IoT MFA/OTA Updates

Jared L Hinze, Jordan M Jorgensen, and Reggie D Van Driel, DSU INSuRE, DSU

Abstract—This paper describes our research for techniques which can be used to harden the security of Internet of Things (IoT) devices through multifactor authentication (MFA). Our goal is to develop a unique and transparent authentication method between two devices that utilizes both frequency hopping and variable packet timing to authenticate to and persist on a network. The progress made consisted of establishing communication between devices, solving basic error correction of packets, creation of a packet structure, as well as the design of the overarching structure of the program and the logic to respond correctly to any packet.

Index Terms—IoT, MFA, OTA, RFcat, YardStick One, Raspberry Pi

Introduction

The goal of this research is to develop an almost fully transparent form of multifactor authentication for use with IoT devices allowing for greater security.

A multifactor authentication system would allow current compatible devices (with possible add-on) and future IoT devices to be more secure and easier to secure by requiring a short-range radio transmission. The transmission would be required to edit the device’s settings to prevent outside connections from changing important settings. Even if unauthorized access is gained via the internet, there would be no way change device settings without the short-range radio communication. This will mean that only the authorized users within a certain proximity of the device will have administrative control over the device settings.

Currently, multifactor authentication is something which requires more work than people want to put into securing devices. Therefore, there needs to be a solution that is easily implemented into future devices allowing a more transparent form of multifactor authentication. There needs to be a reasonable and inexpensive addition to current devices and future devices which, given some testing and research, should also be completely transparent to users.

Literature Review

A. Background

IoT devices over the last few years have become more and more popular. With predictions of billions more devices to be produced over the next decade. Most appliances currently have a computer of some kind inside. Although, those computers are smaller. In the past these smaller computers never had any form of connection to the internet. These computers just handled the inner workings of the device itself.

With technical advances making computers smaller and more powerful it has become the norm to include full-fledged computers into everyday appliances. This brought a great deal of benefits to manufacturers. Instead of writing a whole new operating system for their appliance they can just write software allowing interaction to be handled by the drivers which are simpler to write. This also opened the door for many more features to be included into the appliances since the appliances themselves could now connect to the internet. These features may also provide the user with notifications and the ability to control the appliance remotely.

Appliances now days often utilize little computers capable of running various applications and services for a user. These computers are often also now vulnerable to attack by outside parties. The explosion of IoT devices has increased the attack surface for malicious individuals exponentially.

Currently, IoT device developers are more in a rush to manufacture new devices instead of being concerned about how to make sure those devices are secured when released. This lack of security focus is a huge problem with today’s IoT devices. Hackers have begun creating enormous botnets using these poorly secured appliances. An example of a botnet would be the Linux.Aidra botnet from 2012 which infected thousands of devices from home routers, internet connected televisions, DVRs, security cameras and more. Using these botnets attackers can perform Denial of Service (DoS) attacks that can bring down even the most robust infrastructures. The most famous botnet being the Mirai botnet in 2016 set records for the greatest volume of internet traffic being more than 1Tbps of transfer rate. The worst part about these botnets is that they are not some sophisticated hack which took years to develop. Almost all of them are based on the premise that IoT devices have inherently terrible security. Usually the attacks exploit preconfigurations which cannot be changed by the user like factory default passwords, or passwords which are hardcoded into read-only memory.

B. Motivation

Our project hopes to provide a proof of concept for a multifactor authentication system which requires minimal interaction from the user. This is important because currently one of the big hurdles for multifactor authentication is how much longer it takes compared to just remembering a username and password. Time really isn't that much of a difference in practice, but the general public will almost always lean towards the easier and more convenient option.

The system has to be able to prevent an outside malicious individual from disrupting the workings of the device while only expecting a minimal amount of interaction from the user.

B. Justification for Our Approach

The concept of our solution to this problem is take examples from elsewhere in technology and adapt them to fit the new needs of IoT devices. An example in technology that we are leveraging is key fobs for cars. In our approach your phone becomes the key, and the IoT device is the car. To make this work we identified features which would need to be added or improved upon. To unlock a car the key fob emits a weak signal which can only be heard by the car within a small distance. The same distance requirement is something we want to keep. Although car key fobs only have one-way communication with no way to prove that the signal they are hearing is in fact the correct set of keys. There have been tools made which can fake a key fob signal and unlock other people's cars, and we want to try and eliminate that possibility in our design.

There are many ways to solve this problem. Some of these ways may be viewed as simple while others may be viewed as much more difficult to accomplish. As an example – the best way to resolve the security focus dilemma would be to change the policies and views of manufacturing companies so that all the devices would be secure directly from the start. The problem would be getting all IoT companies on board. Also, there is the possibility of just adding the regular form of multifactor authentication. This would make securing IoT relatively simple, but it would push all of the effort onto the consumer. While this approach could be great for developers – consumers almost always go with the more convenient options. Typically, those options are viewed as less secure.

Our solution only increases the amount of work the user has to do during the initial setup of our device. Once both sides of the connection are synchronized that is the end of active user involvement. They don't even need the device connected to their phone if they only want to use the device and not modify it at all. The plan is to allow normal functioning of the device when not connected to the IoT device, but to prevent any modification of administrative or functional settings during that time. This makes our solution simple and easy for the consumer while still preventing the device from being compromised by a malicious third party.

Methods and Procedures

A. Initial Research

Wireless technology has many legal considerations to be taken into account. Therefore, observation of the laws regarding wireless transmission would be required. Before anything else was worked on, we needed to find out every legal element pertaining to our situation.

In the United States, wireless frequencies are sectioned out and reserved for specific purposes. Some of these purposes are for private companies to use. For example: most cellular companies own a section of the bandwidth to use for their service. For our proof of concept, we settled on using the Family Radio Service (FRS) channels. The FRS channels are reserved for private, two-way, voice and data communications [5]. The FRS doesn't require a license to transmit when below 2 watts of power. For our tests we used frequencies between 462.5625MHz and 462.7250MHz.

We researched radios which would allow us to modify exactly what was being sent. There are many amateur radios that are called "Software Defined Radios". These use software to determine how to send and receive rather than having all the logic built into the hardware. We settled on a USB radio called the YardStick One. The reason we chose it was mainly the price. It provided a few advantages since it was entirely controlled using a python library written by the developers. This made programming for it relatively simple compared to other software defined radios. At this point it was also easy to agree that the best programming language of choice for development using this device would be Python.

Finally, we decided to use Raspberry Pi's as the platform for developing and testing our solution. The Raspberry Pi were very similar to what our intended final use case was expected to be since this is supposed to work for IoT devices and be able to function on very resource constrained devices. The Raspberry Pi's can be viewed as very resource constrained relative to modern computers and mobile devices.

B. Initial Plan

Once we had made decisions on all of the applicable hardware, software, and the intended frequency range we would be testing on we could began discussing in more detail the functional and nonfunctional requirements of our solution.

One of our first ideas were based around the key fobs currently used for locking and unlocking cars as they enforced the requirement that the owner of the car must be within a certain distance of the car for it to allow access to anyone. Translating that to an IoT device in a house would require the user to be inside the house to access any sensitive information on the device. A distance requirement alone wouldn't be enough to make IoT more secure than it already is. Instead of a one-way communication like key fobs we wanted to make our solution a two-way conversation consequently making it much harder to impersonate the user. We now had to make impersonation as difficult as possible for a malicious third party to listen to the conversation and insert themselves into the middle of it all. To prevent this Man-in-the-Middle attack we wanted to include Frequency Hopping as well as Variable Packet Timing. Frequency Hopping is something which is used all over to prevent third parties from listening to communications. The concept of Variable Packet Timing is based around having the packets sent between the devices are on a synchronized timetable. This means that packets can only be received at specific times, and that packets are received outside of the specified time would be considered either an error in the communication or there is a third party attempting to interfere with the system. In either case we plan to securely restart the communication between the devices using known information now stored within both devices.

C. Procedure

1. Initial Communication

Before we could begin work on developing the solution itself we needed to become familiar with the hardware we would be working with as well as gaining an understanding of how to wirelessly transmit data between our two devices. Using the RFcat library for the YardStick One made sending data very simple. To receive data the second radio just needs to tune to the same frequency and be actively listening for messages.

2. Two-Way Communication

Wireless communication is inherently susceptible to interference from other wireless devices. This means that none of the data received can be guaranteed to be correct. While parsing the received message we had to make sure to double check the data to be positive that the data was received correctly. This just means that our code needs to gracefully handle errors and respond correctly after receiving an incorrect packet. To do this we had to solve a few problems with parsing the packet by adding a header and a tail to each message. We also had to solve a major problem with packet error correction for when we received bad packets.

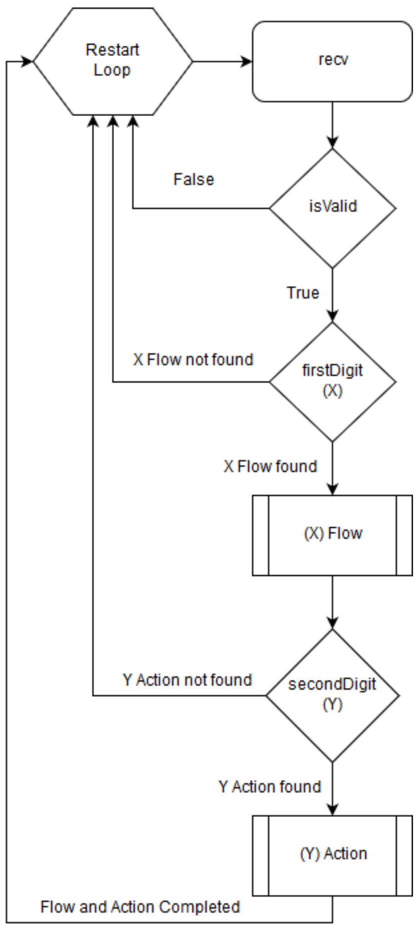
3. Program Structure

Initially, the structure of our program was based entirely around being a proof of concept. It did what was needed for us to better understand the concepts and requirements of what we wanted to accomplish later on in the project. This meant that our initial program was very much a patchwork of solutions to smaller problems, and adding new features was starting to become problematic. We specifically got stuck on the addition and handling of sequence numbers when sending segmented chunks of data. With the original program we would have spent a great deal of time including new functionality and requirements just to allow us to reconstruct a segmented packet in the correct order. That incident is what pushed us to begin our rewrite using all of the knowledge we had acquired thus far.

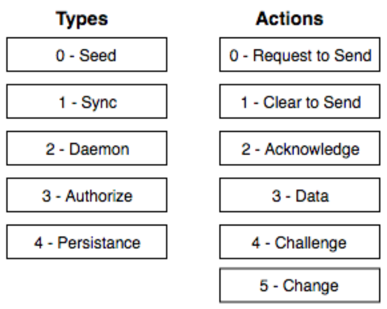
4. Final Program Logic

The final structure we settled on uses the initial characters of the message to determine which logical flow that specific packet should follow. Then once the packet type and action type have been determined the response can be formulated based on the previously sent data.

The packets sent back and forth have 5 fields, Type, Number, Time, Data, and CRC value. The Type (currently two digits) is used in the logical flow pictured on the next page. The Number, Time, Data are used based on where the flow ends up inside of the program.



First a packet is received which then is parsed and validated to have been received correctly. This is determined using the CRC while parsing.



Types	Actions
0 - Seed	0 - Request to Send
1 - Sync	1 - Clear to Send
2 - Daemon	2 - Acknowledge
3 - Authorize	3 - Data
4 - Persistance	4 - Challenge
	5 - Change

Above is a list of all the packet types and all of the actions for each type. We use the first digit to call the correct function to handle the corresponding packet type. Then the second digit is checked and used to call the corresponding action which determines how to respond.

If we had a Type field of '00.' the program would determine that the first digit is a '0' and it would make a call to the "SeedFlow" function. Then the "SeedFlow" function would check the second digit to determine the action. Here the second digit is '0' so we have received a "Seed" packet, with the action of "Request to Send". In this case the program, now inside the "Request to Send" function, would respond with a packet containing '01'. The first digit stays the same as we have not received an action type of '5' indicating to change packet types. The second digit becomes a '1' indicating "Clear to Send" which is the correct response following a "Request to Send". After sending the response the program would wait to receive the next packet, which should be a packet type of '0' or "Seed", and action of '3' indicating "Data".

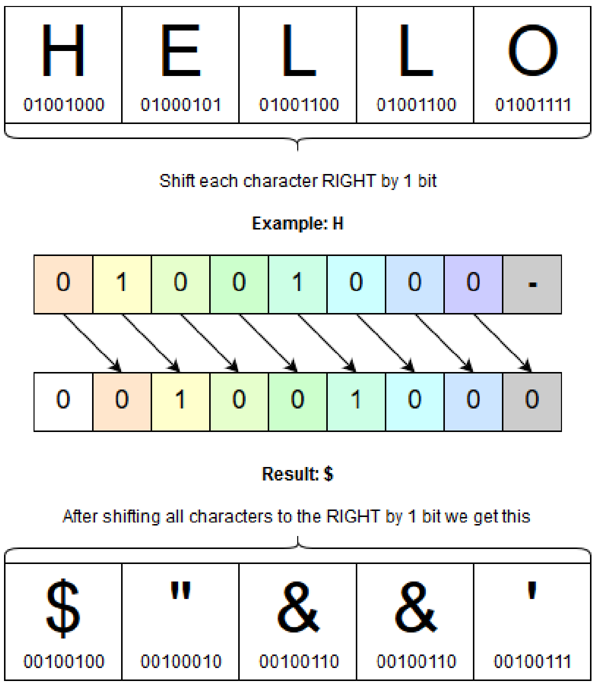
Now if anything goes wrong along the way such as the Type value not being recognized then there is an alternate flow which is used to handle issues while receiving packets. We also have to be careful while implementing this error handling flow to make sure that we aren't sending old or sensitive data.

Findings

Our work was centered around trying to design various proof of concepts for securing IoT devices and identifying how feasible these designs and approaches would be. We've found that it is easier than expected to secure devices using secondary out of band communication methods. The required approach would be to utilize either a full-duplex antenna or a secondary half-duplex antenna inside of these IoT devices to communicate on our out of band frequencies to control administrative settings of these devices. The administrative settings should be only capable of being changed when the communication of these changes is done through the out of band frequency channels while following our control flow patterns.

Issues

One of the first apparent problems with transmitting anything over wireless signals are the errors which come with transmitting real signals through the atmosphere. These errors can be caused by anything from bad signal to interference from other equipment nearby. These errors are called "burst errors". While common these burst errors are usually rather small and only affect a few of the bytes each transmission. Error correction can generally be done to correct these small errors in transmission. To correct these errors we researched forward-error correction methods. The first method to be looked at was Reed-Solomon forward error correction. This method uses very complicated mathematics to generate a series of characters to place after the transmission data. When this method failed to produce the results we wanted we began looking more closely at the data that was being received. We began to notice patterns in the data being received which closely resembled the data we had intended to send but had been equally modified throughout the packet. We realized this to be a bit shift error and developed code to attempt to correct it by shifting the bits in the packet left until we had a string that matched our expected message. This increased our number of correctly received packets.



The second main issue that we had was how to properly structure our program to allow for a logical flow so that we can react to any type of packet we receive correctly. We decided to mimic the concept of recursive descent parsing. We read the packet information step by step and make decisions on how to proceed from the information in each step. This allows us to include proper error checking by having a proper “correct” flow and “alternate flows” for when we encounter unexpected packet values. This also helps allow future developers to read the code and add to it more simply as the flow is very sequential and well defined.

A smaller hardware problem exists with the Yardstick One itself. The cause is currently unknown whether it be incorrect usage or just a problem of the hardware itself. The issue causes the need for the radio to be powered off and powered back on – otherwise no more transmission or reception is possible with the device. The issue also prints “-110” onto the screen on its own line. We attempted to track down where this value was coming from, but we have not found any place in the code we’ve written for this project or in the RFcat library itself that has the value -110 anywhere in it. There is no consistent method of reproducing the error either as it can happen at any point in the initializing or running of the code.

Conclusion

Over the course of our research we have looked into the field of wireless communication and how we can leverage some of the useful features already in current technology, to make upcoming IoT devices more secure. With this research we have determined that our solution to the hardening of IoT devices is very much possible. More research into how the measures we would like to put into place, such as variable packet timing, would impact the overall difficulty of intercepting and impersonating devices on a network. We believe variable packet timing could be very useful for applications where connection is required for authentication instead of entirely for data transfer.

Future Plans

1. Finish translating our old code to the new more formal and well-defined structure.
2. We figured out how to do frequency hopping, but we still need to properly include this feature in the current project. This will add a huge layer of security to the device. By changing the transmission frequencies we further inhibit outside parties from watching the data being sent back and forth. This also makes it very difficult for outside parties to send transmissions to the devices without the proper information. That required information is a key that was shared during the initial setup of the devices.
3. We still need to implement variable packet timing. This means the scheduling of when packets are supposed to be received and when packets should be sent. When added to frequency hopping the number of possible combinations for frequency and timing becomes enormous. Which makes exploiting the system through careful observation extremely difficult for an outside third party.
4. Lastly, we intend to design and implement a replica of how the systems would be set up in a real home as a proof of concept as well as a system to observe for problems and ways to improve the final design. This will be very last if all the other future plans have been accomplished.

Resources

[1] Leonjza, “Reverse Engineering Static Key Remotes with GNUradio and Rfcat”, #!/slash/note, https://leonjza.github.io/blog/2016/10/02/reverse-engineering-static-key-remotes-with-gnuradio-and-rfcat/, Oct, 2016.

[2] Harrison, “PythonProgramming”, http://pythonprogramming.net, 2017.

[3] Andre Nohawk, “Hacking Fixed Key Remotes with (only) RFCat”, Rockwell, https://andrewmohawk.com/2015/08/31/hacking-fixed-key-remotes-with-only-rfcat/, August, 2015.

[4] atlas0fd00m, “atlas0fd00m/rfcat”, Github, https://github.com/atlas0fd00m/rfcat, August, 2016.

[5] FCC, “Family Radio Service”, https://www.fcc.gov/wireless/bureau-divisions/mobility-division/family-radio-service-frs, September, 2017.

[6] NJCCIC, “Adria Botnet”, https://www.cyber.nj.gov/threat-profiles/botnet-variants/aidra-botnet, November, 2016.

[7] Radware, “A Quick History of IoT Botnets, https://www.cyber.nj.gov/threat-profiles/botnet-variants/aidra-botnet, March, 2018.

Biographical Sketches

Jared Hinze - 4th year Cyber Operations and Computer Science major at Dakota State University. I am a programming enthusiast and have been writing programs and scripts since I was 8yrs old playing with an old Macintosh voice activation system. I specialize in programming, networking, math, science, and physics, and I have a fair electrical background consisting of everything from installing electrical wirings in houses to the circuitry of computers and arcade games. I currently have been hired to write a standalone application for the City of Madison that mediates the import and export process of their Payroll software and other applications they use to aggregate their payroll data. Also, I am currently a Cyber Operations and Computer Science tutor for DSU.

Jordan Jorgensen - 4th Year Cyber Operations major at Dakota State University. I have participated in the yearly ACM Programming Competition held in Lincoln, NE. I have worked as a Tech Assistant for Josh Stroschein, an upper level Cyber Operations Professor. I also am currently a Computer Science tutor for DSU. Been interested in how computers function on the lowest levels since I started and have a great interest in things like assembly and reverse engineering.

Reggie Van Driel - 2nd year Cyber Operations and Network Security Administration double major at Dakota State University. I am an officer for my second year in computer club and heavily involved in the other computer related club on campus such as the offensive and defensive security clubs and CybHER. I have also been a part of a few projects with friends as well. My interest in computers grew after I attended the first GenCyber camp. I have helped with the camps for the past two summers.