reality are from an illegitimate source. If the IP Address is not also spoofed, using the reverse address resolution protocol will show multiple IP Addresses for the same MAC Address which is impossible, indicating that spoofing is taking place.

3.7 Sequence Number Analysis For Detecting WLAN MAC Spoofing

The goal of this work [13] is to utilize the nature of sequence numbers to be able to detect WLAN MAC spoofing on a network. The background which enables a detection mechanism like this stems from the allocation of twelve bits for a sequence number which enables frames to be fragmented with a fragmentation number that indicates which order the fragments ought to be arranged. When a whole frame is sent, the sequence number advances to the next number in the sequence, wrapping back to zero after reaching 4095. This number allows for an identification of a flow of frames, and an out of order sequence number can be an indicator that a packet may not be from the source it claims to be from.

By monitoring the sequence numbers, you can also detect if a singular host is creating traffic that is designed to appear as though there are multiple sources. These packets may have various MAC Addresses, however if their sequence numbers align then it is likely that they all stem from one source. Similarly, and with more relevance to our project, if there is a stream of network traffic being monitored and there are erroneous sequence numbers on frames being sent, likely the frames that don't match are from an illegitimate source and can be flagged as possibly from an

attacker.

This work resulted in a solution that was able to determine whether or not an attacker is spoofing a MAC Address based on its sequence number and if it fits the pattern of sequence numbers. However, to do this you must know with some certainty what the pattern of sequence numbers is. Additionally, if the attacker is able to sniff the sequence number and then use the expected sequence number, it will likely go undetected. If the deauthentication frame is received before the frame with the identical sequence number then the attack will be successful. Much like our own work, this system uses a kind of sliding window to check frames. Our window monitors packets sent in close proximity, but this window is monitoring the sequence numbers of packets to pick out the anomalous packets.

3.8 Sequence Number-Based MAC Address Spoof Detection Device

This work [6] proposes an algorithm which can be implemented on an external device to leverage sequence numbers from the link-layer header of IEEE 802.11 management frames to detect spoofing without modifying access points or wireless stations. As discussed in the above section, the sequence number is incremented each time a new frame gets sent out. It follows standard set forth by the IEEE to distribute these numbers based on a counter variable that is modulo 4096 to limit the maximum size of the sequence number to the twelve bits available to the field.

The proposed algorithm incorporates possible normal deviations in the sequence number as a way to increase accuracy over simply assuming any packets out of order are spoofed. Since any sequence number that is smaller than the current sequence number should be a re-transmitted frame, it should share the contents with the previously sent frame that it shares a sequence number with. Since spoofed frames keep duplicating the same sequence number, as new frames come in this sequence number would fall further and further behind what is considered to be normal error and re-transmission. This should lead to them eventually being identified as a spoofed frame. Other ways frames might not appear as sequentially as one might expect are when frames are lost, or sometimes they just are out of order. The algorithm proposed by F. Guo and T. Chiueh incorporates all of this to minimize errors while maximizing successful detection of an attack.

The first thing this algorithm does is compute the gap between the sequence number of the received frame and the previous frame sent from the same source node. Since in general the frame sequence numbers were at most $N + 2$ with N being the previous frame's sequence number, anything outside of that would be spoofed, unless it has been re-transmitted. Re-transmitted frames have a sequence number smaller than or equal to the current sequence number, and the team found frames being as far behind as $N - 3$ were normal. They discovered that a four frame gap was the maximum that would occur in their testing environment. If the current frame is a re-transmitted frame then it can be checked against the last copy of the frame, by sequence number. If one sequence number was found to belong to two frames, then one of the frames must have been spoofed. An additional rule put into place was that any data frame

found to be out of order that was not following either a beacon or probe response frame is likely to be spoofed as data frames are not usually out of order. Putting this together they created a verification process to check for frames with larger gaps than $N - 3$ through $N + 2$. This verification process involves noting the current frame's sequence number, then sending an ARP to the frames source station. It then verifies the frame based on where it is relative to the last and current sequence numbers, and if it is out of the acceptable range as set forth above.

The team had promising results from their testing of the system, with a false positive rate of zero. Beyond this, the false negative rate was nearly zero and it was stated that in the worst case it would detect spoofing activity but would not detect every spoofed frame. However they noted that because sequence numbers do not always follow the incrementation patter, false positives or negatives are possible.

As for implementing this algorithm, existing hardware could be used however in that case firmware would need to be modified which is not a simple task and has prevented implementation of solutions in the past. Instead, if the focus shifts solely to detection, then a separate device monitoring the network can be used which does not require changes in firmware and can be implemented alongside the existing network setup. This is very similar to our work, which implements a device to supplement the existing network and add a layer of protection through identifying the presence of an attack. Both of our works utilize a device acting as a sniffer, which restricts us to detection without the ability to perform actions on the network. The main differences lie in that we are observing timestamps while this system looks at sequence numbers, and that while we focus on deauthentication attacks this group is searching for any spoofed frame.

3.9 MMDS: Multilevel Monitoring and Detection System

This proposed solution [5] attempts to develop a self-adaptive system that in real time will analyse and monitor the network, detect attacks, and respond to intrusions. The authors claim that it will fill a gap left by most intrusion detection systems where they are not detecting distributed attacks. This system seeks to learn from normal behavior, where the system is not facing an active attack. The authors D. Dasgupta, J. Gomez, F. Gonzalez, M. Kaniganti, K. Yallapu, and R. Yarramsettii recognize that normal system behavior changes over time, so their system must likewise adapt to the network as time goes on.

The way they go about this is by introducing four agents. The first is the Manager Agent. Its job is to coordinate the other agents and act as the interface for users to interact with. The next is the Monitor Agent. This agent's job is to detect anomalies by taking in information from the target system. The third agent is the Decision Agent, which serves as an engine for a fuzzy interface. This allows it to take a robust decision in the case of network abnormalities or intrusions. It provides imprecise, or fuzzy knowledge to the system. The last agent is the Action Agent, which has the role of generating alerts or heartbeats to pass along to other systems so they can take action. Some examples of

action include killing a process, or disabling access for a user. This whole system bases all of its functionality on fuzzy logic, which is checking sequence numbers.

Once implemented, this system was able to successfully detect simulated attacks in a wireless network. However, due to the nature of the system, the researchers advised that more testing would be necessary to fully determine the success of such a system. While this work serves a similar function to our own, the methods used are starkly different. While this system must learn a network's normal behavior in order to differentiate an abnormal situation, our detection system looks for hallmarks of a deauthentication attack which remain consistent despite varying normal network conditions.

3.10 Other Solutions to Disassociation and Deauthentication Attacks

The solutions below have been accumulated in [2] by B. Aslam, M. Hasan Islam, and S. A. Khan.

3.10.1 Ignore Disassociation Requests

A solution brought up is having the access point and wireless clients ignore disassociation requests. This will ignore all legitimate and illegitimate disassociation frames, claiming that the association will time out on its own. A problem with this is when a client moves from one access point to another and must disassociate with the first before attempting to join the new access point. This would cause handoff problems.

3.10.2 Authentication Before Association

It is proposed in this solution that by authenticating before associating, keys will then be available for encryption and authentication of management frames. To do this, Authentication Information Elements must be added to management frames. Adding these enables the authentication required to protect against spoofing disassociation requests. However, this also may incur additional delays from the increase in processing power necessary to execute this.

3.10.3 Authenticating Disassociation Notification

By authenticating management frames, in particular disassociation requests, there is a layer of protection against attacks that would otherwise be able to simply spoof requests. While this protection is valuable, it also brings the need for increased processing power as well as a solution for storing the keys used for authentication.

Chapter 4

Proposed Solution

4.1 Overview

In many modern security architectures, a common practice to monitor the status of a network and its associated devices is to utilize a packet sniffer. A sniffer is essentially an omniscient device that is capable of viewing all network over a specific medium. In our solution, we focus specifically on the sniffing of 802.11 Management Frames over WiFi. By utilizing a packet sniffer, we are then able to detect and classify malicious Management Frames within normal smart home traffic. Based on this analysis, we update a web dashboard that is hosted on the local network for network management and security alerts.

4.2 Packet Sniffing

As mentioned before, packet sniffing is a means to capture all network traffic within a specific medium of communication. In performing a packet sniff or packet capture, there are two modes of operation that determine the function of the sniffer as it relates to the network protocol stack. The first of these modes is called promiscuous and this captures frames from the network layer and up. In these types of packet captures, one will typically see IP addresses, DNS messages, and other traffic related to a specific WiFi network. In promiscuous mode, a device must be connected to the network it is seeking to sniff. Because this mode of operation captures packets from the network layer, the MAC header information is unable to be parsed as it has already been discarded. Without the MAC header, a test for a deauthentication attack cannot be properly performed. Thus, we utilize the second mode called monitor mode. Monitor mode allows for the sniffing of all wireless traffic in a surrounding area. The only limits are signal attenuation and antenna design. When a network interface card (NIC) is put into monitor mode, all functionality as it relates to an internet connection is turned off so that traffic can be captured and processed. Monitor mode allows us to bypass parts of the protocol stack using the netlink module in the kernel-space and send all of our packets to the user-space in a quicker manner as the packet doesn't have to be processed in each layer of the stack along the way. The figure below shows two forms of network communication in which the monitor mode interface bypasses the typical protocol stack

whereas the other connection traverses this stack for each packet received.

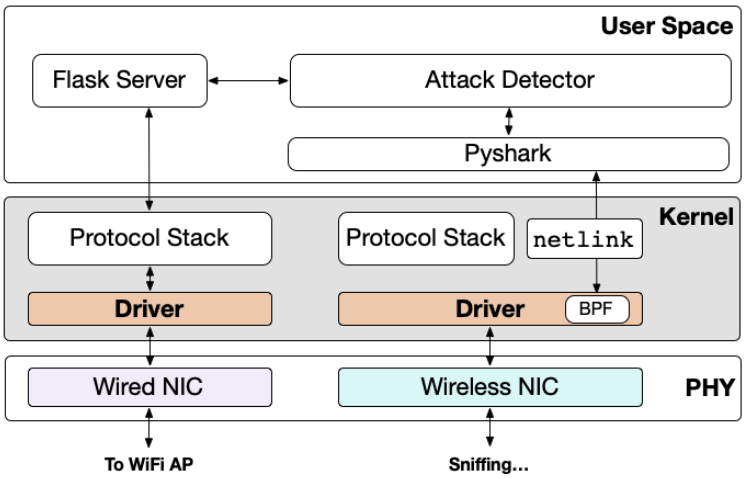

Figure 4.1: Typical Network Traffic versus Monitor Mode Capture in a Device

To perform packet sniffing, we utilize a library called libpcap that is specifically designed to perform what are called packet captures of live network data. Libpcap is an API written in C designed to provide a high level packet capture interface such that even packets not destined for the receiving host can be recorded. Based on this library come two more projects called tshark and Pyshark. Tshark is a software tool with command line interface that ports the utility of libpcap to a network administrator insuch a way that packet captures can be done in a single command with immense functionality. From this CLI tool came Pyshark, a Python wrapper for tshark that uses the netlink kernel module the libpcap API to access and convert sniffed packet data into ”Packet” objects with fields addressable using common display filters that can also be used in other sniffing software such as Wireshark (a GUI version of tshark). Using these ”Packet” objects, we are then able to process and monitor the network for various traffic characteristics to determine the status of a deauthentication attack.

4.3 Berkeley Packet Filter

Over the course of our project development, we discovered that in population dense areas, there is a proportional density of wireless connected devices. As we began packet sniffing for attack packets, we discovered an unusually high load on the processor of our sniffer node. This processor load was due to an excess of frames being received by sniffer. To address this issue, we implemented a Berkeley Packet Filter (BPF). A BPF is a user-space program that is designed to allow user-space processes to load a filter program into the kernal-space specifying parameters that determine which packets should be passed up from the link-layer in the network stack. By dropping packets in the link-layer that do not match the requirements of the BPF, the sniffer node’s Python program running in user-space only has to process packets relevant to the goals of deauthentication detection and network device management. By

reducing the total network traffic analyzed by the program, real-time monitoring is done in a more efficient manner and the processor load on the sniffer node is greatly reduced. Because our system is designed for IoT environments, it is imperative that we conserve computing resources and power consumption as much as possible.

4.4 Attack Identification

To detect a deauthentication attack, there are two key characteristics that we determined were most useful in identifying whether a packet is legitimate or part of an attack. The first and primary characteristic of a deauthentication attack is a flood of management frames labeled for deauthentication that not only disconnects a device from the network but ensures it stays disconnected by maintaining the flow for a duration of time. A malicious flood will typically exhibit a high rate of management frame transmission by sending a dense succession of deauthentication packets directed at a particular device.

To address this, we found the MAC timestamp field to be particularly important in determining an attack. The second key characteristic of a deauthentication attack we considered in our design is the presence of association frames from a device not recognized as a current connection during a flood. In many cases of deauthentication attacks, an adversary may be seeking to obtain a copy of the 4-way handshake in an attempt to break the encryption and gain the password to the network. As such, it's important for our system to consider this attack vector and alert the administrator of such a possibility.

4.5 Experimental Setup

In our experimental setup, we utilized four main components to simulate a deauthentication attack and test our system: a wireless router, a victim node, an attacker node, and a sniffer node. The wireless router is a TP-Link AC1750 running other smart home devices such as a Echo Dot and Google Home. The victim node was connected wirelessly to the access point, while the sniffer node was connected via a Local Area Network (LAN) connection for the purpose of hosting the Flask server on the local network while the wireless NIC was occupied with monitoring the network. Because our design is intended to avoid the introduction of a bottleneck on the network, the sniffer node will sit as an omniscient third party to all traffic within the network. This also means that the system is not equipped to handle the prevention of an attack. However, through detection and the presence of a web server dashboard, our proposed system is meant to be a real-time monitoring system used to assist with identification and recovery.

The sniffer node runs code designed for Python 3.8 utilizing the aforementioned pyshark library to process packets in real time via the provided LiveCapture() function with an enabled BPF for filtering out packets not directed to or sourced from the MAC address of the access point. This improves the run-time efficiency of the code and reduces processor load. As each packet arrives, the MAC address of the client, is parsed from the frame header and added to a

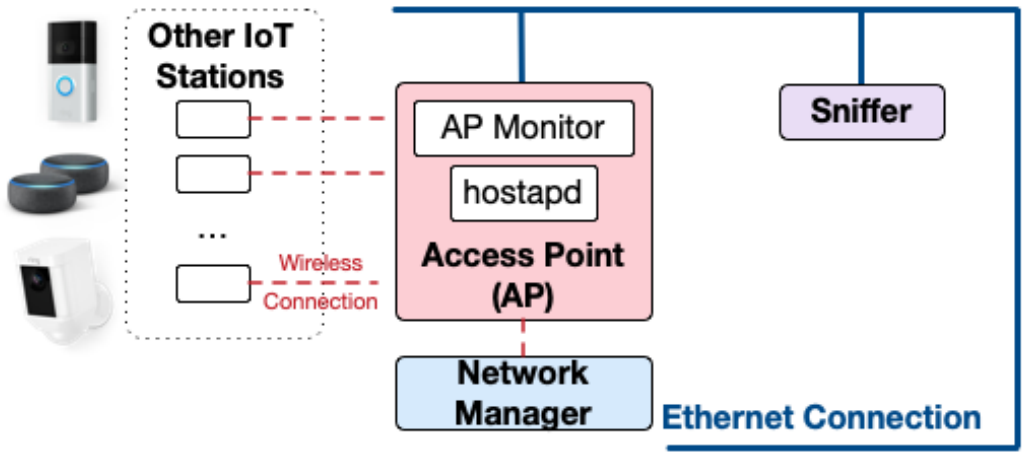

Figure 4.2: System Network Architecture

list of current connections that are monitored by the IDS. If the MAC address appears in a disassociation frame, then the device status in the dashboard list is updated to disconnected. If a packet is not a deauthentication or disassociation frame, no more processing is performed on the packet.

When the sniffer encounters a deauthentication packet, a key is added to a table containing a data structure pertaining to the client address that manages a log of all deauthentication frames received. If other deauthentication packets are received within the specified duration, they are added to this log. In the case that the rate of deauthentication packets implies a flood, an attack alert is raised. A flood, in our tested implementation, is defined as having started when there are more than five deauthentication frames are detected within two seconds. At this point, the system tracks all deauthentication frames until a timeout of three seconds is reached in which no frames are detected for the specific victim device. The attack is then classified as ended, and information about the attack is logged. In the other case where our device is not classified as under attack, the system updates the device status to disconnected in the device list on the dashboard.

All alerts and logs are reported in real-time to a Flask server running concurrently with the IDS (see figure 4.2 for user interface design). The sniffer node and Flask server is designed such that it runs without the use of a wireless connection. It is connected via a patch cable to an Ethernet port on the access point. We used this approach because using a wireless connection for the sniffer also made the node susceptible to the same attacks it attempts to identify.

In our dashboard design, we present only key information to the user to facilitate clear communication and provide an easy to understand breakdown of the network. Three main components are displayed on this page: the list of devices connected, a log of all attacks, and a main alert status. A network supervisor is able to receive all important information about an attack performed on a downed device such as when it started, when it ended, and its duration.

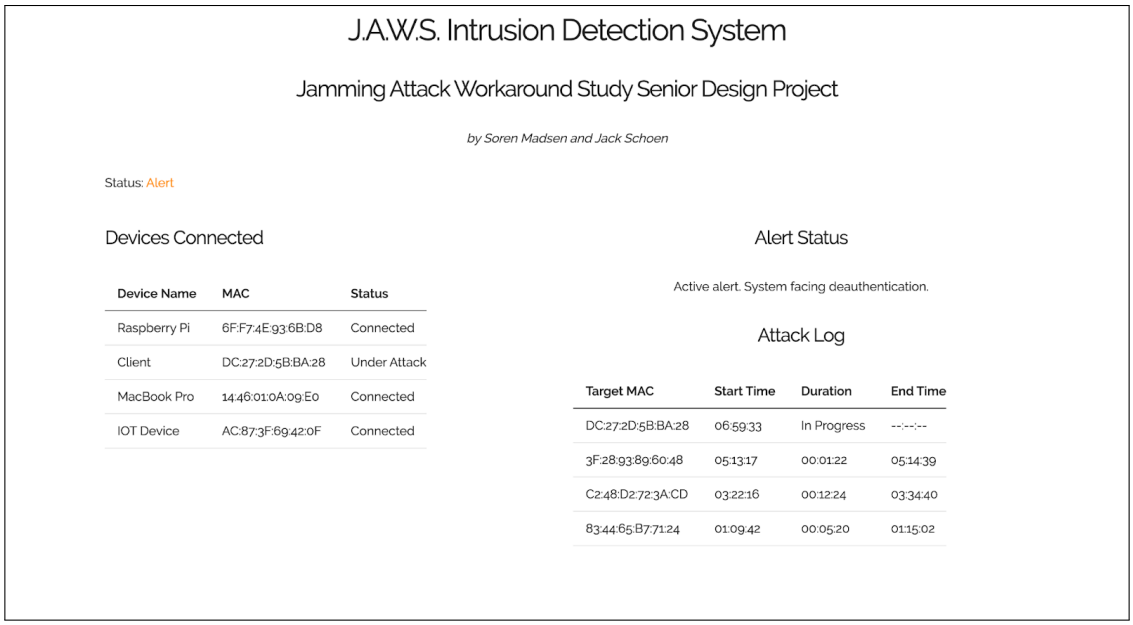

Figure 4.3: Flask Dashboard Sample with Alert Status

Because this sniffer is physically connected to the network, there is a very low chance of an attack that can bring down network monitoring. We ensure that the sniffer node is exclusively vulnerable to a physical attack. In this manner, we maintain a log of all attacks such that no attack can go unnoticed unless there is a physical removal, disconnect, or power off of the device.

Chapter 5

Performance Evaluation

5.1 Testbed

In our experimental setup, we utilized the aforementioned four main components to build a testbed and run simulations on our attack detection system: a TP-Link AC 1750 wireless router, a victim node (Raspberry Pi 4), an attacker node (Raspberry Pi 4), and a sniffer node (Raspberry Pi 4). The wireless router was connected to other smart home devices such as: Amazon's Echo Dot, a Google Home, a Google Chromecast, and smart LEDs. In this way, we could verify that our system was fit to perform in an actual smart home environment.

5.2 Parameters

Based on our network and algorithm design, we decided to verify the accuracy of our system for varying packet reception rates. Our proposed system performs packet sniffing and traffic analysis aimed at the detection of a flood of deauthentication packets. In this manner, we were mandated to question exactly how the system would perform if not all packets of the flood were received by the sniffer. Next, we were curious to see how we could manipulate this rate without the use of lab equipment (due to COVID-19 restrictions).

Varying distances and RF shielding were deemed the parameters for further experimentation that can be seen in the figure below. We moved our sniffer node to 4 different distances and shielded it from RF radiation with a microwave oven in the last distance to create packet loss. The four distances tested were: 1m, 5m, 10m, and 15m. By adjusting these parameters, we could alter packet reception rate and test our system against a range of reception rates for a given distance. Our hypothesis for system performance was that, as packet reception rate dropped, our system's success rate of attack detection should as well.

5.2.1 Packet Reception Rate

Packet receptions rate is a key metric in understand in the behavior of a Layer 2 system such as ours. Because we test for a flood, it's imperative to examine the system's behavior when the sniffer does not receive all traffic in the network.

In WiFi Layer 2 communication, a retransmission protocol is implement such that it is utilized for packets that are not received correctly or at all. If a packet is malformed or does not pass the test of the Frame Check Sequence (FCS), a request will be sent for retransmission marked by a bit in the MAC header. This means that in a system with poor packet reception rate at the Layer 2 level, there can be nearly 100% packet reception in Layer 3 achieved by the use of a retransmission count that is typically set to 7 in most devices. Deauthentication attacks are all based in Layer 2, therefore our packet reception rate has to be tested without the retransmissions. It is not ideal for a security system to request a second sending of specific frames from devices it manages. Rather, the system should be designed to work around this obstacle. In the figure below, we present the essential design for testing packet reception rate.

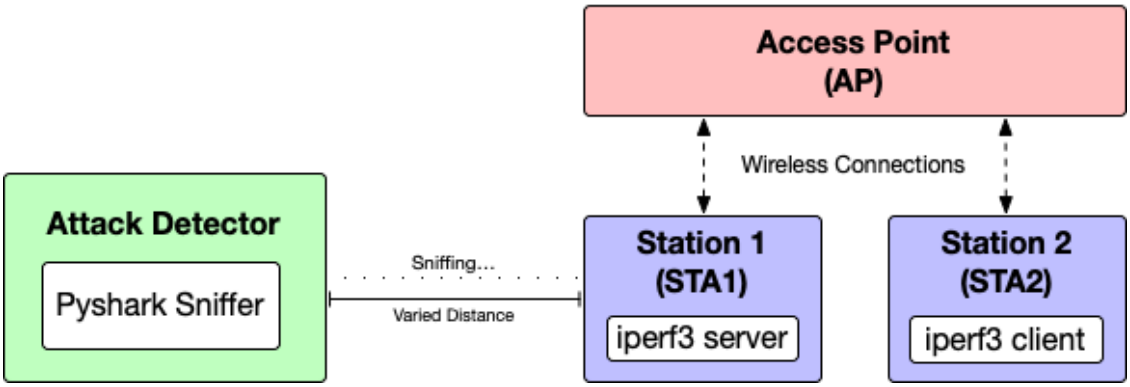

Figure 5.1: Packet Reception Rate Sampling Setup

Utilizing a software called iperf3, one station simulating the position of the sniff was run as a server, while the other station simulating the attack was run as a client. When running the software as a client, the attacker node would send 900 packets at a throughput of 1Mbps to the sniffer node running the server. While each of these trials was done, we wrote a python script utilizing pyshark to capture all Layer 2 transmissions to the server that did not include a value of 1 in the retransmission bit that can be seen in context (labelled as ”Retry”) in Figure 4.2 showing the header information. We would then compare the packet count marked in our Python code with 900 to derive our packet reception rate. We performed 30 trials of this for each of the four varying setups and input this information into a spreadsheet.

5.2.2 Success Rate of Attack Detection

The most important evaluation parameter in the determination of the success of our system is the attack detection success rate. This is the clear cut way to verify proper functionality and future viability as a security solution. In our setup, we followed the architecture shown previously in Figure 2.1 to perform such tests. A test of detection success rate is performed by running our system and launching a series of 30 attacks per test setup. For each attack that was flagged, we marked success. After the series of 30 trials, we derived our attack success rate for each experiment.

5.3 Empirical Results

In the figure below, we present our packet reception rate visualized by the box plots displaying the upper limit, the upper quartile, the mean (green triangle), the median (orange line), the lower quartile, and the lower limit. The X indicates an outlier value for a trial. The attack detection rate is shown by the blue squares. From this plot, it is clear to see that our system performed well across the board. However, because our hypothesis was wrong, we were inspired to ask why.

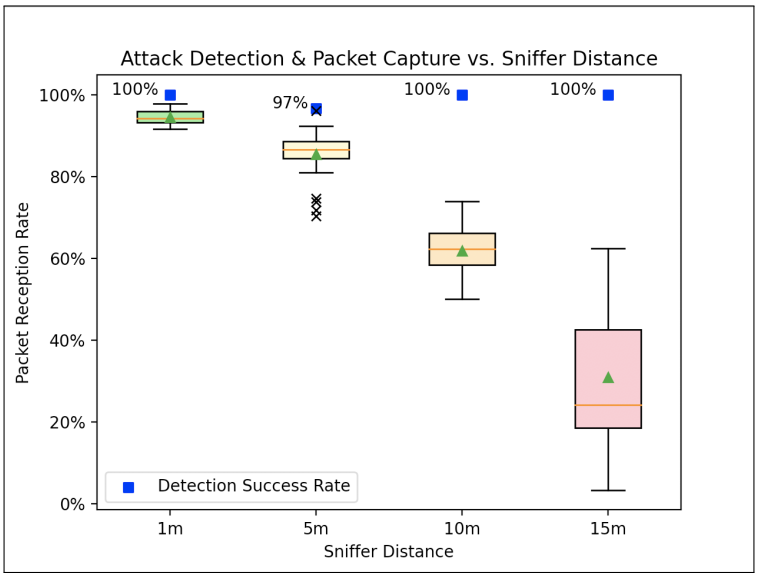

Figure 5.2: Plot of Packet Reception Rate and Detection Success Rate for Four Trials

From further research into these results we discovered a term called the modulation rate, or PHY rate, that is used to control the throughput of frames. Typical data frames can be transmitted at about 70 Mbps, which makes sense for single device communication. However, a network can be made up of both newer and older devices that support a varying array of throughput values. In accordance with such a concept, management frames are transmitted at 1 Mbps. Because the throughput is low, it less susceptible to packet loss. Therefore, our results exhibit behavior attributed to this implemented feature of the 802.11 protocol.

Chapter 6

Future Work

We believe that this work has opened the door to a variety of different routes of possible research to continue with great pertinence to the IoT industry. It is our hope that some of these areas will be explored by the next generations of engineers from Santa Clara University.

6.1 Battery Power

In many IoT systems, it is common for devices like sensors and cameras to be powered off of removable or rechargeable batteries. This reduces the need for cabling and improves the mobility of the devices such that they can be easy moved around depending on the necessity of the system. Our current design would support has limited capability in the realm of battery powered functionality. With a 100 Watt-hour battery, the system can run for approximately 40 hours given standard Raspberry Pi 4 power consumption metrics. This could likely be greatly improved by adjustments to the current hardware setup. The use of a Raspberry Pi Zero W could be an alternative to test in this regard.

6.2 Hardware Variations

While the plug-and-play nature is a perk when implementing our system into a network that has already been set up, this feature could also be useful when integrated into a router. This would reduce the need for extra hardware, extra cables, and excess spending on an additional device. By integrating our system into a router, security could be available to those seeking a low-cost alternative to protect their smart home devices rather than more expensive security-integrated routers.

An additional variation we propose for this system is to adjust the receiving antenna of the wireless network interface. By doing this, the sniffer would need to process more traffic, but a larger antenna could provide the device with extended attack monitoring range. We believe that there would need to be a careful balance between range and processing power, as real-time detection ability diminishes as processor load increases. The detection system should not be delaying the processing of packets. Perhaps, the combination of changing the hardware and adjusting the

antenna could result in a more ideal analysis of the traffic.

6.3 Station Voting

In larger systems where multiple sniffers may be required to encompass all traffic, we believe that a network of sniffers connected in a mesh fashion could vote on deauthentication packets as an additional security measure. For example, if an attacker were to try to deauthenticate a device on outside range of a sniffer, then another sniffer node could monitor this traffic and propose an attack warning to the network. If the device monitoring the attacked node does not also see this deauthentication frame, the alert can be raised. This provides additional protection by utilizing other parts of the network that may not necessarily be focused on a specific victim device.

6.4 Additional Attack Vectors

With WPA and WPA2 technologies, there are a number of other threats that still plague WiFi connected devices. For example, an adversary may utilize a deauthentication attack to force a device to connect to a fake access point, known as a man-in-the-middle attack, to steal information and any data sent or received over the internet. In addition, KRACKs (Key Reinstallation AttaCKs) have illuminated cryptographic vulnerabilities in WiFi technology. By considering these vectors, our project could be improved to mitigate and detection addition forms of malicious behavior.

As the IoT industry begins to integrate WPA3 technology into devices, there will be additional attack vectors to consider for the monitoring of these networks. In [12], Vanhoef and Ronen propose an attack on the WPA3 dragonfly handshake protocol utilizing timing flaws and information leakage that takes place during the initiation of the protected stream. Because this new technology also has security flaws, it will be imperative for future iterations of this system to address them as well.

Chapter 7

Societal Issues Addressed

7.1 Ethical

The central ethical issue that our project is centered around is the fundamental right to privacy. With vulnerabilities still running rampant in WiFi systems, those without the time or funds to upgrade their network are left by the wayside. We believe that the provisions of safety and security for IoT devices that have become paramount to improving the quality of living are fundamental for all. As such, our design revolves around entirely open-sourced hardware and software in it's implementation.

7.2 Social

As our daily routines become more integrated with technology, it is imperative that social values of privacy and safety are maintained. Through our design, we hope to bolster confidence in the use of connected devices so everyone can feel safe and private in their own homes and businesses.

7.3 Political

Because user privacy has been increasingly encroached upon, our project aims at giving power back to the people in the face of government programs that allow for violation of privacy. With the increasing integration of technology into smart homes, smart cities, and more, it is crucial that citizen privacy is protected by our government as well. This project can provide a springboard for future designs securing public infrastructure projects as well.

7.4 Economic

Due to low-cost hardware, our solution provides an alternative and cheaper security system than what the market currently offers to smart home users. In addition, because we address legacy products, businesses running large networks of legacy products need not worry as this device will handle both WPA and WPA2 standards. This reduces

the need to replace products already implemented.

7.5 Health and Safety

As systems become more reliant on wirelessly connected devices, the safety of home users and businesses alike become vulnerable when these devices are not protected. In the case of smart security monitoring, cameras and sensors that can be taken down via a deauthentication attack are of extreme concern in relation safety. By maintaining the integrity of these systems, our project will help to ensure the future safety of homeowners and businesses.

7.6 Manufacturability

Our design, as mentioned previously, is low cost and open-source which makes it easily reproducible and manufacturable. Since the heart of the project is based on the software, hardware adjustments can be made such that the system can be reproduced to better fit the network. All of this considered, we believe that this project can be easily produced to bolster IoT security everywhere.

7.7 Sustainability

The proposed system supports sustainability by providing security for legacy devices. With new technology like WPA3, new hardware must be implemented if the provided security enhancements are desired. This means overhauling existing networks and replacing WPA or WPA2 devices with new ones. Not only is this costly, but it creates an excessive amount of electronic waste, especially when considering that WPA3 still has many flaws that have yet to be secured. The upgrade is simply not worth the cost and waste. By using our system, one can ensure that their devices are still protected without creating more waste than necessary.

7.8 Environmental Impact

The fundamental design of the proposed system revolves around the basic requirements for IoT: low cost and low power. Raspberry Pi's satisfy these requirements and provide us a solid baseline for protection with a small carbon footprint. In addition, the implications of the system, as mentioned previously, mean less electronic waste in landfills by eliminating the need to replace systems with new devices.

7.9 Usability

As mentioned in our project overview, our proposed solution is easy to install, extremely adjustable, and provides quick access for simple network management via a simple user-friendly interface. Through this, we have made smart

home and IoT security easy for the everyday person. No networking experience or coding knowledge is required to use this solution.

7.10 Lifelong Learning

Our design process for this project has provided us with valuable skills of research, knowledge of a booming industry, and practical application of theoretical ideas. In so doing, we are motivated now more than ever to further explore the Internet of Things and cyber security.

7.11 Compassion

In our work, we have provided users with a means to protect their privacy and ensure the safety of connected devices in a low cost and open-source solution available for everyone. We hope that with our design project will inspire others in pursuit of similar goals. It is crucial that these future pursuits also revolve around low cost designs to ensure that wealth and class need not be a factor when securing user privacy.

Chapter 8

Conclusion

In this chapter, we summarize our project design, acquired learning outcomes, and overview the advantages and disadvantages of our proposed solution.

8.1 Summary

Our project proposes a security solution for smart home and industrial IoT networks that provides an easy graphical user interface, quick installation, and a low cost design that monitors and protects both new and legacy devices. In addition, we have evaluated the performance of this system in a smart home environment and found that it to be very successful. We have concluded that this solution, while it still has a long ways to go in terms of greater development, is a suitable candidate for affordable smart home protection. In addition, the future work that can come from this design will benefit the IoT industry in a variety of ways.

8.2 Learning Outcomes

Along the journey of this project, we deeply learned about wireless network functionality and the ins and outs of the 802.11 protocol. In tandem, we honed our research skills, programming in Python, and network configuration skills. From our work, we feel confident in our ability to configure and manage WiFi networks for smart home use. In addition to our improved technical skills, we have attained a better understanding of project management, time management, and public speaking on technical topics.

8.3 Advantages

As mentioned previously, our design proposal has several key advantages. Firstly, the system monitors network traffic for deauthentication attacks in real-time. The efficient design of code and utilization of the BPF allow us to maximize the network traffic that we can analyze. Secondly, our evaluation of the system performance indicates that our solution

is extremely viable for smart home environments. Lastly, the possibilities of future work mean that our work can provide a solid basis for a better design that can account for more attack vectors.

8.4 Disadvantages

While we developed a solid solution for deauthentication attack detection, we have several disadvantages One, we do not perform mitigation of a deauthentication attack. Because it is nearly impossible to detect an attack until it is already happening, mitigation techniques are limited in terms of feasible implementation on a Raspberry Pi. Second, our research into a workaround for recovery from a deauthentication attack did not lead us to a methodology to implement. We found that the 802.11 protocol is based on states, and to recover a connection means to have the attacked device attempt to connect back to the access point. Thus, our software would have to be running from the victim rather than a sniffer node. Lastly, Flask is not the best library for running a web server. We used it because it is lightweight, but Flask could be exchanged for a more functional software in the future if the system is run on a device with more processing capability.

Bibliography

[1] Ananay Arora. "Preventing wireless deauthentication attacks over 802.11 Networks." In: (2018). URL: https://login.libproxy.scu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.1901.07301&site=eds-live.

[2] Baber Aslam, M Hasan Islam, and Shoab A. Khan. "802.11 Disassociation DoS Attack and Its Solutions: A Survey". In: *2006 Proceedings of the First Mobile Computing and Wireless Communication International Conference*. 2006, pp. 221–226. DOI: 10.1109/MCWC.2006.4375225.

[3] John Bellardo and Stefan Savage. "802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions". In: *12th USENIX Security Symposium (USENIX Security 03)*. Washington, D.C.: USENIX Association, Aug. 2003. URL: https://www.usenix.org/conference/12th-usenix-security-symposium/80211-denial-service-attacks-real-vulnerabilities-and.

[4] Edgar D Cardenas. *MAC Spoofing–An Introduction*. Aug. 2003.

[5] D. Dasgupta et al. "MMDS: Multilevel Monitoring and Detection System". In: *In the proceedings of the 15 th Annual Computer Security Incident Handling Conference*, pp. 22–27.

[6] Fanglu Guo and Tzi-cker Chiueh. "Sequence Number-Based MAC Address Spoof Detection". In: *in Proceedings of 8th International Symposium on Recent Advances in Intrusion Detection (RAID*. Springer, 2005.

[7] *IoT Has Quietly and Quickly Changed Our Lives*. Feb. 2019.

[8] Knud L. Leuth. "State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time". In: Mar. 2021.

[9] Aruba Networks. "Detecting deauthentication and disassociation attack in wireless local area networks." In: (2019). URL: https://login.libproxy.scu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edspgr&AN=edspgr.10243974&site=eds-live.

[10] T.D. Nguyen et al. "A Lightweight Solution for Defending Against Deauthentication/Disassociation Attacks on 802.11 Networks." In: *2008 Proceedings of 17th International Conference on Computer Communications and Networks, Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on* (2008), pp. 1–6. ISSN: 978-1-4244-2389-7.

[11] Haitham A Noman et al. *A Lightweight Scheme to Mitigate Deauthentication and Disassociation DoS Attacks in Wireless 802.11 Networks*. Feb. 2016. URL: https://ijens.org/Vol_16_I_01/161901-5858-IJVIPNS-IJENS.pdf.

[12] Mathy Vanhoef and Eyal Ronen. "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd". In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020, pp. 517–533. DOI: 10.1109/SP40000.2020.00031.

[13] Joshua Wright. "Detecting Wireless LAN MAC Address Spoofing". In: IEEE, 2003.

Appendix A

Installation Guide

A.1 Required Libraries

We utilized several key libraries and operating systems for the design, testing, and analysis of our system. The main operating system for our attacker, sniffer, and victim nodes was the 2020.4 distribution of Kali Linux with the nexmon patch. This operating system was chosen because it provided us with the ability to inject packets as well as sniff packets at the Layer 2 level for deep analysis of traffic. The nexmon patch allows the operating system to access the netlink module of the kernel and virtualizes the network interface functionality to bypass restrictions of the Raspberry Pi hardware.

A.2 Sniffer Setup

We mounted our SD card of the Raspberry Pi for with the 2020.4 Kali distribution, once again, with the nexmon patch. To do this, the Raspberry Pi Imager software (distributed online with the board) was used to mount the operating system image to the SD card.

Pre-installed on this operating system is Python 3.8, which is the version of Python in which we developed our sniffer. In addition to Python, `pip` was used to install the `pyshark` library. Before running our sniffer code (see section below), we had to execute two key setup commands:

1. `airmon-ng check kill`
2. `airmon-ng start wlan0`

These two commands allow us to put the network interface card in Monitor mode to sniff the traffic at the MAC layer. To ensure that these commands work properly, verify that the distribution of Kali Linux has the nexmon patch integrated.

A.3 Evaluation Recreation

A.3.1 Packet Reception Rate Measurements

To test the packet reception rate of the system, we installed iperf3 on the attacker node, and the victim node. Next, we ran iperf3 on the victim as a server using:

```
iperf3 -s
```

Next, we ran iperf3 on the attacker node using a similar command:

`iperf3 -c <host>:<port> -n 900` where the host and port are automatically assigned by the DHCP server on

the router and the iperf3 software, respectively.

Using this command, we send 900 packets to the victim node, simulating the deauthentication frames sent. In tandem with this command, we have the sniffer running code to parse our the packets sent by the attacker that are not retransmissions.

A.3.2 Attacker Setup

We again mounted our SD card of the Raspberry Pi for with the 2020.4 Kali distribution, once again, with the nexmon patch. To do this, the Raspberry Pi Imager software (distributed online with the board) was used to mount the operating system image to the SD card. This distribution is used primarily because it comes pre-installed with packet injection tools. In particular, we made use of the `aircrack-ng` suite. Before executing an attack, the NIC must be placed in Monitor mode using the following commands:

1. `airmon-ng check kill`
2. `airmon-ng start wlan0`

Following this, we can inject packets and perform a deauthentication attack using the command:

```
aireplay-ng --deauth 10 -c <victim MAC> -a <Access Point MAC> wlan0
```

We found that this attack could sometimes cause the virtual NIC provided by `airmon-ng` to fail and stop injecting packets. The quickest solution to this problem is to reboot the device and execute the setup commands again. This will allow for further attacks to be placed.

A.4 Code and Supplemental Resources

This is our code for our sniffing system, and it can be found online on GitHub as well.

Sniffer.py:

```
import pyshark
import threading
import time
from flask import Flask, render_template, request, redirect, url_for
import sys

global status, stat, alert, statAlert, clr, log

headings = (''' "Name" ''' "MAC", "Status")
data = [
        ''' Typical Format
    ("Raspberry Pi", "6F:F7:4E:93:6B:D8", "Connected"),
    ("Client", "DC:27:2D:5B:BA:28", "Under Attack"),
    ("MacBook Pro", "14:46:01:0A:09:E0", "Connected"),
    ("IOT Device", "AC:87:3F:69:42:0F", "Connected"), '''
]

bad_status = ["Bad", "Active alert. System facing deauthentication.", "DarkOrange"]
good_status = ["Good", "No alerts active. System protected.", "green"]
status = "good"

logHead = ("Target MAC", "Start Time", "Duration", "End Time")
logData = [
```

```
        '''     Typical Format
    ("DC:27:2D:5B:BA:28", "06:59:33", "In Progress", "--:--:--"), # 00:31:27 07:31:00
    ("3F:28:93:89:60:48", "05:13:17", "00:01:22", "05:14:39"),
    ("C2:48:D2:72:3A:CD", "03:22:16", "00:12:24", "03:34:40"),
    ("83:44:65:B7:71:24", "01:09:42", "00:05:20", "01:15:02"),'''
]

@app.route("/", methods=['GET', 'POST'])
def home():
    global status, stat, alert, statAlert, clr, log

    if status in "good":
        #statAlert = "<span style=\"color:green;\">Good</span>"
        us = "green;\">Good"
    elif status in "bad":
        #statAlert = "<span style=\"color:DarkOrange;\">Alert</span>"
        us = "DarkOrange;\">Alert"
    elif status in "ugly":
        us = "red;\">Warning"
    stat = "<div class=\"container\"><p>Status: <span style=\"color:"
    stat = stat + us + "</span></p></div>"
    page = render_template("index1.html", headings = headings, data = data)
    page = page + stat + render_template("index2.html", headings = headings, data = data,
        status=status, alert=alert, statAlert=statAlert, clr=clr,
        logHead=logHead, logData=logData)

    return page

@app.route("/cycle/", methods=['POST'])
def cycle():
    global status, stat, alert, statAlert, clr, log
    if status in "good":
        clr = "green"
        #statAlert = "<span style='color:green;'>Good</span>"
        statAlert = "Good"
        status = "bad"
        alert = "Active alert. System facing deauthentication."
    else:
        clr  = "DarkOrange"
        #statAlert = "<span style='color:DarkOrange;'>Alert</span>"
        statAlert = "Alert"
        status = "good"
        alert = "No alerts active. System protected."
    return redirect(url_for("home"))

@app.route("/status/<status>")
def user(floop):
    return "Status is {floop}"


# Hard coded for our system
ap = "0a:11:96:8c:11:29"
```

```
deauth_table = {"" : ""}
        # address: adddr
        # pkt_queue: packets
        # warning: true/false

connections = []

def parseMACaddrs(pkt):
        # Adds transmitting/receiving device to list of connections if not present
        addr = ""
        if pkt.wlan.sa != ap:
                addr = pkt.wlan.sa
        else:
                addr = pkt.wlan.da

        if addr not in connections:
                connections.append(addr)
                data.append((addr, "Connected"))
                print("New connection discovered: ", addr)
                cycle()
        return addr

def removeConnection(dev):
        print("Removing connection: ", dev)
        connections.remove(dev)
        data.remove((dev, "Connected"))
        data.append(dev, "Disconnected")

def monitorAttackProgress(dev):
        attack = True
        ts = deauth_table[dev]["first_timestamp"]
        while attack:
                last_ts = float(deauth_table[dev]["packet_queue"][-1].sniff_timestamp)
                if last_ts - ts < 3 and last_ts - ts > 0:
                        print("Attack in progress on", dev)
                else:
                        print("Attack ended on ", dev)
                        start = deauth_table[dev]["first_timestamp"]
                        end = last_ts
                        logData.append((dev,start,end,end-start))
                        attack = False
                        break
                time.sleep(2)
                ts = last_ts
        deauth_table.pop(dev)

def checkDeAuth(dev, pkt):
        #print("Testing deauth")
        if dev not in deauth_table.keys():
                deauth_table[dev] = {"first_timestamp":float(pkt.sniff_timestamp), "packet
        else:
                # Add to table
                deauth_table[dev]["packet_queue"].append(pkt)
                if (float(pkt.sniff_timestamp) - float(deauth_table[dev]["first_timestamp"
```

```
                        # Timeout if the next
                        deauth_table.pop(dev, None)
                if len(deauth_table[dev]["packet_queue"]) > 4 and deauth_table[dev]["warnir
                        deauth_table[dev]["warning"] = True
                        print("Deauthentication attack detected on: ", dev)
                        mon = threading.Thread(target=monitorAttackProgress, args=(dev,))
                        mon.start()


def startIDS():
        filter = "ether host 0a:11:96:8c:11:29"
        cap = pyshark.LiveCapture(interface="wlan0mon", bpf_filter=filter)
        dev = ""
        for pkt in cap.sniff_continuously():
                #print(pkt.wlan.fc[3:7])
                dev = parseMACaddrs(pkt)
                if pkt.wlan.fc[3:5] == '00':
                        if pkt.wlan.fc[5:7] == '0a':
                                # Disassociation is OK for now
                                removeConnection(dev)
                        if pkt.wlan.fc[5:7] == '0c':
                                checkDeAuth(dev, pkt)

if __name__ == '__main__':
        if "--no-flask" in sys.argv:
                noFlask = True
        else:
                noFlask = False

        ids_thread = threading.Thread(target=startIDS)
        ids_thread.start()
        if not noFlask:
                app.run(host='127.0.0.1',port=5000,debug=True)
        ids_thread.join()
```

This code is used to evaluate the packet reception rate of the sniffer at various locations.

Capture.py:

```
import pyshark
filter = "ether host dc:a6:32:c9:e5:b9" # attacker MAC address
capture = pyshark.LiveCapture(interface="wlan0mon", bpf_filter=filter)
count = 0
for pkt in capture.sniff_continuously():
        if pkt.wlan.fc_retry != '1' and pkt.wlan.fc_type_subtype == '40':
                count += 1
                print(count)
                print("Retry: ", pkt.wlan.fc_retry)
                print("Type_Subtype: ", pkt.wlan.fc_type_subtype)
```

Additional resources can be found on GitHub at the following link: https://github.com/sorenjmadsen/JAWS