Please ~~insert~~ inject more coins

Defcon XXI

Press start

Me ?

- Nicolas Oberli (aka Balda)
- Swiss security engineer
- CTF enthusiast
- Retro gamer
- Beer drinker / brewer

It all started so simply...

- I wanted to add coin handling to my MAMEcab
- Bought a coin acceptor on an auction site

Coin handling devices

- All kind of machines use coin handling devices
  - ATMs
  - Vending machines
  - Casino game machines
  - ...
- Multiple devices are used in those machines

Coin / Bill acceptors

- Used to count coins and bills
- Can detect coin/bill value
- Detects false coins/bills

Coin hopper

- Used to give coins back to the customer
  - One hopper per coin value
  - Gives back coins one by one

Communication protocols

- Multiple protocols are used to communicate with these devices
  - Parallel
  - Serial (RS232)
  - MDB
  - ccTalk
- The protocols are very vendor-specific
- ccTalk is what we will be talking about

ccTalk ?

- “coin-controls-Talk”
- Semi-proprietary protocol
  - Maintained by Money Controls LLC, England
  - Protocol specs available on cctalk.org
    - Some parts of the specs are only available after signing a NDA :-(

ccTalk ?

- Request / response messages
- RS232-like data transmission
  - Uses only one wire for both sending and receiving
  - 9600 bits/s, 8N1, TTL signals (0 - 5V)
- Each device has its own address on the bus
  - By default 1=controller, 2=coin acceptor

ccTalk message format

- All frames use the same format

  - Header is the actual command sent to the device
    - Header == 0 means it's a response
- Payload length can vary from 0 to 252
  - Data length != packet length
- Checksum is the complement to 0xFF of the packet

ccTalk headers

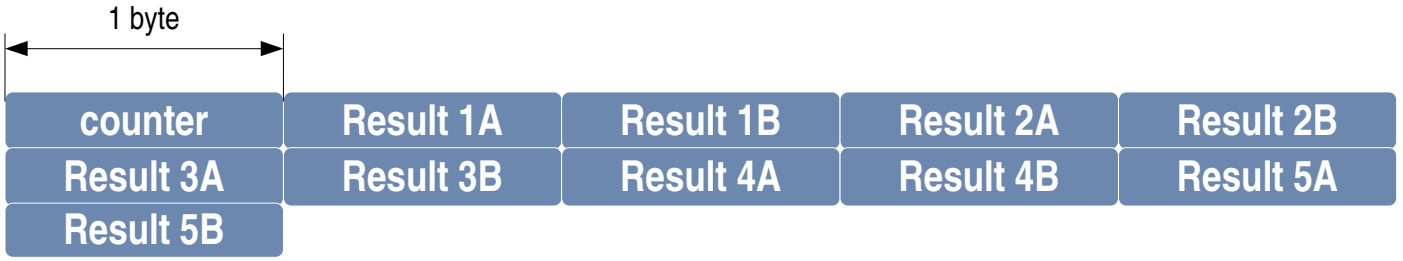
- Each command is assigned a *header*
  - Since its coded in a byte, 256 possible commands
    - From the doc :

Sample communication

- 02 00 01 FE ff
  - Sample poll from @01 to @02
- 01 00 02 00 FD
  - Response from @02 to @01
- 02 00 01 F6 07
  - Request manufacturer ID
- 01 03 02 00 4E 52 49 11
  - Response (length 3) : NRI
    - (ASCII encoded)

Coin acceptor handling

- The controller polls a coin acceptor using header 229
  - The response contains the following payload



- Counter is incremented for each event generated by the acceptor
  - Event counter cycles from 1 to 255

Coin acceptor results

- The last five results are sent in the response
  - Result A contains the validation channel
    - A device can recognize a certain amount of different coins which are organized in channels
    - Either set by the manufacturer or by config
  - Result B contains the error code (Bad coin, mechanical error, ...)
    - Again, the codes are vendor specific
  - Sometimes, results A and B are switched

Initial project

- Implemented the ccTalk protocol to handle a coin acceptor
- Use a Teensy in keyboard mode
  - When a coin is inserted, determine its value and send the corresponding number of keystrokes to MAME

Can we do more ?

- Other vending machines may use other headers and / or functions
- It is difficult to track responses
  - You need to decode the request first
- There is no open source sniffer for ccTalk...

Introducing ccSniff/ccParse

- Python utilities used to sniff data on a ccTalk bus and parse the sniffed data to a readable format
  - Use a ccTalk library developed from scratch
- Can use a bus pirate to sniff
  - It's the best way, since it can handle UART signals correctly

Demo !

```
ccParse 0.2 - 67 messages

<cctalk src=2 dst=1 length=4 header=0 data=01040404>
<cctalk src=1 dst=2 length=0 header=230>
<cctalk src=2 dst=1 length=2 header=0 data=ffff>
<cctalk src=1 dst=2 length=0 header=188>
<cctalk src=2 dst=1 length=1 header=0 data=01>
<cctalk src=1 dst=2 length=0 header=229>
<cctalk src=2 dst=1 length=11 header=0 data=00000000000000000000>


= In response to : Read buffered credit or error codes
<cctalk src=1 dst=2 length=0 header=229>

= Payload decoding
Event Counter : 0
Result A 0 - Result B 0
Result A 0 - Result B 0
Result A 0 - Result B 0
Result A 0 - Result B 0
Result A 0 - Result B 0

= Raw dump of packet
010b020000000000000000000000f2
```

Can we do even more ?

- What if we can inject some data on the bus ?
  - Like telling the controller “Hey ! I'm the coin acceptor and I received a LOT of money !”
- The problem is, we only have one wire for the whole bus
  - Both us and the device receive the request at the same time
  - This means we would answer at the same time and jam the signal

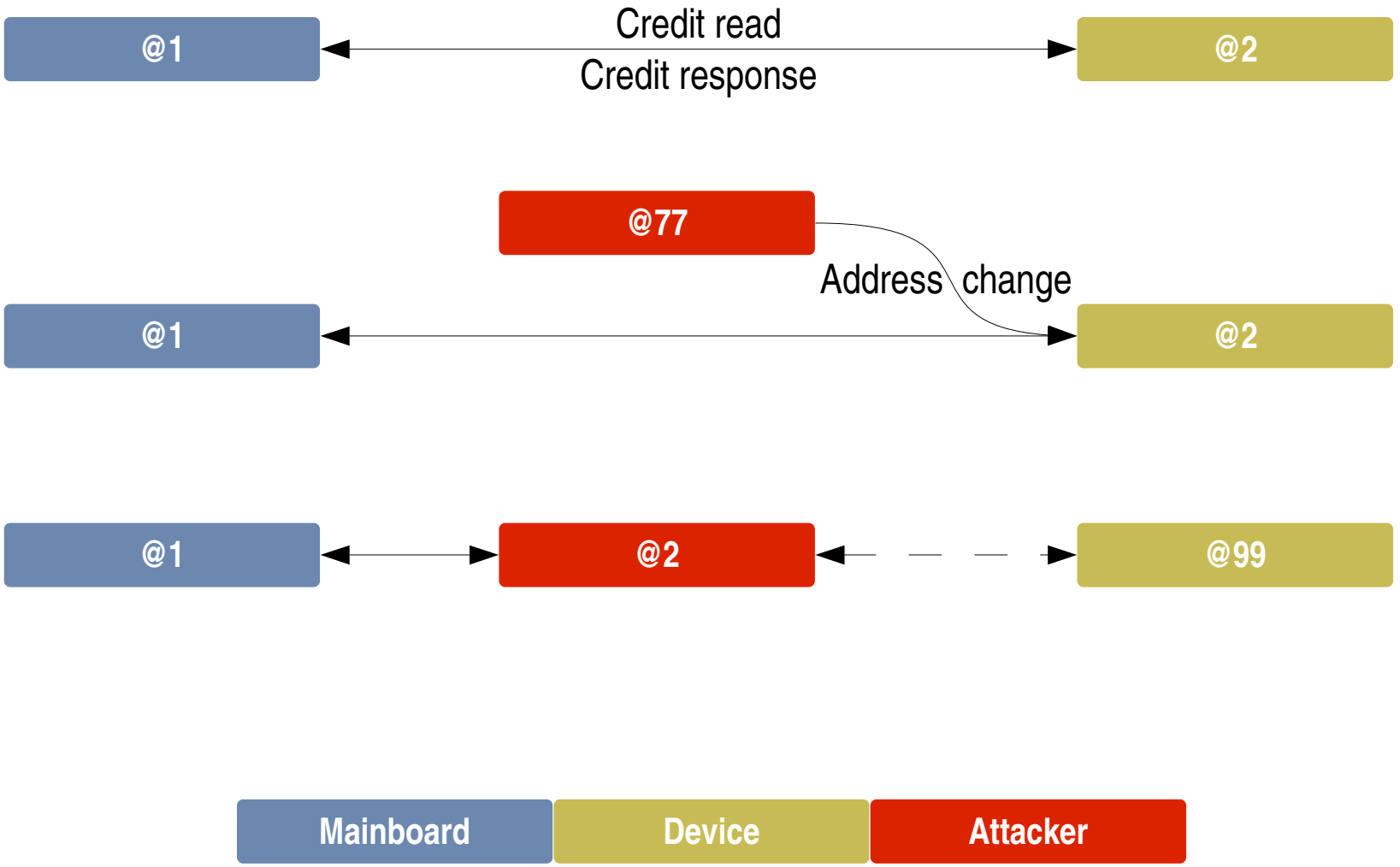
ccTalk multidrop commands

- Used by the controller to resolve addressing conflicts
  - Header 251 – Address change
  - Used by the controller to force a device to change its address in case of conflicts

Device in the middle

- No checks are made to ensure that the request is valid
  - Simply tell the device at address *x* that it needs to change its address to *y*
- Using these requests, we are now able to hijack the device
  - It allows us to intercept all communication between the controller and the device

Injection scheme



Timing

- We need to be sure that we won't jam the current traffic
- At 9600b/s, it takes 1.04ms to send a byte
- Specs indicate that devices need to be polled every 200ms
  - Largely enough time for us

Device hijacking

- To hijack a device on the bus :
  - Scan the bus to search for silence
  - If sufficient periods of silence, prepare injection
  - Craft an address change packet
  - Wait for silence period, then inject packet
  - Respond to requests from the controller
  - When finished, set the device to its original address
- Remember, we need to do this while the bus is in use

Introducing ccJack

- Automates the hijacking process
- Can emulate any device by sniffing the current responses and reply the same
- Can use a bus pirate to sniff and inject

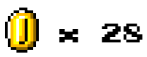
Example : Inject coins !

- Once the coin acceptor is hijacked, just respond by incrementing the counter
  - It is also possible to modify the coin code to increase the value of the injected coin
- Be careful ! The counter must be higher or equal to the last value
  - Any lower value will make the controller throw an error and likely reset itself

More ?

- As the acceptor is “offline”, we can do whatever we want to it
  - Some coin acceptors can be recalibrated by ccTalk
    - Look for headers 201 and 202
    - What if one cent becomes $1 ?
  - The path the coin takes after being accepted can be modified
    - Look for headers 209 and 210
    - What if the new path is the money return ?

Demo !



Hopper handling

- Hoppers follow a special schema to release money (simplified)
  - Controller asks for a challenge (Header 160)
  - Hopper responds with 8 random bytes
  - Controller encodes this challenge and sends the response with the number of coins to release (Header 167)
  - Operation is checked periodically by the controller (Header 166)

Hopper bias

- To simplify these steps, some vendors provide hoppers with no challenge/response support
  - Sometimes, you just need to send the hopper serial number as the response
  - Sometimes...

```
If the hopper Product Code is "SCH2-NOENCRYPT", then the
DISPENSE COINS command still needs an 8-byte code, but the
value of the code does not matter.
```

Grab the money !

- After a hopper is hijacked, just tell it to dispense 0xff coins
  - Will only work if the hopper does not use the challenge/response method
- Better : Use the “Purge hopper” command (Header 121)
  - Does not take any challenge/response
  - Hardly ever implemented in practice

Isn't there any protection ?

- Some devices only respond after having been provided a PIN code
  - Only for a subset of commands
    - Depends on the device / firmware / vendor
  - Well, just wait for the PIN to be sent by the controller
    - Check for header 218
  - We can “help” it by pulling the power cord
  - It *could be possible* that the PIN code is the same for a vending machine model

Encryption

- In later versions of the specs, the ccTalk payload and headers can be encrypted
  - Two encryption methods available
    - Proprietary encryption – 24 bit key
    - DES encryption – 56 bit key
  - Use a pre-shared key between the controller and the devices
- Encryption uses different headers
  - Header 229 vs header 112
  - Still possible to get values from the “unencrypted” header

Future – Research fields

- More things to discover on the protocol
  - Encryption support seems suspicious
    - Keys can be transferred using ccTalk
    - Proprietary and closed-source encryption could be weak
  - Some devices accept dumping their internal memory by ccTalk
    - Maybe there are vulns in the firmwares ?
  - It is possible to upload a new firmware to the devices using ccTalk
    - Evilgrade ccTalk edition ?

Conclusions

- Specific protocols can be fun to analyze
  - You never know where you can find exotic protocols
- ccTalk definitely needs more attention
  - Since it transports money-related information, there are interesting applications
- If you don't have one, get a bus pirate
  - It's pure awesomeness !

Availability

- ccTools available on my GitHub account
  - https://github.com/Baldanos/ccTools
- More information about ccTalk after Defcon on my website
  - http://www.balda.ch

Many thanks !

Any questions ?

@Baldanos
http://www.balda.ch

Did I mention I LOVE beer ?