Security Analysis of the August Smart Lock

Megan Fuller, Madeline Jenkins, Katrine Tjølsen
{*fullerm, mhj, ktjolsen*} *@mit.edu*

Massachusetts Institute of Technology — 6.857

May 24, 2017

Abstract

The growing network of connected devices, often collectively referred to as the Internet of Things (IoT), is subject to a vast array of potential security vulnerabilities. The August Smart Lock is a household IoT device that lets users replace their house key with their smartphone. It is imperative that security vulnerabilities in such a device are identified and patched, and as such we performed a security review of the August Smart Lock. The contributions of this paper include a discussion of the August Smart Lock security policy, a description of how the smart lock system works, an account of a series of attacks, and the results of those attacks. None of our attacks have revealed serious vulnerabilities in the August Smart Lock.

1 Introduction

The August Smart Lock, a connected device that attaches to a deadbolt, offers convenience to its customers in the form of greater lock usability than the unmodified deadbolt. However, users may not have the technical savvy to understand and identify potential security vulnerabilities in such a product. Accordingly, it is important for the security community to subject these higher-risk connected devices to greater scrutiny, such that weak points can be discovered and patched before an adversary misuses them. For these reasons, we chose to do a security review of the August Smart Lock.

While traditional locks have vulnerabilities of their own via lock-picking, Internet-connected locks open the possibility of doing this remotely, potentially allowing hackers to break into many locks at once. A compromised lock could give an adversary both access to the house and information about when the residents are home. Although a typical breaking and entering perpetrator is unlikely to have the technical skills necessary to conduct attacks against the August Smart Lock, this information could be obtained by a remote hacker and subsequently sold to nefarious third parties in illicit marketplaces.

The August Smart Lock, shown in Figure 1, is a battery operated Bluetooth device that attaches to the inside of a standard deadbolt and allows a user to lock and unlock the door with their smartphone. The smart lock also tracks which users lock or unlock the door. It is important that such a device be secure—users would not want adversaries to gain easy access to the house.

Figure 1: *The August Smart Lock.* The lock attaches to the deadbolt on the inside of the door and can receive commands via Bluetooth to lock and unlock the door.

We first articulate the security policy of the lock to clarify the security goals of the system. Second, we describe how the August Smart Lock system works. Third, we discuss a number of attacks and the results of attempting these attacks. Finally, we summarize the contributions of this security review.

2 Security Policy: Who can do what with the August Smart Lock?

In this section, we describe the security policy of the August Smart Lock. We outline the types of principal actors and what actions each actor is permitted, and how these permissions relate to the confidentiality, integrity, and availability of the system.

The August Smart Lock classifies users into the two types: `OWNER` and `GUEST`. An `OWNER` user is assumed to be a resident of the house and is effectively an administrator of the system. `OWNER`-level access implies a very high degree of trust, and would typically be granted only to, for example, a spouse or a co-owner of the property. A `GUEST` user is assumed to be someone the `OWNER` wants to grant temporary house access to. Any user who is not a resident of the house or someone a resident wants to grant access to falls outside of these two groups and should not have any of the permissions of the `OWNER` or the `GUEST`.

Users in the category `OWNER` have permission to:

- Lock and unlock the door
- Enable auto locking/unlocking
- Set the name of the house and the lock
- Calibrate and factory reset the lock
- Invite other users to be `GUESTS` or `OWNERS`
- View the list of all current `OWNERS` and `GUESTS`, their access levels, and their phone numbers
- Revoke `GUEST` or `OWNER` status from other users
- Configure when and for how long `GUEST` users should be able to unlock the door
- View log of all activity that has happened with the lock. This includes when the door was locked / unlocked by whom, when the door was locked / unlocked manually, and when access was granted / revoked from whom

- Delete their own account

Since an `OWNER` can view the logs of who locked and unlocked the door when, the system makes no claims to protect the confidentiality of the other users from the `OWNER`. Similarly, since an owner is allowed to enable or disable others' accounts, an owner has complete control over the availability of the system, including to other `OWNERS`.

The integrity of the system rests primarily in the log files, which are expected to accurately report who entered the house when. An `OWNER` should not be able to impersonate any other user and compromise that integrity. August accomplishes this by assigning each user a unique key for encryption and authentication. Even an `OWNER` does not know the keys of the other users, and so cannot impersonate them.

Users in the category `GUEST` have permission to:

- Lock and unlock the door at the times they are allowed by the `OWNER`
  - E.g. `OWNER` can give a guest access permanently, at regular weekly times, or for a specific one-time interval
- Enable auto locking/unlocking
- Delete their own account

Notice that `GUEST` users have limited knowledge of activity in the system and are not allowed to see unlock/lock logs or who the other `GUESTS` and `OWNERS` are. `GUEST` permissions should be limited like this in order to ensure confidentiality of the system.

Relatedly, `GUESTS` cannot grant access to or revoke access from other users. This restriction is necessary to maintain integrity and availability of the system for `OWNERS`.

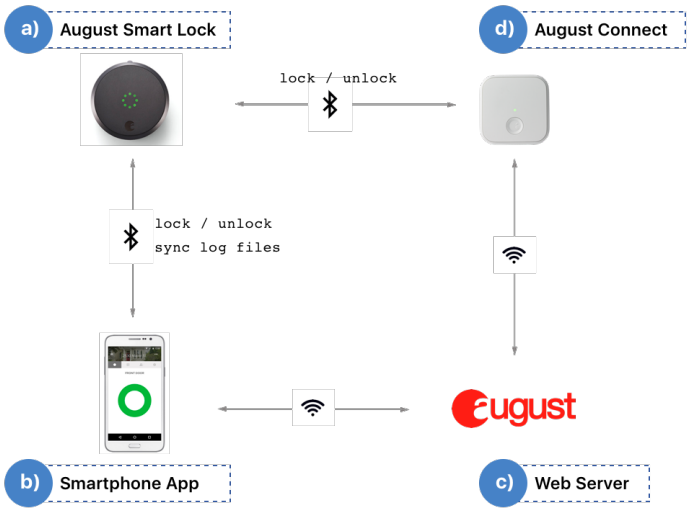
`GUESTS` and `OWNERS` both identify themselves by logging into the app with their phone number or email address and a password they chose. We will discuss this log-in system in more detail in section 4.1.

3 Background

3.1 How does the August Smart Lock work?

We first illustrate a high-level view of how the August Smart Lock system works and which modules it consists of. Then we discuss certain modules in more detail and the communication between particular modules. A high-level view of how the August Smart Lock system works is shown in Figure 2. The system consists of a) the smart lock itself, b) an app on a smartphone, c) the August web servers, and d) an optional accessory device called the August Connect, which allows remote control of the lock.

Figure 2: *Overview of the August Smart Lock system.* The August Smart Lock (a) communicates with a smartphone app (b) via Bluetooth, allowing users to lock and unlock the door. The smartphone app communicates with the August web servers (c) to grant access to other users or change related smart lock settings. The accessory device August Connect (d) allows remote control of the lock.



(a) *The August Smart Lock.* The smart lock is a battery operated Bluetooth device that attaches to the inside of a standard deadbolt. The lock communicates via Bluetooth Low Energy (BLE) with a smartphone. Through the August application on a smartphone, users can unlock and lock the door from their phone. The lock also stores a log of all activity. When an `OWNER` gets within Bluetooth range of the smart lock, the smartphone app syncs with the lock to update the log of all activity.

(b) *The smartphone app.* Homeowners and guest users operate the August Smart Lock system through the smartphone app. The app communicates with the lock via BLE and with the August web servers via WiFi. `OWNERS` can unlock the lock without WiFi, but `GUESTS` must communicate with the August web servers to unlock the door. Both `OWNERS` and `GUESTS` can enable auto-unlock, in which case the smartphone app tracks user location and sends an unlock message to the smart lock when the user gets in range.

(c) *The August web servers.* The August web servers manage user access to the lock. When an `OWNER` grants access to a new user via the app, the app sends a request to the August web server to update which user has which permissions.

(d) *The August Connect.* The August Connect is an accessory device that allows remote control of the lock. Without the August Connect, users must be within Bluetooth communication range in order to unlock the door using the app. With the August Connect, a user can send a request to the web servers to unlock the door; the web servers relay the request to the August Connect, and the August Connect unlocks the smart lock via Bluetooth.

Next, we provide a more detailed description of the secret keys associated with the August Smart Lock (Section 3.1.1), the Bluetooth communication between the smart lock and the smartphone (Section 3.1.2), and the WiFi communication between the smartphone and the smart lock (Section 3.1.3).

3.1.1 August Smart Lock: Secret Keys

The August Smart Lock communicates via Bluetooth with the smartphone app or the August Connect, and every such Bluetooth communication session starts by establishing a unique session key. The unique session key is then used to AES-encrypt messages between the lock and the phone. Section 3.1.2 describes the protocol for establishing this unique session key. We note here that the protocol for establishing a session key requires both the smart lock and either the web servers or the smartphone to already share some other secret key. This section elaborates on the different types of secret keys.

The lock itself can store up to 256 encryption keys, numbered from 0 to 255. The hacker jmaxxz found that the key in slot 0 has special privileges and coined this key the "firmware key." The August servers, users who are `OWNERS`, and the smart lock all know the firmware key, but `GUEST` users do not. The firmware key never changes, so an attack that successfully retrieves the firmware key would be a major vulnerability (4).

The other encryption keys stored on the lock are so-called "offline keys." A user

with `OWNER` permissions receive an offline key, i.e. a key which is available to the user even when the user is not connected to WiFi. This offline key can be used to establish a unique Bluetooth session key with the smart lock so the `OWNER` can unlock the smart lock without internet access. How these offline keys are initially shared so that both the lock and the appropriate owner's phone know them is unclear. We will discuss this in more detail later in Section 4.4, when we explain our attempt at decompiling the app.

3.1.2 Smartlock–Smartphone Communication via Bluetooth Low Energy

The August Smart Lock communicates via Bluetooth Low Energy with the smartphone. These packets are used for the lock to:

- Authenticate the phone
- Receive messages about granted and revoked user access
- Update the phone app about who has locked/unlocked the device

Type of encryption. The packets themselves are encrypted using AES encryption. AES is a symmetric encryption algorithm, so for sessions encrypted with the firmware key (for example), both parties must send ciphertexts encrypted with the firmware key.

How a communication session is established for `OWNERS`. Generally, each communication session uses a unique key. When an `OWNER` is communicating with the lock, this key is established jointly between the phone and the lock:

- The phone sends 64 random bits encrypted with the phone's offline key to the lock.
- The lock returns 64 random bits, also encrypted with the phone's offline key.
- The phone and the lock both decrypt the messages they received and concatenate the randomness, giving a new 128 bit AES-key to be used for the session.

How a communication session is established for `GUESTS`. The process is similar when a `GUEST` wants to communicate with the lock. However, because guests don't have offline keys, the guest's phone must communicate with the August servers as follows:

- The guest's phone sends 64 plaintext random bits to the server, which returns the encryption of those 64 bits with the firmware key.
- The phone transfers this encrypted message to the lock, which sends a new encryption of its own 64 bits to the phone. The phone passes the ciphertext from the lock to the server, which then decrypts the ciphertext and sends the plaintext back to the phone via SSL.
- The phone and lock both concatenate the jointly generated randomness to form a 128-bit session key (5).

Notice that this protocol opens two potential avenues of attack. First, a malicious guest could try to learn the firmware key by conducting a chosen plaintext attack–he would simply send messages of his choosing instead of randomness to the server when establishing the session key. Fortunately, in our research we found no chosen plaintext attacks that succeed against 128-bit AES.

Second, the session key is sent in plaintext between the server and the phone, and so an eavesdropper could learn it. However, since the session key is used immediately for communication between the phone and lock and is then discarded, this knowledge is unlikely to be of any use to an adversary.

3.1.3 Smartphone–August Web Server Communication via WiFi

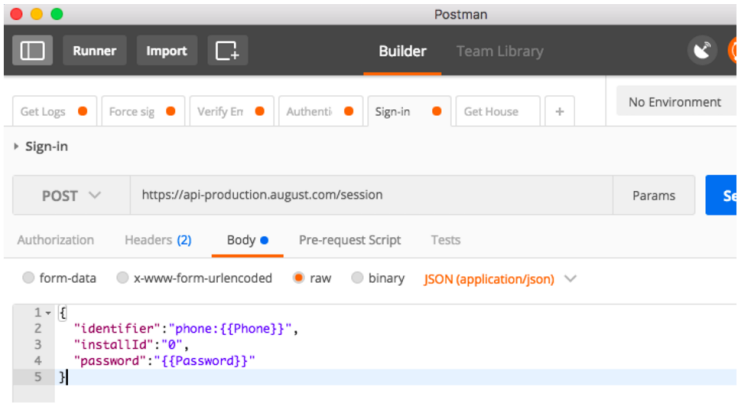
The August smartphone app communicates via WiFi with the August web servers. The first time a user signs into the app on the phone, the phone generates a random ID, called the installId, which is sent to the August web servers. This installId is used for future authentication of the phone by the August web server. For example, API calls from the smartphone to the August web servers typically require the installId and a general API key.

The hacker jmaxxz created a collection of API requests to be used with Postman, a tool that makes it easy to send different API requests. An example of an API request is shown in Figure 3 (3).

3.2 Previous attacks

In the past, other hackers searching for vulnerabilities have discovered problems with the Smart Lock's security measures. In 2015, during Security Compass's Hack Week, Paul Lariviere and Stephen Hall found a vulnerability in one of the API end-points that did not properly authenticate the user modifying the API. Unfortunately for August, this was the end-point that allowed adding guest access to the lock, so long as the attacker knew the lock's UUID. The attacker

Figure 3: *API example.* Jmaxzz created a collection of different API requests to the August app for the tool Postman. One example is a sign-in request, which is illustrated here, requiring an identifier, the installId of the phone, and the user password. The installId is randomly generated by the phone and sent to the August web servers the first time the user signs into the app. The installId is then used in the future to authenticate the user.



can read off this ID from any in-range lock. As a result, Lariviere and Hall were able to add themselves as guests to any in-range lock. August, to their credit, patched this vulnerability within 24 hours of its disclosure. This attack now fails (6).

In late 2016 at DEF CON 24, jmaxxz, whose work was mentioned above, found a variety of vulnerabilities with the August Smart Lock. For example, he was able to grant himself extra permissions by changing a string from 'user' to 'superuser' in the API. However, he also notes that August was very responsive in addressing the vulnerabilities that he identified (2).

4 Attacks

We describe four types of attempted attacks: 1) an adversary changing the August smartphone app password on a temporarily compromised phone 2) an adversary turning off WiFi to prevent revocation of access 3) an adversary with temporary guest-level access at a scheduled point in the future changing the date and time settings on their smartphone to gain access at a different time, and 4) snooping on the Bluetooth traffic between the smartphone and the smart lock. Finally, we will also describe further miscellaneous directions that we explored.

4.1 Password attack

August requires users to identify themselves by entering a password when they first log in to the app. They require this password to be reasonably complex: It must be at least eight characters and include one uppercase letter, one lowercase letter, one number, and one non-alphanumeric character. Once this password has been entered, the user stays logged into the app until he logs himself out.

If a user has not logged himself out of the app and an adversary has access to that user's phone for a few minutes, the adversary can change the user's app password. The adversary does not need to verify the old password in order to set a new password. However, this attack fails when the adversary tries to log in as the user on another device because the app requires email/SMS verification of the new device. The verification code is sent to the real user's phone or email account, so an adversary without access to at least one of these will not be able to log in. These verification codes are single-use only, preventing the adversary from using an old verification code from the SMS log of the user's phone.

The email/SMS verification message, however, only includes the verification code with no other information. The message does not suggest to the user that an adversary may be attempting to access their account if they were not expecting such a message. This isn't a security vulnerability, but it may confuse some users.

4.2 "I'm not Listening" Attacks

There are two ways a malicious actor can exploit the communication protocols between the app, lock, and servers to maintain access after it has been revoked. In the first case, an actor who used to be an owner is able disable WiFi and maintain access. In the other, a thief is able to circumvent August's remote log-out system. Both cases will now be discussed in detail.

4.2.1 Owner-Level Access Not Revoked

Owners can communicate with the lock offline. Consider the following steps:

1. Alice gives Bob `OWNER`-level access.
2. Alice gets out of Bluetooth range of the lock.
3. Bob maliciously puts his phone in airplane mode, preventing it from communicating with the August servers, but leaving Bluetooth enabled.
4. Alice revokes Bob's access.

In this case, Alice is unable to communicate with the lock because she is out of Bluetooth range. She is also unable to communicate with Bob's phone because Bob has disabled Internet connectivity. Therefore, neither the lock nor Bob's app receive the message that Bob no longer has access. Bob can then continue locking and unlocking the door as though his access had not been revoked. Bob's access cannot be revoked until Alice communicates with the lock. Furthermore, it seems that there is a bug in the lock's logging code and the log files will not properly report Bob's access during this period.

While the described attack is a little alarming in theory, it is unlikely to be a real problem in practice. `OWNERS`, by definition, can revoke each other's access. In fact, if Bob were truly malicious, he could have revoked Alice's access after he was granted `OWNER` status. For this reason, the original owner should not give `OWNER` status to anyone she does not trust immensely.

Furthermore, users with `GUEST` access cannot perform this attack, because `GUESTS` must access the August servers during every communication session with the lock. The August servers will have received the message from Alice that the `GUEST` no longer has the proper permissions and the `GUEST` will be denied access to the lock as expected.

4.2.2 Stolen Phones

If an adversary permanently compromises a user's phone, there are two possible scenarios. In the first case, the adversary might manage to unlock the stolen phone. If the user has not logged themselves out of the app, this gives the adversary direct access to the app. There is no password protection and no functionality that automatically logs out the user after a certain amount of time has passed. The lack of password protection or auto-logout is potentially concerning, as phone theft would give an adversary access to the user's home. Note that this attack of stealing someone's phone is similar to stealing someone's regular keys.

In the second scenario, the adversary might physically have the phone but not be able to unlock it. In this case, the thief could still have access to the user's home if the user has enabled auto-unlock. Auto-unlock automatically unlocks the door when the phone is in proximity without requiring the user to actually access the app, enabling the thief to get into the home.

In either scenario, the user can protect herself by following a set of instructions on the August website to report a lost device and log out of the account on all devices. However, the thief can prevent the app from logging out by putting the stolen phone in airplane mode (which generally does not require the phone to be logged in). In this case, the app will not get the message to log out and will stay logged in. Note that the thief would have to re-enable Bluetooth, which

may or may not require logging in to the phone.

In practice, the stolen phone attack is not a major security vulnerability—it is no less secure than having a key stolen, and a user can always deny a thief electronic access by simply removing the batteries of the smart lock. The only real problem is the false sense of security given by the instructions on the August webpage. We have suggested that they add instructions to remove the smart lock batteries to ensure that the thief does not have access, even if he disabled WiFi and tricked the app to stay logged in.

4.3 Changing date and time settings

Another attack we tried aimed to allow a `GUEST` who was only authorized to unlock the door at certain times to gain access at other times. The idea was simple: By changing the date and time settings on the phone, we hoped to trick the phone—and therefore the app—into thinking it was a time other than it was.

This attack was unsuccessful. As `GUESTS` cannot communicate with the lock unless they communicate with the August servers, we conclude that the permissions were probably based on the time recorded by the August servers, irrespective of the time recorded by the phone.

4.4 Snooping Bluetooth packages

We investigated the possibility of a replay or replay-like attack. Although it will be obvious to anyone reading this report that such an approach will not work, at the time of the attempt we did not know the phone-lock communication protocol. It was through investigating the possibility of a replay attack that we learned about the defenses against it, so we will still report the steps we took.

The phone and the lock communicate over BLE. On an Android phone, it is easy to log BLE packets by checking the appropriate box in "Developer Options." This means that a `GUEST` with temporary access to the house would certainly be able to log BLE packets. Additionally, BLE sniffers exist and are not too expensive. It is conceivable that an adversary could get close enough to use a BLE sniffer when an owner unlocks the door.

BLE is optimized for low-energy, so the packets are small and at most 20 bytes. We inspected multiple unlock commands from the same phone, hoping that we could either completely determine a sequence of bits that would unlock the door or that we would be able to determine most of the bits of such a sequence so that a brute-force search of the remaining space was feasible.

We were disappointed on both counts. We examined the Bluetooth logs in Wireshark and found that the unlock sequence was not a single 20 byte packet. Instead, the unlock sequence was a complicated exchange of many packets between the lock and the phone. In addition, the packets for two unlock commands did not look alike. The lessons from this attempt eliminated the possibility of a straightforward replay attack.

The differences between the packets for the two unlock commands are due to the existence of the session key described in Section 3.1.2. Recall that the lock contributes half the randomness of the session key. We have not found a way to trick the lock into generating the same randomness repeatedly. The existence of the unique secret session key protects the lock from a straightforward replay attack, because a message encrypted with a key other than the session key would be meaningless to the lock and therefore ignored by the lock.

Similarly, a brute-force attack was infeasible. The messages exchanged by the protocol were far too long for brute-force guessing a ciphertext to be a reasonable approach. One alternative is to attempt to guess the randomness that the lock returns. This approach requires that the attacker knows the plaintext that needs to be encrypted, so that the she can use the guessed randomness to generate a correct ciphertext. Remember that the lock contributes 64 bits of randomness to the session key. Even if we assume the attacker knows the plaintext, the chance of successfully guessing the lock's randomness in one try is only 1 in 2^{64}, meaning the attacker could expect to succeed in about 2^{63} tries. Each try requires the exchange of at least 40 bytes of data over BLE, which uses a communication rate of about 2 Mbits/s. This means 2^{63} guesses would require:

$$\frac{40 * 8 \text{ bits/packet}}{2e6 \text{ bits/s}} * (2^{63} \text{ packets}) = 1.48e15 \text{ seconds} = 4.7e7 \text{ years}$$

Therefore, we conclude that the requirement of a session key is a good defense against both replay and brute-force attacks.

The logged Bluetooth files we captured as well as a more detailed analysis of the differences between commands is included in the auxiliary material submitted with this report.

4.5 Further Attempted Attacks

We pursued a few other kinds of attacks that we briefly describe here. We tried decompiling the app code, sniffing the TCP packets between the smartphone and the August web servers, making unauthorized calls to the August web server API, and acquiring special keys by inspecting the August app file system.

4.5.1 Decompiling the App

Using several programs available online (specifically, APK Extractor, the Android Studio debugger, and JD-GUI), we were able to extract and decompile most of the source code for the August app. This resulted in about 97,000 lines of code, spread across about 3,300 files, most of which had names like aaa.java. Furthermore, not all of the files were successfully decompiled, including some with tantalizingly informative names, such as "AugustEncryption.class." We don't know if the failure to decompile these files was the result of intentional obfuscation by August or if it was just a natural result of the limitations of the decompiler. We did not try other decompiler programs to see if they could produce better results.

Despite the incompleteness of the decompiled code, and the difficulty of reading the code that was successfully decompiled, a determined adversary could probably reverse-engineer most of what the app does. There are enough print statements and variables that somehow did get descriptive names that a search for keywords is useful. For example, we searched for the string "invite" as we were attempting to learn how the initial key exchange was done. This search returned about 100 hits throughout the files. Sorting through the 100 hits to find the definition of the protocol was a tedious process that we did not have time to finish, but in principle we believe it could be done. Even the files that don't decompile properly can be processed using the javap tool, which disassembles the .class files and outputs the machine instructions in somewhat human-readable format.

4.5.2 Sniffing TCP packets

We wanted to see if we could learn anything about the August communication protocol or capture certain encryption keys by inspecting the TCP packets between the phone and the August web servers.

First, we collected the TCP packets by 1) connecting a phone to laptop, 2) finding the UDID of the phone, 3) starting a virtual network interface with the phone, and 4) using tcpdump to collect TCP traffic to and from the phone. We inspected the resulting TCP packet file in Wireshark. We verified that the packets were encrypted, but were not able to glean any other information from them.

4.5.3 Man-in-the-middle Attacks

There are a number of conceivable attacks if one can communicate directly with the August web servers. The hacker jmaxxz was able to do so using Postman and

made public the work he did with Postman to call the August API. We installed Postman and imported jmaxxz's code, but learned that it requires certain keys to communicate with the August API. We will discuss the difficulties we had with acquiring these keys in the next section.

4.5.4 Retrieving `OWNER` offline keys

Because owners receive offline keys, these keys must be stored somewhere. We attempted to find these offline keys by inspecting the August app file system using the tool i-Funbox for iPhone file management. However, we discovered that we needed to jailbreak the iPhone in order to further inspect the file system. Jailbroken iPhones are able to gain root access to their own operating system, and thus are able to install third party applications that are not available in the standard App Store.

Unfortunately, jailbreaking iPhones is non-trivial. Apple has a clear incentive to prevent users from downloading third party applications, so on newer versions of iOS it is often impossible to jailbreak the phone. Most jailbreak utilities exploit some vulnerability in order to gain root access. Apple in turn patches vulnerabilities in a sort of ongoing escalation.

We were working with an iPhone 4S running iOS 8.2, but the August Home application requires iOS 9.0 or later. As such, we needed to update the iPhone 4S to iOS 9.3.5. Although there are jailbreaks available for iOS 9.0 to 9.3.3, Apple prevents users from selectively downloading and installing an older version of iOS. Unfortunately, we found no jailbreaks available for iOS 9.3.5 (7).

We also had access to two iPhone SEs, but both were already running later versions of iOS that could not be jailbroken nor downgraded to an earlier version of iOS. It may also be possible to root a recent Android phone, but we did not investigate this possibility in detail because we did not have access to an Android phone that we were willing to attempt to root. Consequently, we were unable to proceed with the attack to retrieve the `OWNER` offline keys.

Notice that even if we could obtain the offline keys from the app and use them to get persistent access to the lock from another device, that does not in itself pose a relevant security concern. First, another owner could still, most likely, revoke the access of the compromised owner's keys and prevent the attacker from entering the home. Secondly, an attacker would need to steal the unlocked phone of an owner who has not updated their operating system since July 2016, given the current lack of more recent jailbreaks. As such, this potential avenue does not seem like a security concern.

It's also possible that an `OWNER` could retrieve the firmware key in this way. As previously mentioned, this poses a bigger security concern since the firmware

key doesn't change and gives those who know it the ability to add new users to the lock. This illustrates the importance of tightly restricting `OWNER`-level access and taking proper precautions if an `OWNER`'s phone is stolen.

5 Areas for Future Work

Our work so far has focused on the interactions between the August app, the server, and the smartlock. We did not have time to investigate the August Connect, which also communicates with the lock. It would be interesting to do a review of this device and see if there are any vulnerabilities lurking in those protocols.

Like previous work assessing vulnerabilities in the August Smart Lock, our work was done with a new device. Therefore, we do not know what, if any, vulnerabilities might emerge as the device ages. It is possible that wear on the hardware could make some attacks easier, particularly if physical sensors degrade over time.

What new vulnerabilities exist after months or years of use? How robust are the motors and sensors in the device? While most of these concerns are more related to usability than to security, some reviews say that if the deadbolt does not correctly slide into place then the lock may report that it is locked when it in fact is not, a clear violation of the integrity of the system (1).

6 Conclusion

We have discussed August's security policy and provided background on the device, its communications, and past successful attacks. We have discussed attacks that we attempted as well as defenses against those attacks that August currently employs. We also outlined a few areas for future work, where vulnerabilities could potentially exist.

In our security review of the August Smart Lock, we were unable to uncover any major vulnerabilities in the device. We have no reason to believe that the August Smart Lock is any less secure than a conventional lock, and for a typical user, it seems unlikely to be a weak point of defense in home security. That said, there have been concerning vulnerabilities in the August Smart Lock uncovered in the past. These are weaknesses that once threatened the security of all August users. Unlike a conventional deadbolt and key, which could not be compromised en masse for thousands of homes at once, a savvy attacker discovering a new vulnerability could theoretically sell access to homes of August users worldwide.

The security team at August seems aware of these risks and works proactively to protect the security of their users. To the security team's credit, the vulnerabilities identified by jmaxxz, Lariviere, and Hall were all repaired within days. Furthermore, August was very willing to work with us in conducting a security review of their device. We believe that their apparent culture of responsiveness and proactivity are powerful drivers of the security of their system.

7 Acknowledgements

We would like to thank our Professors Ron Rivest and Yael Kalai for their insights on the field of computer security, as well as our TAs Wei-En Lee, Cheng Chen, and Heeyoon Kim for their help and advice. We are also very grateful toward Andy Rothfusz, Director of API Integrations at August, for his continued help and support. This project would not have been successful without his assistance.

References

[1] Hollister, S. August smart lock and connect review: I'll use keys, thanks. http://gizmodo.com/august-smart-lock-and-connect-review-ill-use-keys-tha-1685319060, 2015.

[2] Jmaxxz. Def con 24 - jmaxxz - backdooring the frontdoor.

[3] Jmaxxz. The August smart lock's not so 2-factor authentication (part 1). https://jmaxxz.com/blog/?p=476, 2015.

[4] Jmaxxz. August lock firmware keys. https://jmaxxz.com/blog/?p=550, 2016.

[5] Jmaxxz. BLE session. https://github.com/jmaxxz/keymaker/wiki/BLE-Session, 2016.

[6] Lariviere, P., and Hall, S. Making smart locks smarter (aka. hacking the august smart lock). http://blog.perfectlylogical.com/post/2015/03/29/Making-Smart-Locks-Smarter, 2015.

[7] The iPhone Wiki. Jailbreak, 2017. [Online; accessed 17-May-2017].