Secure Read/Write Authentication for Data Port

Brendan Horng, David Kim, William Uchegbu, Congyu Xu

Abstract

Onity locks are widely used by hotels, however the locks have a considerable security flaw. Without a security layer in place, the lock's memory can be directly accessed by connecting to the data port under the locks. The information in memory of each lock is sufficient to unlock any other lock in a hotel via the sitecode. To avoid leakage of that information, we propose a security layer that, instead of being able to access a lock's memory via data port directly, secures the data port by only processing commands with valid device ids. This adds minimal computational overhead to the current system while addressing the most significant security vulnerability of the Onity lock system.

Introduction

Electronic locks in hotels are very useful for hotel staff to manage access to rooms. The front desk can easily give or withdraw access to certain guest rooms. However, Brocious found the vulnerabilities of locks from Onity [5]. Through the use of homemade hardware and Brocious's script [2], by plugging the hardware into the DC port of Onity locks, an attacker can read the lock's memory in which the sitecode can be found, and issue an "open" command, which then will open the lock of the door [7][10]. With the sitecode and a little other information in the memory [7], an attacker can also produce a "programming card", which can be used to enter all rooms in the hotel.

Onity provided a trivial solution of torx screws for the lock case[3], plastic plugs for the data ports[8], and offered a more comprehensive solution for a fee[3]. Unfortunately, plastic plugs and Torx screws won't stop an intentional attacker who has already built a hardware system in order to hack, especially for the many hotels that chose not to upgrade at a cost.

To fix this vulnerability, our team has worked out a security layer to protect the lock's memory through separating the data port from the lock's logic. Instead of direct access to lock's memory, the data port makes an RPC call to the lock with the portable programmer device ID as credential. Only when authentication is successful, the lock will send information back to the data port.

Background

The Onity lock is designed to use a stored site code in order to encrypt and decrypt key values stored on guest cards. This authenticates them and gets them through the door. The three essential aspects of the system are the programmer, the encoder, and the lock itself. The programmers are used to load key values and other pertinent information onto the locks while the encoder is used to load the programmer and encrypt the key cards.

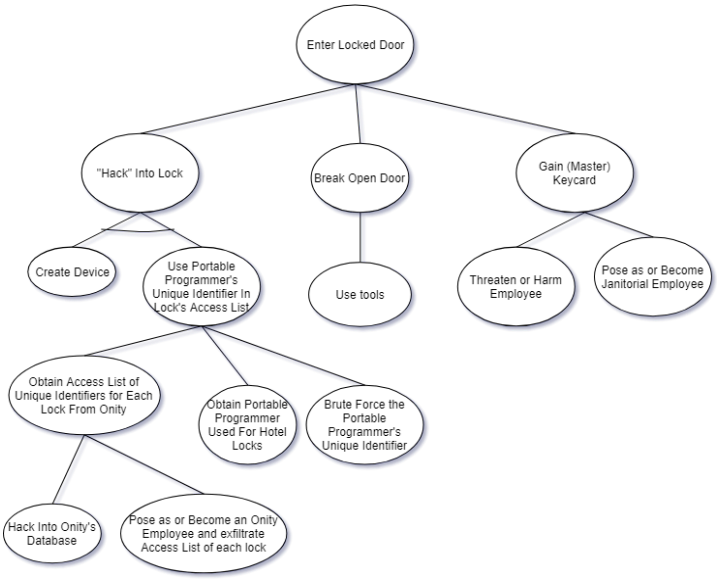
The sitecode is a 32-bit value that is critical to the system's function. The code is used to mint keys by encrypting and decrypting cards, opening locks, and programming them.

The lock communicates by using a single wire protocol that moves data in both directions. The bottom of the lock contains a port mainly used to provide power and to carry data as well. A noteworthy aspect of the lock communication is that the device that communicates with the lock is in control of the sync pulses that have the actual communication happening on them.

The lock communication system allows for read access to the locks memory. Taking advantage of this, the sitecode can be pulled directly from memory as long as the attacker has knowledge of where the sitecode is located in address space. Knowledge of the basics of the lock system allows for an attacker to grab key data such as the sitecode and use it to create key cards. The main security flaw in this system is the unauthenticated communication that enables the memory access. Brocious notes three key conclusions when discussing the flaws of Onity locks, one being, “the lock communication port is unauthenticated and enables direct memory access, which allows arbitrary reading of memory”[2].

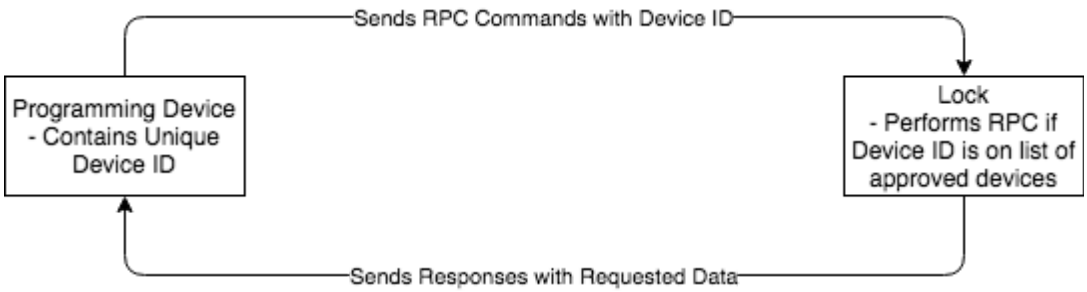
Threat Model

Considering the vast vulnerabilities with Onity locks, it’s essential to consider not only the importance of the assets we are trying to protect, namely hotel guests as well as property belonging to both the guests and the hotel, but also the threats that exist to the Onity locks themselves. The solution we have come up with is a part of the ultimate solution in securing the hotel’s assets, however, threats continue to exist regardless of the situation and/or solution. An attack tree found below can address some of these concerns in relation to our proposed solution which will be discussed further in the next few sections:



Design

The proposed implementation is an additional security layer protecting read and write access to the data port. This security layer would be implemented on the individual locks themselves and will require an approved device ID for any connected device prior to allowing read and write access. Additional allowed devices can be added by a preexisting approved device. This device code would be sent along with any Remote Procedure Call (RPC) request made by a connected device to the lock. Failing to send an approved device code will result in the RPC being dropped and no response will be given to the connected device. This will prevent approved devices from accessing and manipulating the memory of the lock.



Implementation

Device IDs would be a unique 128 character sequence for each programming device. These would be programmed to each device when a lock system is set up. The programming devices can then communicate to the locks via the data port. Each programmer would also be secured with authentication to reduce the risk of a stolen or taken programmer becoming compromised. An administrator will need to log into the programmer and then connect the programmer to the data port. The data port would implement a serial connection schema such that it would take an RPC command of the form: <command> <device_id> …<params> where “params” are a variable amount of parameters corresponding to the command. Each parameter in the command is separated by a space, and the command is terminated by a new line character. The lock would store the hashes of all of the approved device ids. Upon receiving an RPC command, the lock will first hash the provided device id and check if it is in its list of approved devices. If it is not, the lock will drop the command and continue to listen for requests. If the device id is on the list of approved devices, the lock will process the RPC command and send back a response to the programmer. Below is a sample implementation of the logic that would occur to process an RPC command on the lock. The serial communication is simulated through standard input and output.

```
from hashlib import sha256

approvedIds=["2bb80d537b1da3e38bd30361aa855686bde0eacd7162fef6a25fe97bf527a25b"]
SECRETKEY="4723897593823"

def getHexDigest(data):
    return sha256(data.encode('utf-8')).hexdigest()

def handleRPC(cmd, params):
    if cmd == "readSecretKey":
        print(SECRETKEY)
    elif cmd == "addApprovedId":
        approvedIds.append(getHexDigest(params[0]))
    elif cmd == "removeApprovedId":
                approvedIds.remove(getHexDigest(params[0]))

while True:
    cmd=input()
    params=cmd.split()
    if len(params) < 2: # Insufficient Parameters
        continue
    if not getHexDigest(params[1]) in approvedIds:
                continue
    handleRPC(params[0], params[2:])
```

Evaluation

Evaluation of the proposed solution to secure read and write access to the data port prevents the most significant vulnerability of the current Onity lock system while introducing minimal overhead and additional logic. The added authentication system is not computationally taxing and should not introduce much more power consumption as it would only occur when a device is connected to the data port. Additionally, power could be fed from the external device to the lock through the data port to offset additional power consumption if needed. Potential downsides to this solution is that currently installed Onity lock systems will have to either purchase new locks or flash new software to existing locks. Current systems may also need to purchase new programming devices if maintenance is needed. There are additional vulnerabilities with Onity lock systems, such as the low search keyspace of the code key values making it susceptible to brute force algorithms [2], that were not addressed in this paper. This solution is particularly targeting unauthorized access to the memory of the locks through the data port to retrieve the site code thus instantly allowing access to any lock. A potential vulnerability to the presented solution is there is no encryption on the communication between the lock and the programmer. This is because there is minimal chance of any sort of man in the middle attack regarding these two devices connected physically by a wire. Though it is worth noting, potentially an attacker could open the lock up and attach a device to the data port. An unsuspecting or ignorant administrator could then attach the programming device to the attacker's device without noticing. At this point, the device could store the traffic between the programmer and the lock while still allowing data to pass through as to not arouse suspicion. The attacker could then just retrieve the device and read through the traffic. This form of attack would be difficult and can be combated through proper training of administrators using the programming devices.

Related Solutions

Our solution to the problem addresses a security layer that is placed to block unauthorized communication with the lock hardware. Some other solutions have been outlined in the following:

A. Onity's *Solution*

Some of Onity's solutions [3][8] to the problem were to replace the regular screws for the lock case with Torx screws, place plastic plugs for the data ports, or replace the circuit boards and firmware for a fee. Their firmware solution is still obscure to the public, and may or may not have fully addressed the issue of direct memory access.

B. Cryptographically Secure Communication

Aquila et al. [9] have also proposed a theoretical solution, prior to the Onity hack, regarding cryptographic keys of each interaction between the components of the keycard management system, which deals with the security of communication between the key management components. Aquila et al. utilizes AES for the encrypted communication between the following: key and lock, lock and portable programmer, and portable programmer and encoder. This means that each component will have their own keys to match against so that not only is there a recognition among the components, but also security in their interactions.

Conclusion

In this paper, the solution presented addresses the most significant vulnerability in Onity's lock system and a central component to the hack done by Cody Brocious. The solution presented manages to solve this problem while introducing minimal computational overhead while providing the same functionality as the current system. Further work can be done regarding the relatively low keyspace of the sitecode and the code key values to further secure this system. In addition, further work can be done to improve the current solution by providing additional security layers on the programming devices in order to minimize breaches of valid device ids.

References

[1] Media, S. (2012, July 31). Black Hat Video: Hotel room locks may not stay locked. Retrieved from https://www.youtube.com/watch?v=JIOnjGihUgg

[2] Daeken. (2012, August 10). Daeken/LockResearch. Retrieved from https://github.com/daeken/LockResearch/blob/master/paper.md

[3] Greenberg, A. (2012, December 06). Hotel Lock Firm's Security Fix Requires Hardware Changes For Millions Of Keycard Locks. Retrieved from https://www.forbes.com/sites/andygreenberg/2012/08/17/hotel-lock-firms-fix-for-security-flaw-requires-hardware-changes-for-millions-of-locks/

[4] Frentz, M. (2015, January 21). IMM Asset Management Solutions - Onity Portable Programmer. Retrieved from https://vimeo.com/117446829

[5] Greenberg, A. (2012, November 26). Hacker Will Expose Potential Security Flaw In Four Million Hotel Room Keycard Locks. Retrieved from https://www.forbes.com/sites/andygreenberg/2012/07/23/hacker-will-expose-potential-security-flaw-in-more-than-four-million-hotel-room-keycard-locks/#79b0d4d3eb85

[6] Seiders, T. (2012, October 12). News Archives. Retrieved from https://www.hotel-online.com/archive/archive-45555

[7] Mattoon, C. (2017, November 03). Onity Hotel Lock Exploit. Retrieved from http://cmattoon.com/onity-hotel-lock-exploit/

[8] Silent Pocket. (2017, March 31). Onity Locks Still Used In Many Hotels Vulnerable. Retrieved from https://silent-pocket.com/blogs/news/onity-locks-still-used-in-many-hotels-vulnerable

[9] D'Aquila, K. J., Gerratana, F. L., Oteri, A. P., & Rosenberger, J. A. (2004, March 10). *Crypto-SmartLock: Applying Cryptography to Physical Security* [PDF]. Worcester Polytechnic Institute. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.470.3312&rep=rep1&type=pdf

[10] Jaku. (2012, October 05). Onity Lock Unlocked with a Dry Erase Marker Extended. Retrieved from https://www.youtube.com/watch?v=n1JEQ6ZsX0Y