Attacking the Square for Chip Cards

Mark Cummins
Undergraduate

Cynthia Okuno
Undergraduate

Department of Information and Computer Science
University of Hawaii at Manoa
Honolulu, HI
{markrc, okunoc} @hawaii.edu

1. Abstract

We consider the security of Square Inc's newest rendition of its mobile readers, the Square for chip cards and its Square Register app. With the original Square Reader being prone to a number of vulnerabilities such as device modification and credit card skimming [1], we use the same methodology to identify any vulnerabilities in the Square for chip cards. Even though chip and pin cards, also known as EMV, have been regarded as secure because of their new designs, many tests have shown that it's still susceptible to the same problems magnetic strip cards had. We attempt to attack the Square for chip cards using the known vulnerabilities EMV has as well as the design flaws that were prevalent in previous Square readers. This investigation will confirm whether these new mobile point-of-sale systems suffer from the same software and hardware design flaws as their precedents and whether it leaves them vulnerable to third parties or merchants with malicious intent.

2. Introduction

The innovation of phones and tablets with internet has allowed businesses to utilize new forms of transaction hardware. Companies such as Square, Paypal, and Venmo introduce solutions to businesses that cannot utilize the traditional and expensive point-of-sale systems. With such an affordable entry point, street vendors and food truck owners can accept cards and process them just like traditional systems within limited space. Sales from Square's original reader reached $20 billion in 2013 alone [2] and has even processed $100 million in a single day [3].

But these achievements in increasing sales volume and hardware convenience should not overshadow the original Square reader's numerous security vulnerabilities. During the first iterations of the Square reader, the hardware didn't include an encryption chip. By sampling the the device's microphone input quickly, an application like Audacity or other sound recording applications can read the voltages produced. Then, by examining the zero-crossings, the sound could be decoded into unencrypted credit card information [1].

Even with the enhanced security brought by EMV. EMV by itself cannot protect consumers from fraud. In numerous papers, researchers have found ways to commit fraud through EMV through defects in ATMs that do not produce unpredictable numbers and creating man in the middle devices that nullify the entire authentication process of a pin. Though it has been reported that the use of EMV has reduced fraud, there is a flaw in this statistic. Ever since EMV has been implemented in Europe and Asia, there has been a liability shift from the credit card companies to the customers. Numerous reports of fraud have been regarded as customer negligence and thus rejected because of EMV even though papers have shown that fraud can still be conducted [2].

In our investigation of the ongoing attacks on mobile card reading devices, we continue on to the Square for chip cards. Being founded in 2009, Square has innovated mobile transactions with its first Square reader. Since then, Square has created four versions of its original reader and has now delved into the EMV reader and contactless payment. With the latter two being new to the industry, there is always a possibility that there are vulnerabilities that need to be addressed.

3. Background

Traditional credit cards encoded information through the magnetic stripe on the card. When swiped, a voltage signal is sent out that is modulated uniquely on every card. Using a Manchester coding scheme, microcontroller or a similar device can decode the signal. Numerous vulnerabilities have been exposed that have pushed credit card companies to develop a new card that strengthens the card against fraud.

EMV (Europay, Mastercard, and Visa) also known as chip and pin cards were developed in mind to replace magnetic strip cards. EMV works by requesting the terminal or ATM to send a nonce or unpredictable number to authenticate and secure the transaction. EMV has been touted as a bona fide solution to the fraud and vulnerabilities stemming from the vulnerabilities in the magnetic strip design. But independent security researchers, through theory and practice, have shown that protocol of EMV is flawed when the ATMs accepting EMV have very weak or predictable random number generators. Because of this design flaw, EMV is essentially susceptible to fraud like its predecessor [2].

Terminals and ATMs execute EMV protocol with the chip on EMV cards. This exchange transaction data embedded with a cryptographic message authentication code (MAC) which is calculated with a symmetric key. This key is stored in the card and is known to the card-issuer. This was designed with the bank being able to detect a counterfeit card since they wouldn't contain the key and extracting this key from a real card would be very difficult. In addition to this design came the latter half of EMV's nickname: the PIN. The idea of the PIN for cards was to make it harder to use a stolen card in the day to day world. The chip is used against card counterfeiting and the PIN is used to combat stolen cards. Alas, chip and PIN.

Unfortunately, EMV did not decrease fraud as its designers optimistically predicted. The transactions that were beyond the scope of EMV like online, mail, and phone based orders still remained prevalent. And in the scope of EMV's design, it still didn't decrease fraud. For our research, Square's reader for chip cards, like other POS terminals, will be tested for vulnerabilities.

Because there is no difference between the Square reader for chip cards and other POS terminals that support EMV besides physical size, the Square reader for chip cards will follow the same transaction flow.

Based off an ATM transaction, a POS terminal would be similar:

1. Card Authentication: Card details are read and authenticated by ATM or POS terminal
2. Cardholder Verification: Card presenter is verified as card owner through PIN or signature
3. Transaction Authorization: Issuing bank decides if the transaction should be authorized or declined.

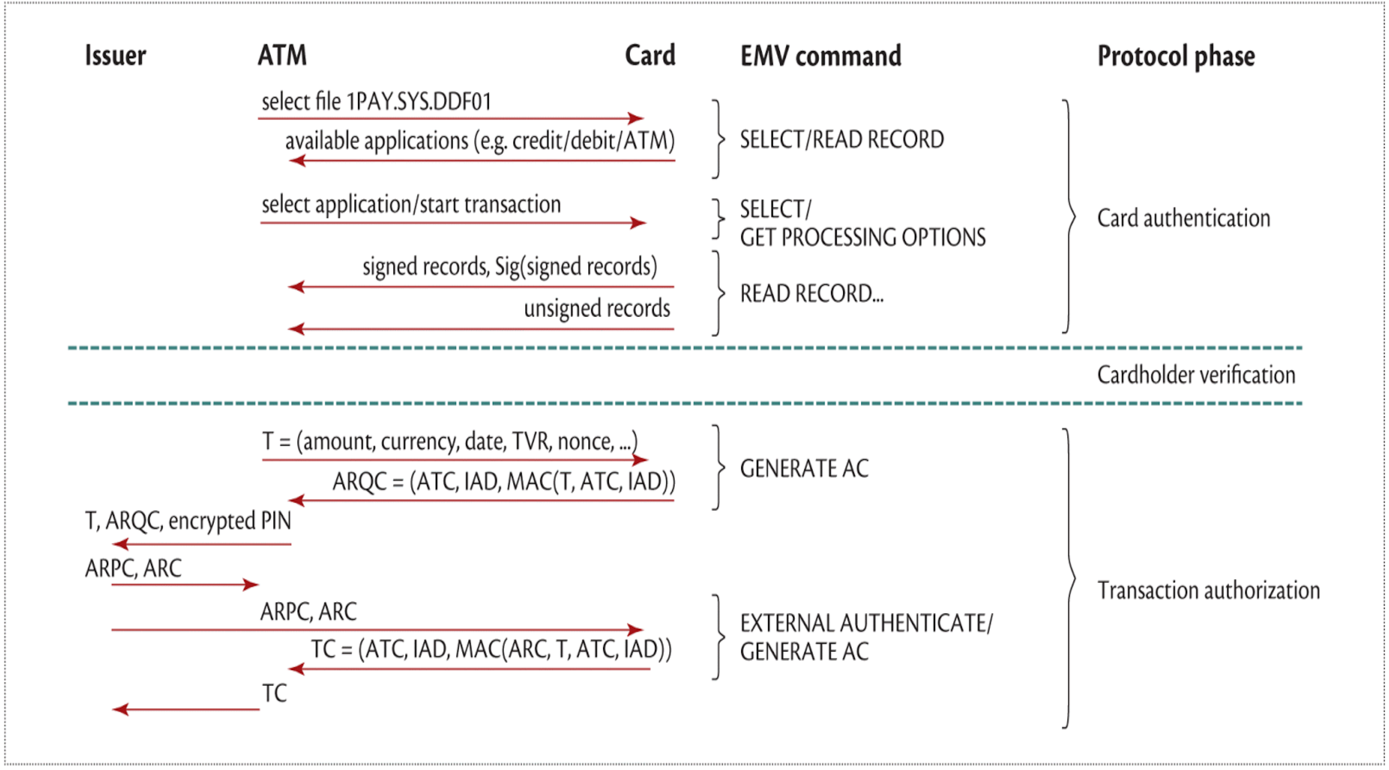

Issuer
ATM
Card
EMV command
Protocol phase
select file 1PAY.SYS.DDF01
available applications (e.g. credit/debit/ATM)
SELECT/READ RECORD
select application/start transaction
SELECT/
GET PROCESSING OPTIONS
Card authentication
signed records, Sig(signed records)
unsigned records
READ RECORD...
Cardholder verification
T = (amount, currency, date, TVR, nonce, ...)
ARQC = (ATC, IAD, MAC(T, ATC, IAD))
GENERATE AC
T, ARQC, encrypted PIN
ARPC, ARC
ARPC, ARC
TC = (ATC, IAD, MAC(ARC, T, ATC, IAD))
EXTERNAL AUTHENTICATE/
GENERATE AC
TC
Transaction authorization

Figure 1: Outline of an EMV transaction at an ATM.

In the process of card authentication, data records are traded back and forth between the card and the ATM. This data includes the card number, start and expiration dates of the card, and permissions the card has (credit, debit, etc.). The card also sends a static RSA digital signature over certain records to prevent criminals from making counterfeit cards from an account number.

During cardholder verification, the ATM requests a PIN or signature for verification (the card is actually not involved in this authentication). The card owner then types their PIN, gets encrypted, then sent to the card issuer for verification awaiting authorization.

The last part, transaction authorization, is carried out. The ATM sends the card owner's amount, the currency the amount is shown in, date, the terminal verification result, and the nonce (the EMV unpredictable number). After, the card responds with an authorization request cryptogram (ARQC) which is a cryptographic MAC using a combination of the supplied data, the application transaction counter (ATC) and the issuer application data (IAD). The ATC is a 16 bit number stored by the card and incremented after each transaction and the IAD is a proprietary data field that carries information from card to issuer.

The ARQC is sent from ATM to the card issuer along with the encrypted PIN. The card issuer network then verifies both fields by recalculating the MAC over the received data fields. Another checks are made, such as whether there is enough funds in the card, the card has been reported stolen, and whether the risk analysis algorithm determines the transaction to be suspicious. After the issuer verifies everything, an authentication response cryptogram

(ARC) is sent in response allowing the ATM to fulfill the transaction.

The Square reader for chip cards follows a very similar outline. The cashier either inputs the cost or selects a predetermined cost from inventory. Next, the reader asks which type of transaction will be used. In our case, the main focus is on the credit/debit card. The cashier then takes the customer's EMV card and inserts it into the reader. The reader then goes through the three phases that an ATM transaction would do. If the transaction is verified and accepted, the cashier then hands the device to the customer to agree to the purchase by signing the device. The cashier is then allowed to remove the card from the reader and hand it back to the customer.

There is an additional layer of communication that needs to be done in the case of the Square chip and pin reader. The chip and pin reader uses Bluetooth to pair with a user's phone to send the information to the phone, then via the Square app, it sends the appropriate data back to Square via 3G or Wi-Fi. With Bluetooth sniffers like the BlueSniper Rifle, it's easy to eavesdrop and attack users from a mile away or more [4].

Man-In-The-Middle attacks on Bluetooth are rare, but very doable. It is possible to grab a normal $30 Bluetooth dongle and create a Bluetooth sniffer [4]. Tests done by Keijo Haataja and Pekka Toivanen show that Bluetooth sniffers for various versions are very capable of eavesdropping on various Bluetooth devices. They also go over the reverse engineering tools available for Bluetooth devices, which allow a potential attacker to poke holes in firmware, and to even overwrite official firmware and write their own malicious firmware in its place.

4. Novel Idea

The purpose of this paper is to use the same information presented in EMV and its appliances and apply it to the Square reader for chip cards. Currently, there has been no discovery of vulnerabilities present in Square's newest hardware (most likely because no one has tried finding any yet). We want to find out whether the reader for chip cards can be attacked on both fronts, hardware and software.

5. Experiments

The experiment will run as described. It will consist of both experiments on the hardware side and on the software side. It will range from scenarios where malicious intent will be based on a merchant who owns a reader to someone spying on data being transmitted through it.

5.1 Experiment 1

Disassemble the reader carefully (without physically destroying it), analyze its parts, remove the encryption chip (if there is one), and reassemble it in a way that makes it look like it was not tinkered with. Then, using a wifi network to connect the reader to the internet, use computer using either Wireshark or snoop to read the credit card information being transmitted through the Wi-Fi channel. This will attempt to exploit the reader and capture important information by intercepting it from Wi-Fi. This will emulate a store using a reader that's connected to the internet through the store's Wi-Fi.

Results: We successfully disassembled the reader without damaging the device. It appears completely untampered with when the device is put back together.

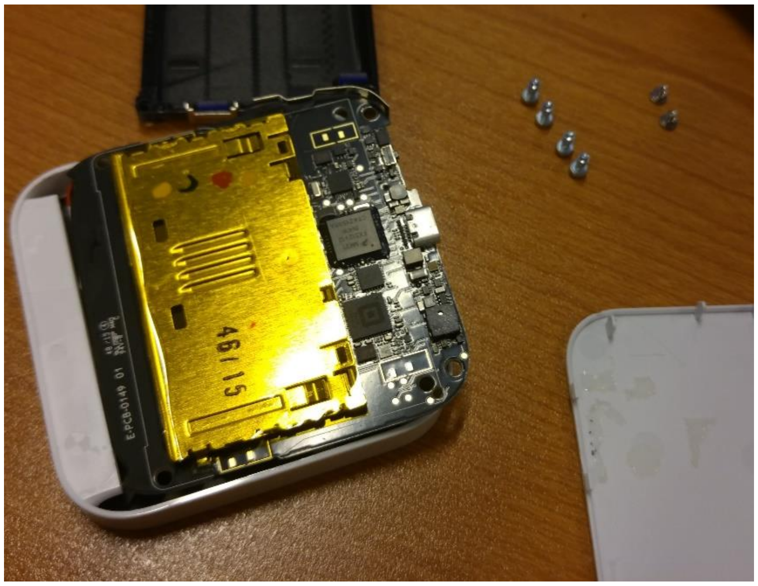


Figure 2: Disassembled Square Chip-And-Pin Reader

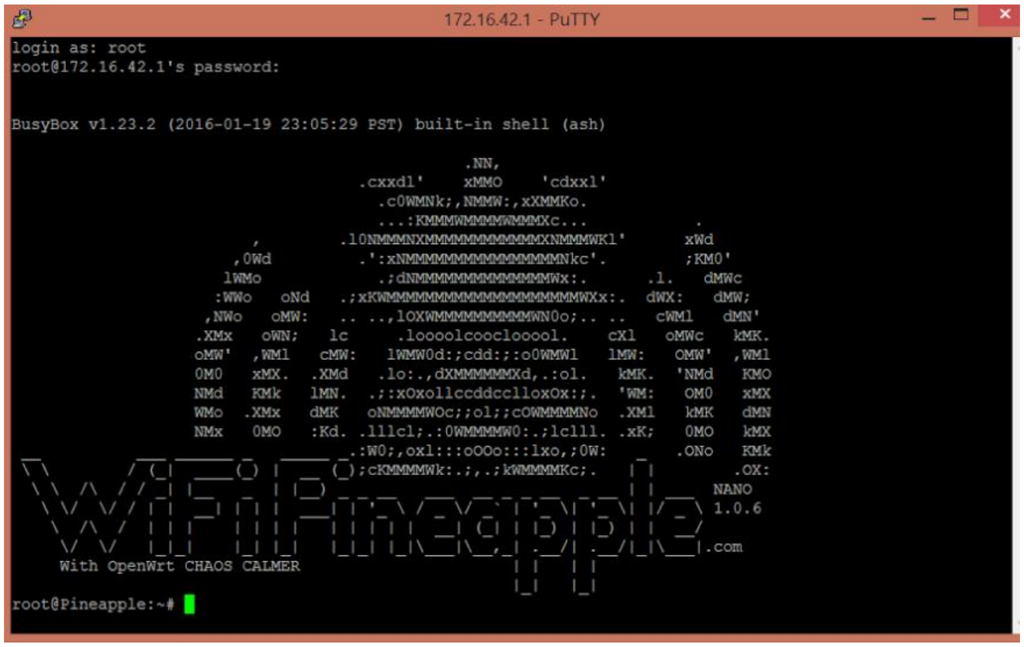
We then went through with a typical Man-In-The-Middle attack using the WiFiPineapple as our collection point. We set up the WiFiPineapple to beacon the SSID of the UHM wireless. We then used an Android phone to connect to the wireless, and then connected the reader to the phone.

Upon trying to open the Square Reader application, we found that the application itself would not load. To get around this, we first opened the application on its normal 3G network, then connected the phone to our fake Wi-Fi connection. Once in transaction mode, however, the Square Reader would again refuse the internet connection. The one transaction that was "completed" while on this network, later showed as "pending," due to "network connectivity issues." We later determined this was due to a bogus certificate that the WiFiPineapple was using.

After figuring out that the certificate was the issue, we decided to create our own certificates and keys. On the WiFiPineapple, we installed OpenSSL, then we ran:

```
openssl genrsa -out certificate.key 4096
openssl req -new -x509 -days 365 -key certificate.key -out certificate.crt
```

to create our own certificate and key using a 4096 bit key size. We then replaced the bogus certificate the WiFiPineapple had with this one and ran the test again. Again, the same network connectivity issues plagued the test. On the Android phone, we decided to manually install the certificate, which would automatically trust any device



```
login as: root
root@172.16.42.1's password:


BusyBox v1.23.2 (2016-01-19 23:05:29 PST) built-in shell (ash)

                                                   .NN,
                                         .cxxdl'    xMMO    'cdxxl'
                                           .c0WMNk;,NMMW:,xXMMKo.
                                           ...:KMMMWMMMMWMMMXc...              .
                          ,              .lONMMMNXMMMMMMMMMMMMXNMMMWKl'       xWd
                        ,0Wd              .':xNMMMMMMMMMMMMMMMMNkc'.          ;KM0'
                       lWMo                 .;dNMMMMMMMMMMMMMMMWx:.         .l.   dMWc
                     :WWo    oNd    .;xKWMMMMMMMMMMMMMMMMMMMMMWXx:.   dWX:    dMW;
                    ,NWo   oMW:    .. ..,lOXWMMMMMMMMMMMWN0o;.. ..    cWMl    dMN'
                   .XMx   oWN;    lc      .loooolcoocloool.     cXl   oMWc    kMK.
                   oMW'  ,WMl   cMW:   lWMW0d:;cdd:;:o0WMWl   lMW:   OMW'   ,WMl
                   0M0   xMX.  .XMd   .lo:.,dXMMMMMMXd,.:ol.   kMK.  'NMd    KMO
                   NMd   KMk   lMN.  .;:xOxollccddcclloxOx:;.  'WM:   OM0    xMX
                   WMo  .XMx   dMK   oNMMMMWOc;;ol;;cOWMMMMNo  .XMl   kMK    dMN
                   NMx   0MO   :Kd. .lllcl;.:0WMMMMW0:.;lclll. .xK;   0MO    kMX
                                      .:W0;,oxl:::oOOo:::lxo,;0W:          .ONo    KMk
                                        ;cKMMMMWk:.;,.;kWMMMMKc;.                  .OX:
                                                                                  NANO
                                                                                  1.0.6

                                                                              .com
     With OpenWrt CHAOS CALMER

root@Pineapple:~#
```

that used said certificate, to bypass this issue. This was only an issue on the Android phone. We did the same test on an iPhone, and there were no issues whatsoever with the certificate.

To overcome the encrypted channel, we used SSLSplit with the certificate we previously generated, and we rerouted traffic from all ports to port 8888. We then started SSLSplit and began logging data on port 8888.

./sslsplit -k certificate.key -c certificate.crt -D -l connections.log -S /tmp/sslsplit/ -L contentlog ssl 0.0.0.0 8888

After setting up the SSLSplit on the WiFiPineapple, we tried to run the test again under the same conditions described above. We were able to successfully complete a transaction while connected to the WiFiPineapple. While everything appeared to work, our logs somehow remained empty.

Figure 3: WiFiPineapple connected via PuTTY

We have determined that the issue is probably in our IPtables script which is in charge of rerouting all traffic to port 8888.

```
#!/bin/sh
echo ‘1’ > /proc/sys/net/ipv4/ip_forward
iptables -X
iptables -F
iptables -t nat -F
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
        iptables -t nat -A PREROUTING -p tcp
-d 168.105.94.204--dport 443 -j REDIRECT --to-
ports 8888

iptables -t nat -A POSTROUTING
-j MASQUERADE
```

The next step is to figure out what the issue is with our script, and to then read logs for the traffic to analyze data.

5.2 Experiment 2

Emulate another malicious merchant scenario where the merchant’s device is on another app that records your credit card information. Then, he pretends as if the card wasn’t read properly, switches applications, then actually makes the transaction.

In the interest of time, we were unable to get to the second experiment. Given more time, we will be able to properly finish up the first experiment, and potentially the second experiment.

6 Conclusion

The Square reader for chip cards is not secure. We have successfully disassembled the device without showing any signs of tampering. We were able to utilize the WiFiPineapple to orchestrate the beginning steps to a man-in-the-middle attack. Despite not being able to read the logs that were snooped from the reader, we can say that a person with malicious intent can go to a convention, for example, and snoop everyone’s credit card information if everyone’s reader device is connected to the convention’s Wi-Fi. Keeping our results and progress in mind, future works will include completing our Wi-FI man-in-the-middle attacks, and doing Bluetooth man-in-the-middle attacks. In theory, this will also be doable via Bluetooth Man-In-The-Middle attacks, which can be conducted from a mile away.

References

[1] Losev, Mellen, Moore. Mobile Point of Scam: Attacking the Square Reader. Boston University.

[2] Anderson, Bond, Choudary, Murdoch, Skorobogatov. Chip and Skim: cloning EMV with the pre-play attack. Computer Laboratory. University of Cambridge.

[3] “Square Sellers Just Made $100M in Sales in One Day” [Online] Available: https://squareup.com/townsquare/100m-day/

[4] Haataja, Keijo, and Pekka Toivanen. "Two Practical Man-In-The-Middle Attacks on Bluetooth Secure Simple Pairing and Countermeasures." IEEEXplore. IEEE, Jan. 2010. Web.