black hat USA 2023

AUGUST 9-10, 2023

BRIEFINGS

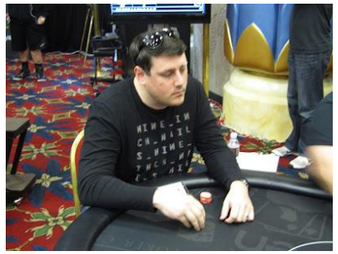
Shuffle Up and Deal

Auditing the Security of Automated Card Shufflers

♠ Joseph Tartaro ♡ Enrique Nissim ♣ Ethan Shackelford ♢

IOActive.

Introduction



Joseph Tartaro

Enrique Nissim

Ethan Shackelford

Embedded Security Consultants at **IOActive**

- Low-level code review
- Reverse engineering (Operating Systems, Drivers, Firmware)
- Specialized tooling development

What and Why?

- Hustler Live Cheating Scandal
  - Suspicious play occurs with accusations of cheating
  - Independent investigators hired

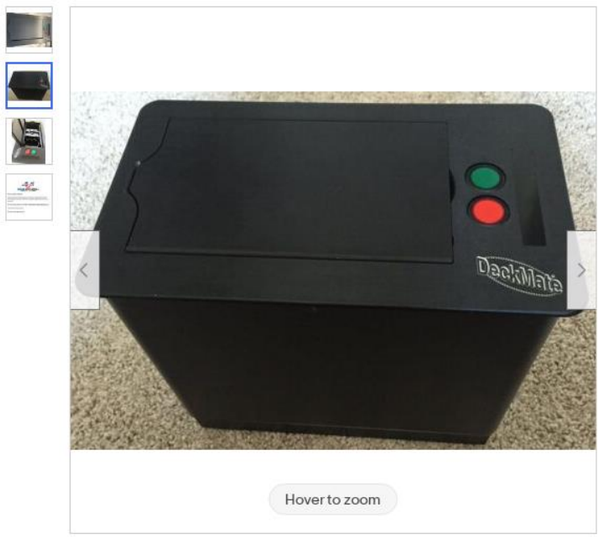
What and Why?

- Investigators Focus Areas

Table 1: Examined Areas

Potential Attack Vectors	Estimated Priority	Estimated Complexity
Table	Low	Complex
RFID	None	Highly Complex
Card Shuffler	Low	Highly Complex
Production Booth and Operations	High	Easy
Network, PC Workstations, And Systems	High	Easy
Communications	Medium	Complex

What and Why?



$ Have one to sell? Sell now

ShuffleMaster Card Shuffler Deckmate One / Deckmate 1 Shuffler

Condition: Used
"fully reconditioned, working like brand new."

Quantity: 1 3 available / 3 sold

Price: US $6,000.00
$289 for 24 months with PayPal Credit*

Buy It Now
Add to cart
♡ Add to Watchlist

Ships from United States | 7 watchers

Shipping: FREE Standard Shipping | See details
Located in: Los Angeles, California, United States

Delivery: Estimated between Tue, Oct 18 and Thu, Oct 20 to 90274

Returns: Seller does not accept returns | See details

Payments: PayPal VISA MC AMEX DISCOVER

PayPal CREDIT
*$288.04 for 24 months. Minimum purchase required. |
See terms and apply now

Earn up to 5x points when you use your eBay Mastercard®.
Learn more

ebay Shop by category Search for anything All Categories

Back to search results | Listed in category: Collectibles > Casino Collectibles > Collectible Casino Card Shufflers

$ Have one to sell? Sell now

ShuffleMaster Card Shuffler Deckmate Two / Deckmate 2 Shuffler

Condition: Used
"fully reconditioned, working like brand new."

Quantity: 1 3 available

Price: US $8,500.00
$409 for 24 months with PayPal Credit*

Buy It Now
Add to cart
♡ Add to Watchlist

Ships from United States | 6 watchers

Shipping: FREE Standard Shipping | See details
Located in: Los Angeles, California, United States

Delivery: Estimated between Mon, Oct 17 and Thu, Oct 20 to 90274

Returns: Seller does not accept returns | See details

Payments: PayPal VISA MC AMEX DISCOVER

PayPal CREDIT
*$408.06 for 24 months. Minimum purchase required. |
See terms and apply now

Earn up to 5x points when you use your eBay Mastercard®.
Learn more

What and Why?

- ShuffleMaster Deck Mate Series
- Most popular automated shufflers
- Used across the world in casinos, card rooms and home games
- Official shuffler of the World Series of Poker (WSOP)

What and Why?

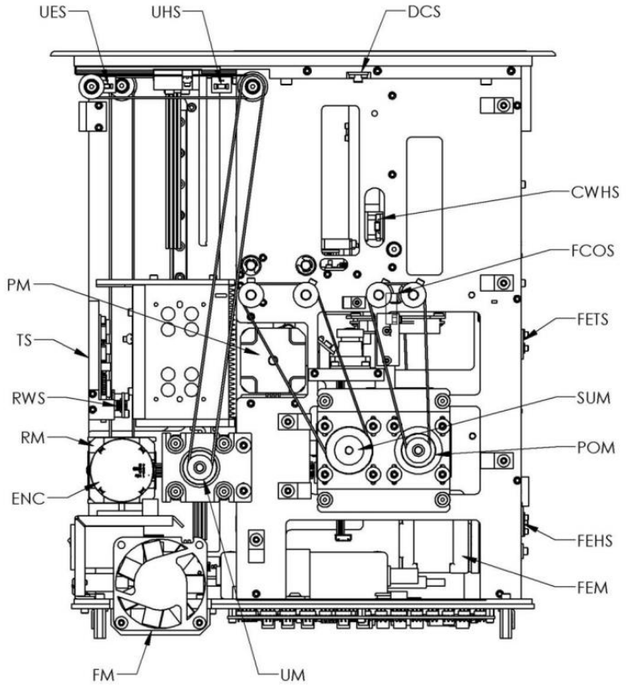
Deck Mate 1	Deck Mate 2
	
• Single Deck Shuffler • Detects missing / additional cards	• Single Deck Shuffler • Detects missing / additional cards (w/ details) • Shuffles significantly faster than DM1 • Supports remote management via network • Supports external display module • Player clock feature

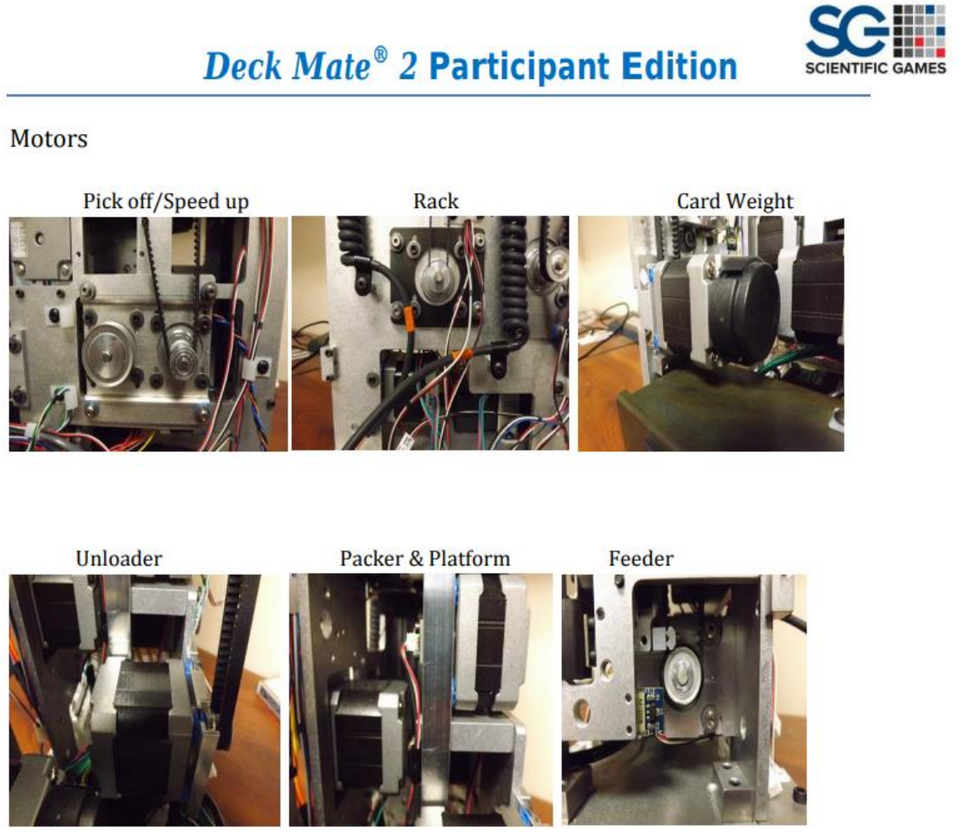
Demo

Attack Scenarios

Maintenance Employees

- Extremely complex
- Contains
  - Rubber belts
  - Sensors
  - Motors
- Requires
  - Regular maintenance
  - Contractual service agreements





Gaming Operator Employees

- Casino Employees / Device Operators
- Unrestricted access to shufflers
- Access to exposed external ports

- Manager/Operator
- Dealer
- Chip Runner
- Security
- ...

Attacker at Poker Table (DM2)

- Shuffler cutouts in table
- External interfaces exposed to players
- Ethernet, USB, Power

Attacker with Network Access (DM2)

- Various network services
- Unnecessary attack surface

```
# Nmap 7.92 scan initiated Tue Nov  8 18:16:17 2022 as: nmap -sV -p- --open -oA nmap-deckmate2 169.254.0.1
Nmap scan report for 169.254.0.1
Host is up (0.026s latency).
Not shown: 65529 closed tcp ports (conn-refused)
PORT     STATE SERVICE     VERSION
22/tcp   open  ssh         Dropbear sshd 0.53.1 (protocol 2.0)
23/tcp   open  telnet      BusyBox telnetd
80/tcp   open  http        lighttpd
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
6000/tcp open  X11         (access denied)
Service Info: Host: shuffler; OSs: Linux, Unix; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Tue Nov  8 18:16:35 2022 -- 1 IP address (1 host up) scanned in 18.73 seconds
```

Attacker with Cellular Network Access (DM2)

- Documents identified during research suggest the cellular modem can be used for pay-per-shuffle rental of shufflers
- No firewall or network or iptables rules prevent Ethernet/USB network services from also being exposed on the cellular interface

```
<modem set='1' >
        <connect>
                <string1 set='AT+CGDCONT=1,"IP","[redacted]","0.0.0.0"' >
                        <wait-for set='OK' />
                </string1>
                <string2 set='AT&K0' >
                        <wait-for set='OK' />
                </string2>
                <string3 set='AT#SGACT=1,1' >
                        <wait-for set='' />
                </string3>
                <string4 set='AT#SD=1,0,80,10.255.255.4,0,0,0' >
                        <wait-for set='OK' />
                </string4>
        </connect>
        <disconnect>
                <string1 set='+++' >
                        <wait-for set='OK' />
                </string1>
                <string2 set='AT#SH=1' >
```

Enabling Pay-Per-Shuffle

When *ShuffleFlex* "Pay-Per-Shuffle" will be used, the feature must be manually enabled.

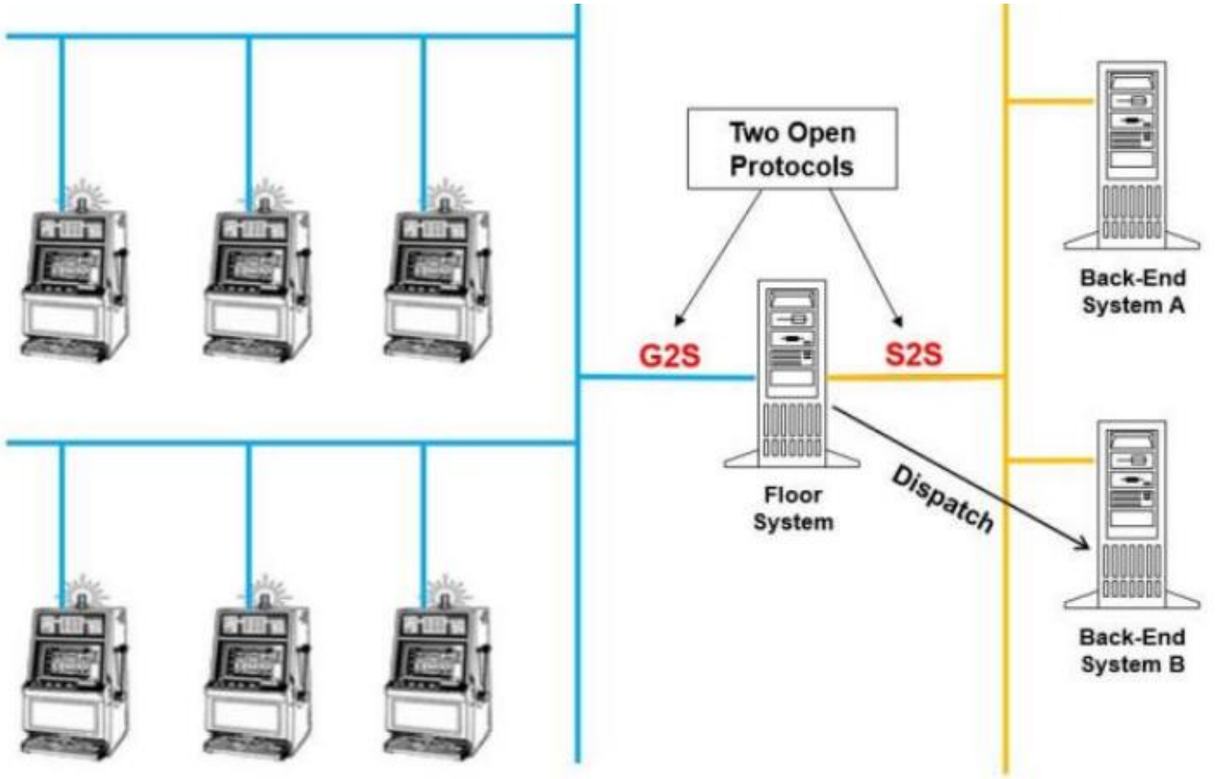
If the shuffler is equipped with the *ShuffleFlex* conversion kit and when power is turned ON for the first time after software installation, the operator is given the option to select whether the Pay-Per-Shuffle feature will be enabled.

Refer to Section 3.1, 'Enabling *Shuffle Flex* Operation in the *Shuffle Flex* Conversion & Setup Instructions manual for further details.

Casino Architecture and Standards

Modern Casino Floor

- The International Gaming Standards Association (iGSA) is the entity responsible for the standards implemented across the gaming industry.
- Different types of specifications
  - Communication
  - Regulatory
- G2S
- S2S



IGSA Unleash the Power of Your Floor, 3rd edition

Gaming to System (G2S)

- Standardizes communications between gaming devices and management systems
- Asynchronous XML based messages
- TCP (with optional SSL) and other IP protocols for transport
- P2P and Multicast

Message

```
<g2sMessage>                                    // Message Level
    <g2sBody>
        <anyClass>                              // Class Level
            <anyCommand />                      // Application Level
        </anyClass>
        <anyClass>                              // Class Level
            <anyCommand />                      // Application Level
        </anyClass>
    </g2sBody>
</g2sMessage>
```

Acknowledgement

```
<g2sMessage>                                    // Message Level
    <g2sAck />
</g2sMessage>
```

G2S Classes

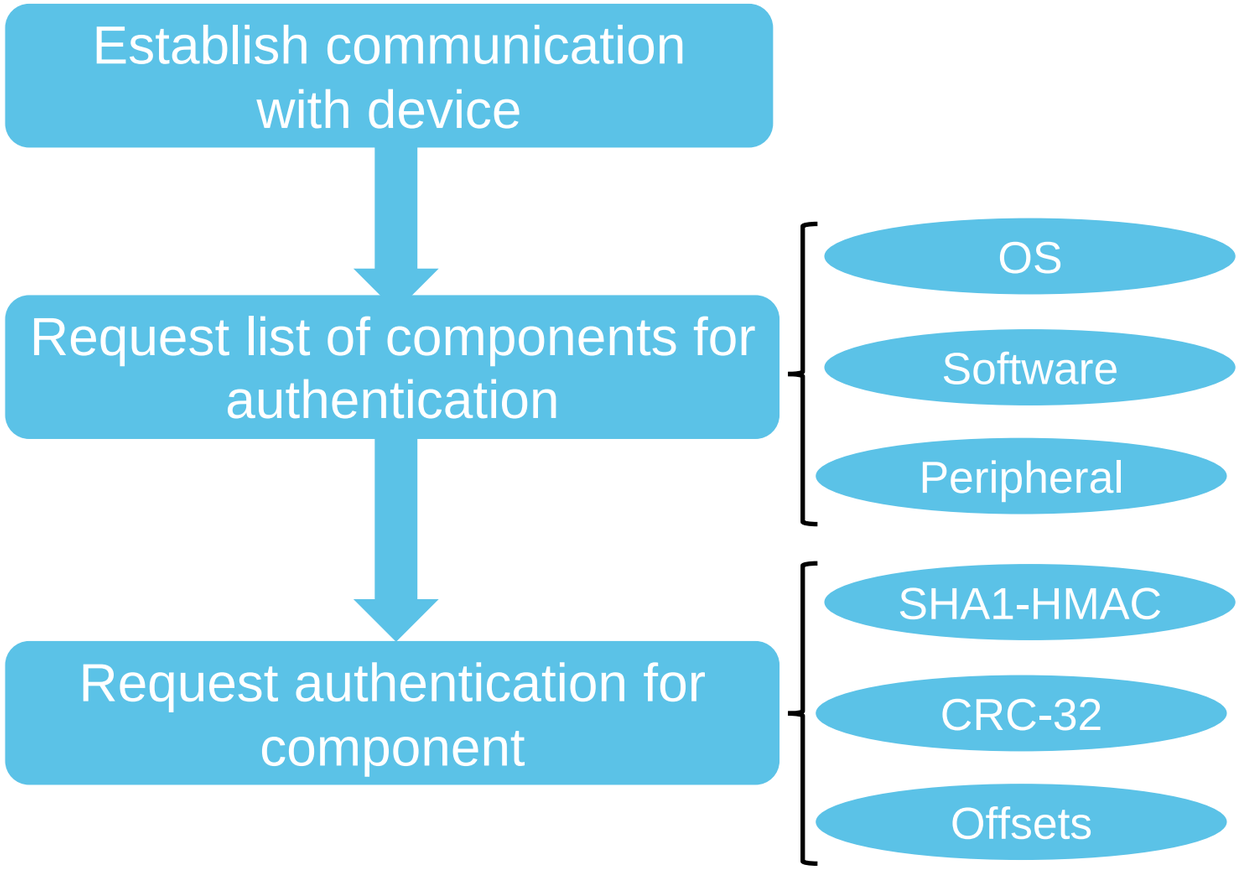
G2S define classes of functionality a device can implement

These classes relate to specific functions or features of the EGM, e.g. meters, cabinet, jackpots, vouchers, etc.

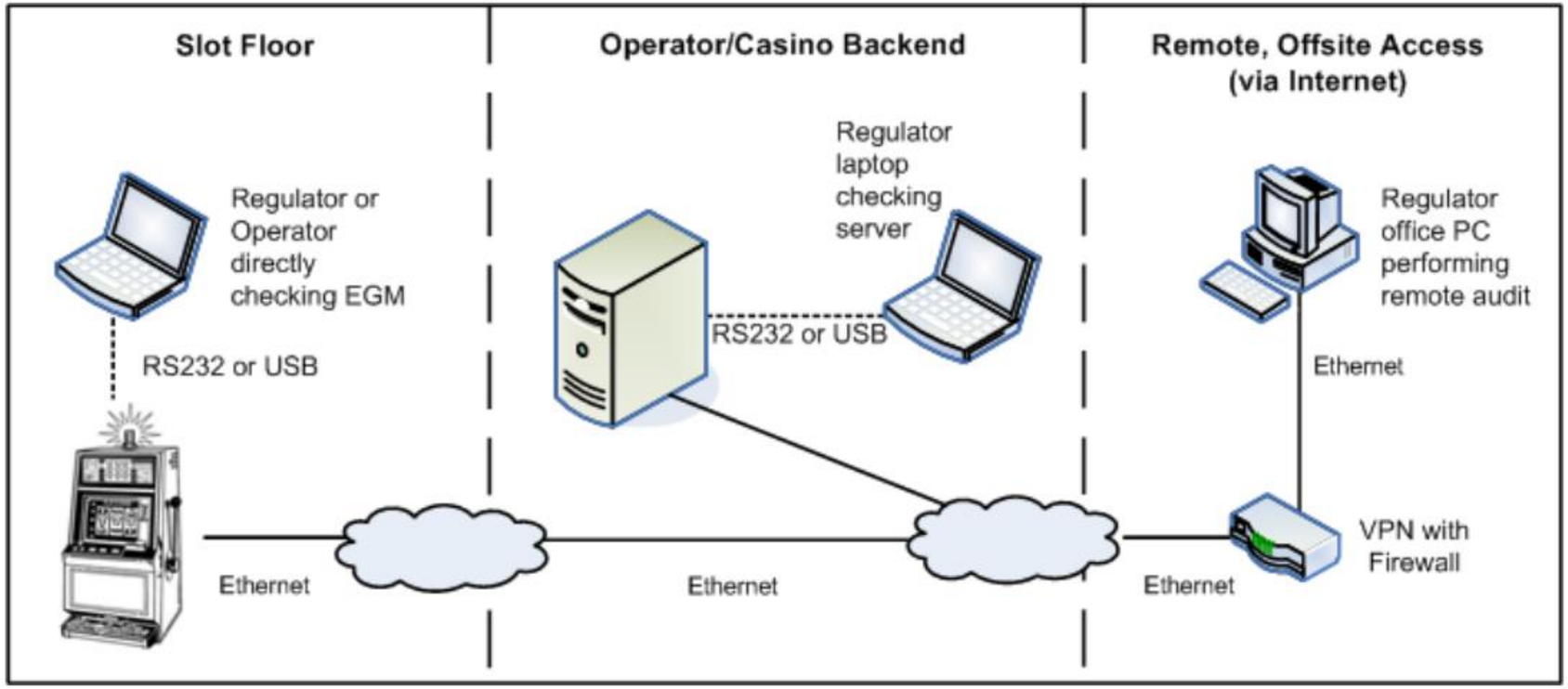
- *communication*
- *cabinet*
- *eventHandler*
- *meters*
- *gamePlay*
- *deviceConfig*
- *printer*
- *progressive*
- *idReader*
- *bonus*
- *hopper*
- *noteDispense*
- *optionConfig*
- *download*
- *handpay*
- *coinAcceptor*
- *noteAcceptor*
- *commConfig*
- *player*
- *voucher*
- *wat*
- *gat*
- *central*

Game Authentication Terminal

- This class provides a set of commands that regulators can use retrieve errors logs and authenticate EGMs and peripherals
  - Serial GAT
  - Network GAT
- Permits ensuring the software running on devices has not been modified
- GAT does not define or require a particular authentication algorithm



Serial GAT and Network GAT



GAT Protocol

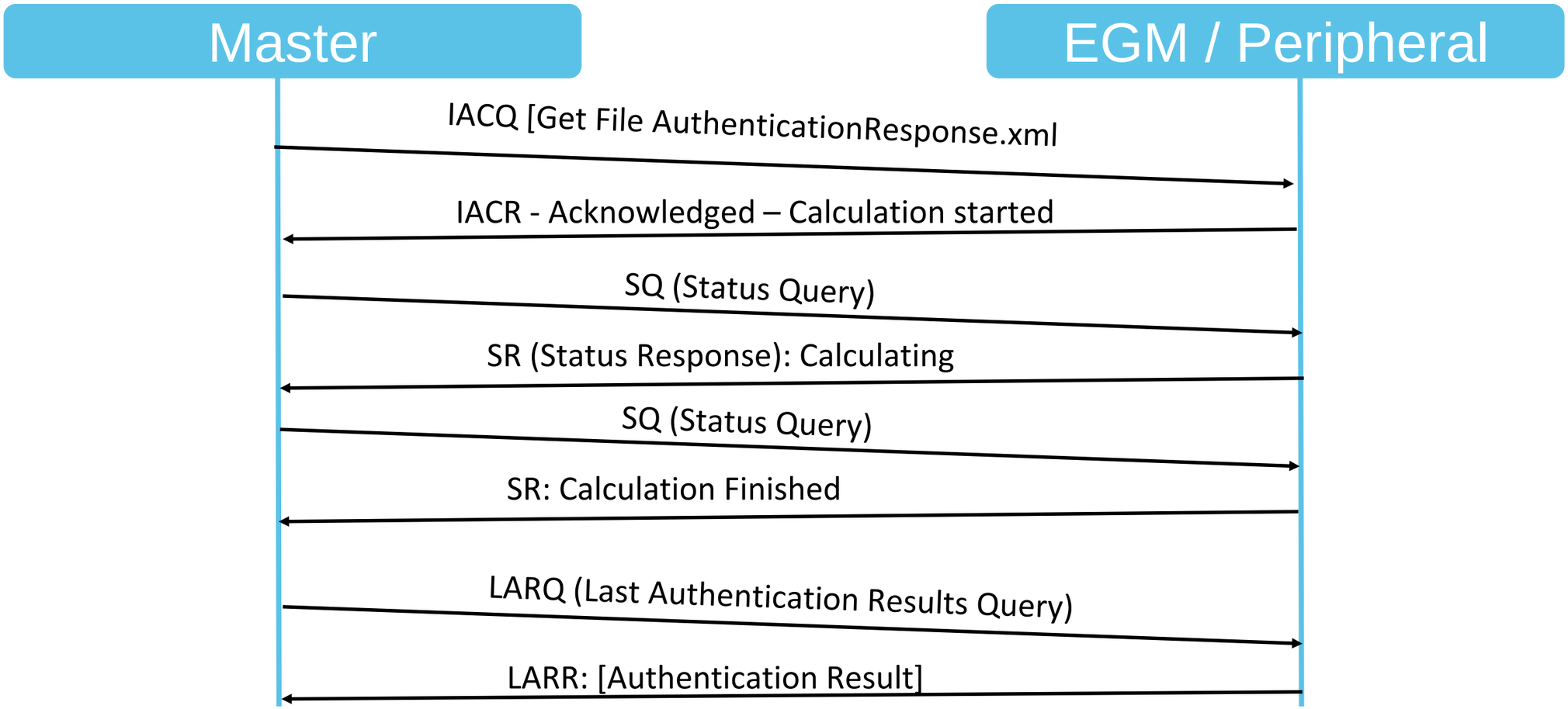
Application Layer

Command	Length	Message Data	CRC
1 Byte	1 Byte	0 - 251 Bytes	2 Bytes

Commands in GAT

Request	Description	Response	Description
0x01 SQ	Status Query	0x81 SR	Status Response
0x02 LASQ	Last Authentication Status Query	0x82 LASR	Last Authentication Status Response
0x03 LARQ	Last Authentication Results Query	0x83 LARR	Last Authentication Results Response
0x04 IACQ	Initiate Authentication Calculation Query	0x84 IACR	Initiate Authentication Calculation Response

GAT – IACQ Get File



GAT Requires Transaction Logs to be G2S compliant

23.2 Transaction Logs

Within the gat class, the EGM MUST store critical data related to GAT accesses and responses in persistent memory. This data is designed to provide an audit trail of all actions related to GAT devices. Log entries MUST be generated for GAT actions initiated by devices local to the EGM, such as through an RS232 connection, as well as GAT actions initiated by host systems. See Chapter 1 for more details on transaction logs.

GAT Security

- The GAT authentication is inherently flawed: relies on the response the EGM or Peripheral hands it (which could be compromised)
- There is no mention of Public Key Infrastructure in the G2S and GAT specifications
- The algorithms defined for authentication are cryptographically weak or not suitable for cryptographic purposes: HMAC-SHA1, CRC16, CRC32
- The HMAC-SHA1 algorithm provide some randomness to the process, but nothing more

Shufflers and GAT

DM1

- It does not implement any GAT concept
- Modified firmware cannot be easily detected
- It features "History Logs" but these are not G2S Transaction Logs

```
Deck Mate Shuffler History Log.
Serial Number: 108292
Report T:10:02:39
Report D:04-29-00
Current Gripper Offset:203
Current Platform Offset:189

Powered Up    :  Time:06:58:23,Date:04-24-18,P:243,G:224,Cyc:2982
Auto Setups. :  Time:23:18:03,Date:01-11-00,P:96,G:245,Cyc:0
Powered Up    :  Time:23:19:45,Date:01-11-00,P:250,G:100,Cyc:0
Extra Card.   :  Time:23:20:26,Date:01-11-00,P:270,G:100,Cyc:0
Jam Recovery:  Time:17:12:45,Date:08-14-15,P:0,G:246,Cyc:0
Powered Up    :  Time:23:40:52,Date:01-30-00,P:82,G:246,Cyc:24
Auto Setups. :  Time:00:04:46,Date:01-31-00,P:189,G:203,Cyc:24
Powered Up    :  Time:10:07:13,Date:02-11-00,P:189,G:203,Cyc:36
Powered Up    :  Time:02:03:56,Date:02-13-00,P:189,G:203,Cyc:37
Door Opened. :  Time:07:49:54,Date:02-13-00,P:209,G:203,Cyc:190
Missing Card.:  Time:07:54:27,Date:02-27-00,P:199,G:203,Cyc:1052
Power Ups = 89
CDS Failures:0
Missing Card  = 201
Extra Card  = 4
Door Open Jams = 231
Jam Recoveries = 2
Auto Setups = 11
Platform OOP = 631
Total Cycles:1056
```

DM2

- It features a HMAC-SHA1 Authentication
- Serial GAT only
- No transaction records of GAT accesses

Get Special Functions Result

```
Feature: Get File
  Parameter: AuthenticationResponse.xml
  Parameter: %%SHA1_HMAC%%

Feature: Component
  Parameter: DeckMate2_UI_2.0.254
  Parameter: %%SHA1_HMAC%%

Feature: Component
  Parameter: DeckMate2_CardRec_5.0.023
  Parameter: %%SHA1_HMAC%%

Feature: Component
  Parameter: DeckMate2_NXP_NXP 1.0.172
  Parameter: %%SHA1_HMAC%%

Feature: Component
  Parameter: DeckMate2_Games_1.0.095
  Parameter: %%SHA1_HMAC%%
```

IACQ Get File AuthenticationResponse.xml

```
<?xml version='1.0'?>
<Components GatExec='default'>
 <Game>
  <Name>Deck Mate 2</Name>
  <Manufacturer>Bally Technologies</Manufacturer>
  <Component>
   <Name>DeckMate2_UI_2.0.254</Name>
   <Checksum>32A66E4EAFB35AE6DC51268104111118377EE254</Checksum>
  </Component>
  <Component>
   <Name>DeckMate2_CardRec_5.0.023</Name>
   <Checksum>2EA4D5A836604B7676D7C7EB3EA8BEC84B410903</Checksum>
  </Component>
  <Component>
   <Name>DeckMate2_Games_1.0.095</Name>
   <Checksum>39FF668A8BEE8B19E72A7DED15F4B043A1D2DE2B</Checksum>
  </Component>
  <Component>
   <Name>DeckMate2_NXP_NXP 1.0.172</Name>
   <Checksum>97285778B41B5CDFD151A4A9702DA8D32A839AF9</Checksum>
  </Component>
 </Game>
</Components>
```

Deck Mate 1

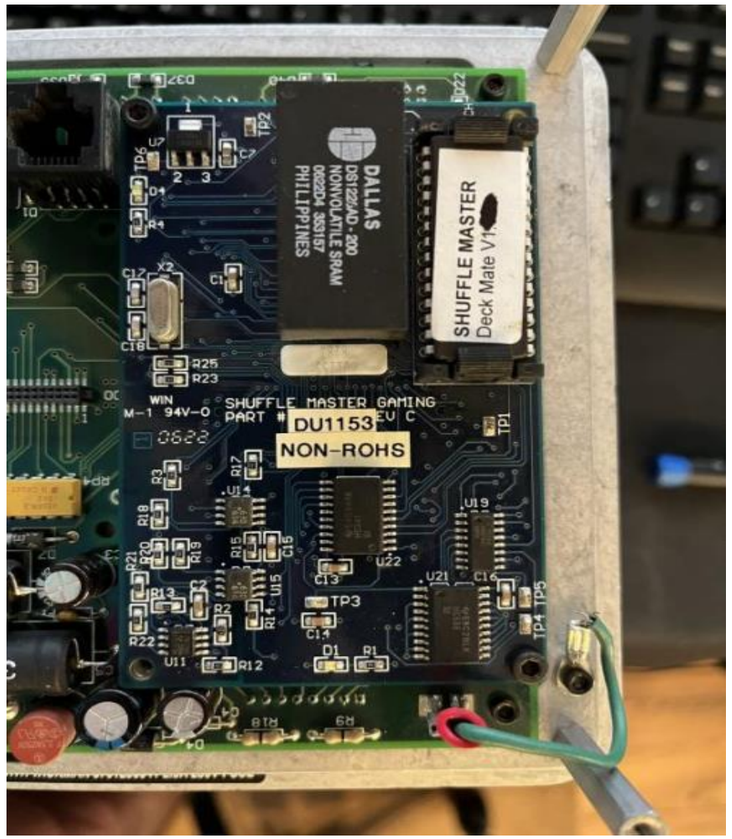

Push GREEN Button
To Start


Load Cards
Poker
(52 Cards)


Model: Deck Mate
S/I
AC~100/120/200-240 V 50/60 Hz
0.8/0.5 A
c UL us
LISTED
Warning: Shock Hazard
Do Not Open
Warning: For continued
protection against risk
of fire, replace only
with Bussmann fuse
type GDC (S506) 2A
250V time delay.
T2AL250V

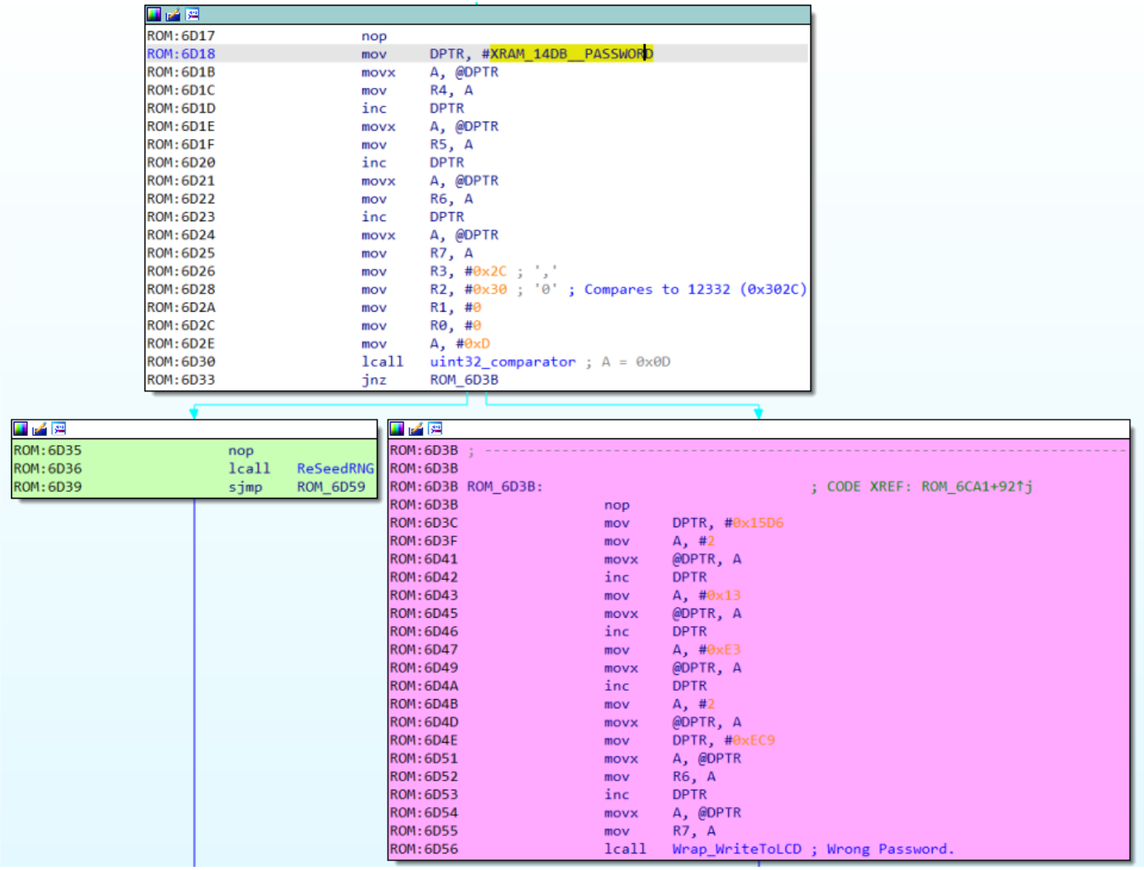
Reverse Engineering DM1

- Goals: understand operation, RNG and shuffling algorithm
- The ROM code was extracted from the M27C512 EEPROM
- The MCU is AT89S53 (Intel 8051). Old 8-bit architecture. Fun to reverse
- Bare Metal. No symbols, no debug information



Setup Menu

- Set Game Type
  - Poker
  - Blackjack single deck
  - Blackjack double deck
- Set number of cards
- Set time
- Set date
- Configure delay after platform drop
- Read serial number
- Read total cycles
- Read reset cycles
- Reset history logs
- Re-Seed RNG



Timer Interrupt Setup

- Shuffler Xtal is 11.0592 MHz
- Configures 8051 Timer0 to Mode1 (16-bit mode)
- Sets TH0|TL0 to 0xFF1E, to interrupt every ~245us
- A TIMER_TICK variable is incremented on each interrupt

```
ROM:CA98 timint0_0:                              ; CODE XREF: timint0↑j
ROM:CA98                push    ACC              ; Accumulator
ROM:CA9A                push    DP0H             ; Data Pointer High Byte
ROM:CA9C                push    DP0L             ; Data Pointer Low Byte
ROM:CA9E                nop
ROM:CA9F                clr     ET0              ; Interrupt Enable Register 0
ROM:CAA1                nop
ROM:CAA2                mov     TL0, #0x1E       ; Timer 0 Low Byte
ROM:CAA5                nop
ROM:CAA6                mov     TH0, #0xFF       ; XTAL = 11.0592 * 10**6 (11.0592 MHz)
ROM:CAA6                                         ; Standard Mode: 12 clock cycles per instruction
ROM:CAA6                                         ; Clock = XTAL / 12 => 921600.0 = 921.6 KHz
ROM:CAA6                                         ; 1/Clock = 1.08506 microseconds per tick
ROM:CAA6                                         ;
ROM:CAA6                                         ; Clock Interrupt => 1.08506 * ((0xFFFF + 1) - 0xFF1E) = 245.22 microseconds
ROM:CAA9                nop
ROM:CAAA                setb    RAM_26.2
ROM:CAAC                nop
ROM:CAAD                jnb     P3.2, ROM_CAB7   ; Port 3
```

```
ROM:CAB0                nop
ROM:CAB1                mov     DPTR, #XRAM_13D2
ROM:CAB4                mov     A, #1
ROM:CAB6                movx    @DPTR, A
```

```
ROM:CAB7
ROM:CAB7 ROM_CAB7:                               ; CODE XREF: timint0_0+15↑j
ROM:CAB7                nop
ROM:CAB8                mov     DPTR, #XRAM_151_TIMER_TICK ; TICK is increased as part of timer interrupt
ROM:CABB                movx    A, @DPTR
ROM:CABC                inc     A
ROM:CABD                movx    @DPTR, A
```

RNG

```
void reseed_rng() {
  UINT32 *seed = XRAM_014Dh;
  *seed = 0;
  for (int i = 0; i < 4; i++ ) {
   // Wait for green button input
   BYTE timer_count = XRAM_151_TimerTick;
   *seed = *seed | (((UINT32) timer_count) << 8 * i);
  }
}
```

GenerateRandomDeck()

↓

GetRandom(min, max)

↓

GetNextSeed()

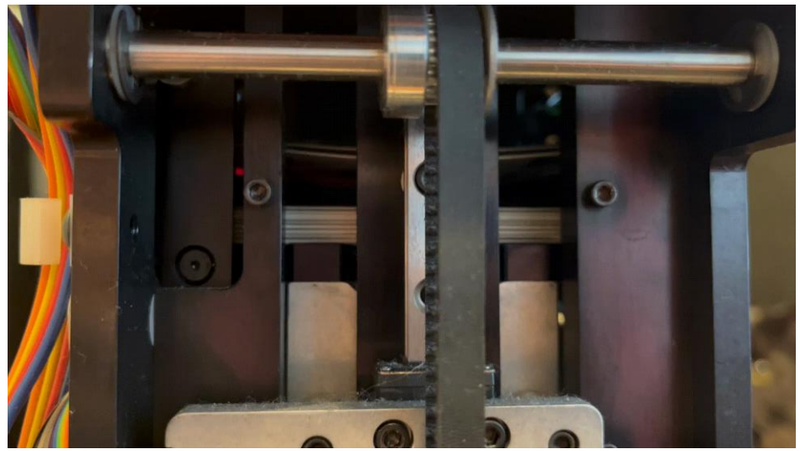
Seed = 0x19660d * seed + 0x3c6ef35f

```
ROM:5374 GetNextSeed:                            ; CODE XREF: GenerateRandomDeck+183↑p
ROM:5374                                         ; GetRandom+98↓p ...
ROM:5374                 nop
ROM:5375                 mov     DPTR, #0x14D
ROM:5378                 movx    A, @DPTR
ROM:5379                 mov     R4, A
ROM:537A                 inc     DPTR
ROM:537B                 movx    A, @DPTR
ROM:537C                 mov     R5, A
ROM:537D                 inc     DPTR
ROM:537E                 movx    A, @DPTR
ROM:537F                 mov     R6, A
ROM:5380                 inc     DPTR            ; load current SEED into R4R5R6R7
ROM:5381                 movx    A, @DPTR
ROM:5382                 mov     R7, A
ROM:5383                 mov     R3, #0xD
ROM:5385                 mov     R2, #0x66 ; 'f' ; Linear Congruential Generator
ROM:5387                 mov     R1, #0x19
ROM:5389                 mov     R0, #0
ROM:538B                 lcall   uint32_mul      ; multiplies two uint32
ROM:538E                 mov     R3, #0x5F ; '_'
ROM:5390                 mov     R2, #0xF3
ROM:5392                 mov     R1, #0x6E ; 'n'
ROM:5394                 mov     R0, #0x3C ; '<'
ROM:5396                 lcall   uint32_addition
ROM:5399                 mov     DPTR, #0x14D
ROM:539C                 mov     A, R4           ; Update SEED at 0x14D
ROM:539D                 movx    @DPTR, A
```

Shuffling Algorithm

1. Cards are physically loaded into the first compartment
2. Based on the configured game settings, the algorithm expects a specific number of cards. For Poker, this number is 52
3. A new deck configuration is randomly generated.
   a. This is represented by an array of numbered positions.
   b. This also indicates how many cards the set of grippers should grip at each step
4. Shuffling starts => the deck configuration is "executed". Cards are placed into the correct location one at a time starting from the bottom of the deck
5. Upon error-free completion, the shuffled deck becomes available

```
[n3k@thanatos shuffler]$ ./GenerateDeck
Randomized Deck:
34 48 25 37 49 43 11 24 28 01 12 16 19 51 10 21 31 45 27 38 08 33 20 14 09 07
46 04 35 32 00 26 06 23 41 18 39 03 13 44 30 15 17 42 02 22 40 47 05 50 36 29
Grips to perform:
00 01 00 02 04 03 00 01 03 00 02 03 04 13 01 06 10 14 09 14 01 13 07 05 02 01
23 01 20 18 00 16 03 15 28 12 29 02 11 34 24 13 15 36 02 21 37 44 05 48 35 29
```



Cheating with DM1

Due to the limitations in the hardware architecture of the DM1, if a bad actor has internal access to the device, they can flash or replace the EEPROM chip and the MCU will simply execute the code.

AT89S53 MCU do not support secure boot

DM1 does not support GAT => there is no trivial way for an auditor to detect a modified EEPROM.

Bypassing Card Count Detection

The DM1 keeps track of the number of cards that were fed into the shuffling compartment.

This permits the detection of missing or extra cards.

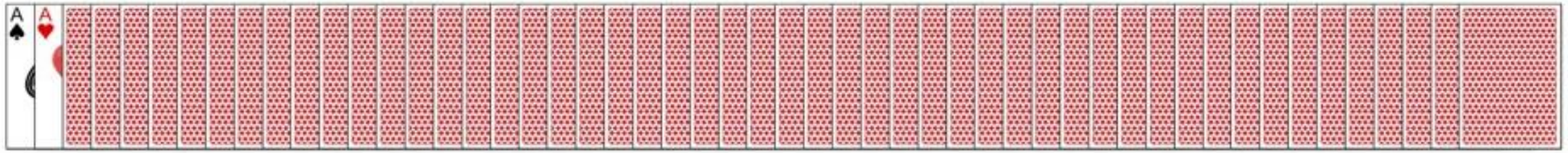
By manipulating the firmware, an attacker can alter the code logic to avoid
failing when too few or too many cards are processed.

This would allow an attacker at the poker table to keep an ace back (hidden up their sleeve) and the dealer would shuffle a deck of only 51 cards without being alerted

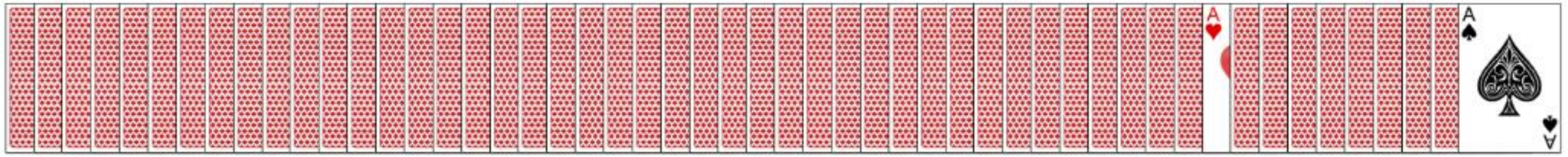
```
ROM:3EFA                movx    @DPTR, A
ROM:3EFB                mov     DPTR, #XRAM_13D1_CARDS_IN_THE_GAME
ROM:3EFE                movx    A, @DPTR
ROM:3EFF                mov     R7, A
ROM:3F00                mov     R6, #0
ROM:3F02                mov     DPTR, #XRAM_F3A__PROCESSED_CARDS
ROM:3F05                movx    A, @DPTR
ROM:3F06                mov     R3, A
ROM:3F07                clr     A
ROM:3F08                mov     R2, A
ROM:3F09                mov     A, R7
ROM:3F0A                clr     C
ROM:3F0B                subb    A, R3
ROM:3F0C                mov     R7, A
ROM:3F0D                mov     A, R6
ROM:3F0E                subb    A, R2
ROM:3F0F                mov     DPTR, #0x161F
ROM:3F12                movx    @DPTR, A
ROM:3F13                inc     DPTR
ROM:3F14                mov     A, R7
ROM:3F15                movx    @DPTR, A
ROM:3F16                mov     R5, #1
ROM:3F18                mov     R6, #0x14
ROM:3F1A                mov     R7, #0x92
ROM:3F1C                lcall   WriteOutSerial2 ; Missing %ld Cards
ROM:3F1F                nop
ROM:3F20                mov     DPTR, #0x15D6
ROM:3F23                mov     A, #1
ROM:3F25                movx    @DPTR, A
ROM:3F26                inc     DPTR
ROM:3F27                mov     A, #0x14
ROM:3F29                movx    @DPTR, A
ROM:3F2A                inc     DPTR
ROM:3F2B                mov     A, #0x92
ROM:3F2D                movx    @DPTR, A
ROM:3F2E                inc     DPTR
ROM:3F2F                mov     A, #1
ROM:3F31                movx    @DPTR, A
ROM:3F32                mov     DPTR, #0xEC9
ROM:3F35                movx    A, @DPTR
ROM:3F36                mov     R6, A
ROM:3F37                inc     DPTR
ROM:3F38                movx    A, @DPTR
ROM:3F39                mov     R7, A
ROM:3F3A                lcall   Wrap_WriteToLCD ; Missing %ld Cards
```

Partial Deck Order Knowledge

Following the way the shuffling algorithm works, a compromised device could place specific cards into known locations with the help of the dealer.



[0, 9, x2, x3, x4, x5, x6, x7, x8 …]



This could be concealed by for example, requiring the dealer pressing the green button N times before inserting the deck.

False Shuffles

The device could be configured to perform false shuffles periodically or after a rogue dealer presses a button sequence before the shuffle.

The dealer can keep the deck in the state as after the previous hand and the cheater will be aware of the previous flop, turn, and river cards, as well as their hands, which would be on top of the deck.

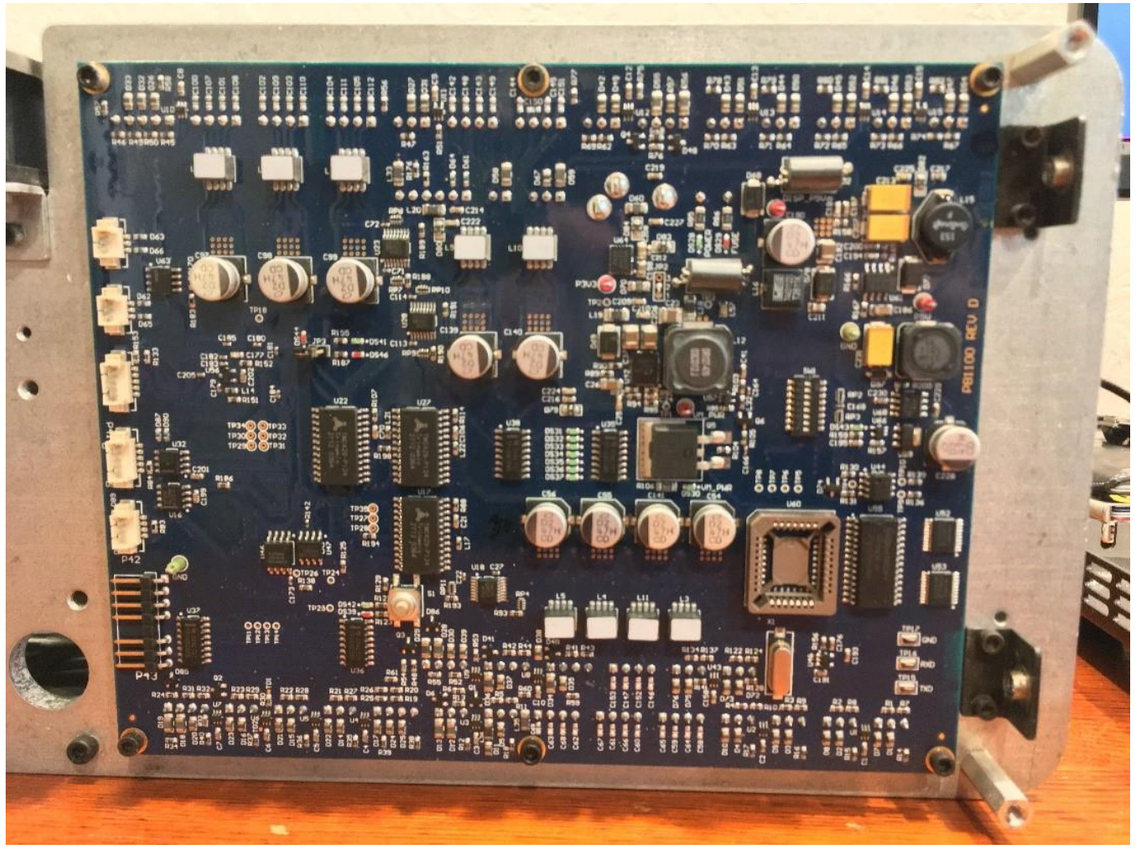
[51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Given this knowledge, upon the deck is cut by the dealer, those known cards could be considered dead, giving the cheater a significant edge.

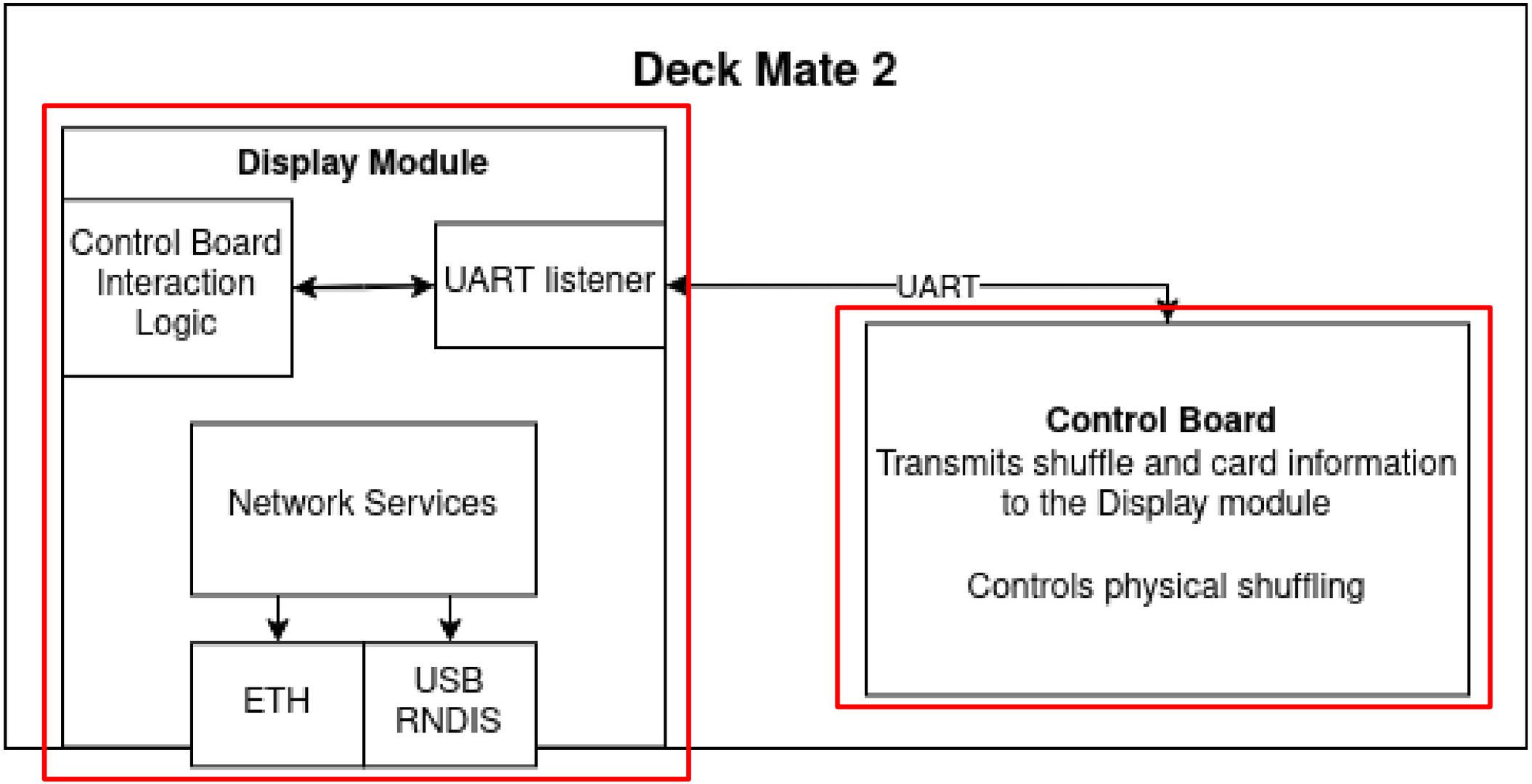
Deck Mate 2

Reverse Engineering DM2

- Goals: understand operation, RNG and shuffling algorithm
- Display Board firmware extracted via dumping unencrypted NAND
  - CPU is i.MX28 NXP ARM CPU
  - Linux 2.6.35.3
- Control Board firmware extracted from Display Board updater
  - MCU is NXP LPC1769 cortex-m3
  - QP/C Real Time Embedded Framework
- No symbols, no debug information



DM2 System Architecture



Display Board

- Reach Technology touchscreen display development module
- Embedded Linux Environment
- Responsible for connecting user interface (buttons, screen) to Control board
- Hosts various network services, used during operation and for maintenance
- Ethernet and USB RNDIS for networking

Display Board Network Services

Port	Service	Description
22	ssh	Secure Shell, used for remote display
23	telnet	Telnet, unused
80	http	Configuration Web Server
139	SMB	SMB server, no shares
445	SMB	SMB server, no shares
6000	X11	X11 Remote display server

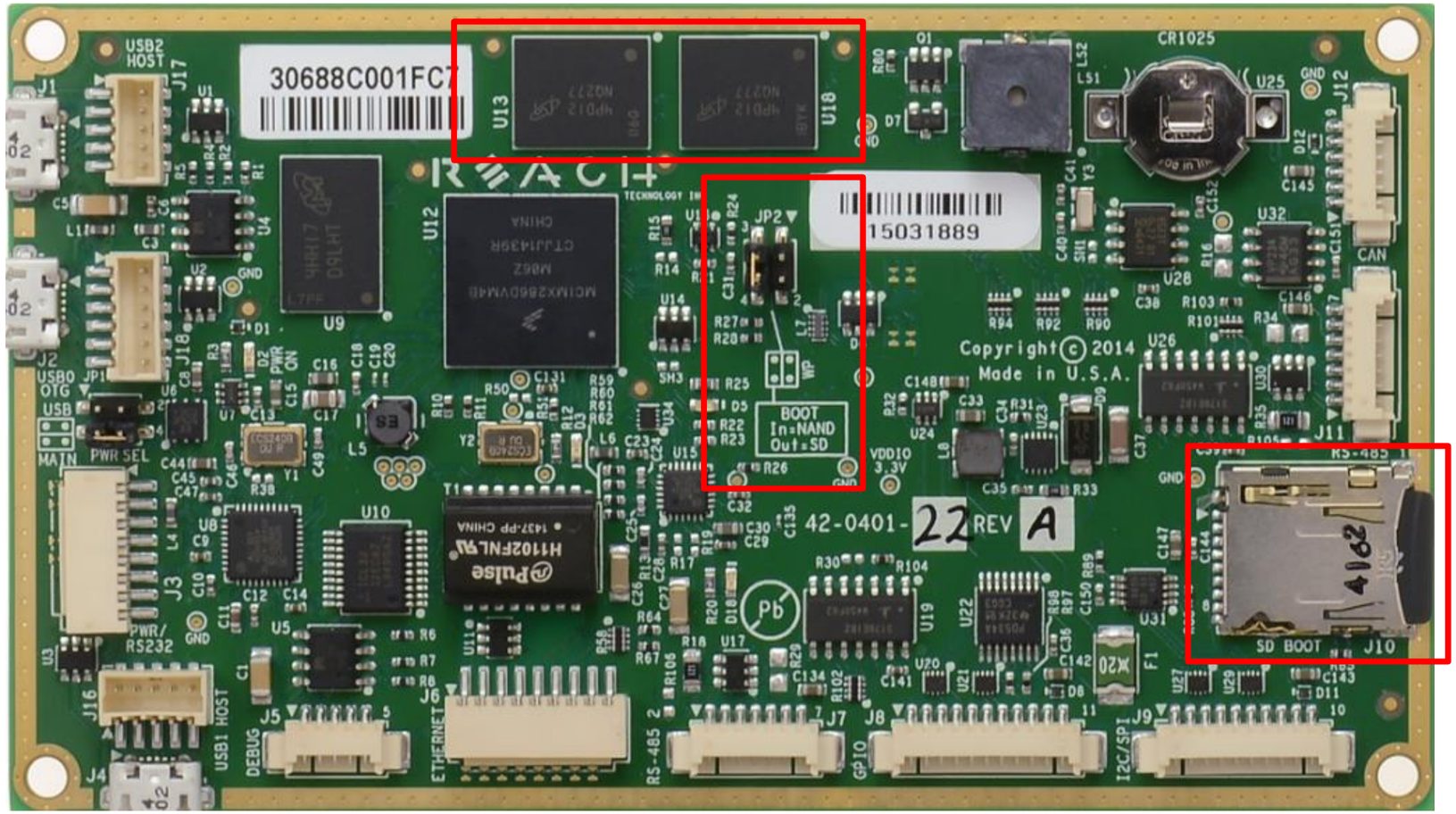
Display Board: Initial Foothold

- USB and Ethernet both expose network services – primary initial attack surface

- SSH and Telnet – Linux login prompt (no creds yet)
- SMB – No shares available
- Configuration web server requires creds, only one low-priv set available at outset

Display Board: Initial Foothold

- Need more information for network attack surface, get physical
- Reach Technology Display module can be booted from NAND or SD Card
- Built OS image with known creds, booted from SD
- Dump on-board NAND flash with Shuffler Firmware



Display Board: OS Review

- No real privilege separation
- Significantly outdated Linux kernel
- Weak, hardcoded, universal system passwords
- SSH and Telnet unrestricted beyond login prompt, login as root permitted
- No Secure Boot, filesystem integrity

```
root:$1$<redacted>:0:0:99999:7:::
daemon:*:14250:0:99999:7:::
sshd:*:0:0:99999:7:::
ftp::0:0:99999:7:::
```

```
$ time john --format=md5crypt remote-root.hash
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and
variants) [MD5 128/128 AVX 4x3])
Will run 80 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key
for status
Almost done: Processing the remaining buffered
candidate passwords, if any.
Proceeding with
wordlist:/usr/share/john/password.lst, rules:Wordlist
Proceeding with incremental:ASCII
<redacted>              (root)
1g 0:00:08:59 DONE 3/3 (2023-08-04 10:58) 0.001851g/s
799083p/s 799083c/s 799083C/s 3KDYL..411s5
Use the "--show" option to display all of the cracked
passwords reliably
Session completed
john --format=md5crypt remote-root.hash  41246.77s
user 15.83s system 7630% cpu 9:00.76 total
```

Display Board: Software Update Security

- Weak update authentication – faulty SHA1 logic and authentication key same as encryption key
- Hardcoded, universal encryption/authentication key
- Update format (self extracting bash script) easily exploitable for code execution as root
- IOActive extracted key and logic for encryption/authentication from on-board utility
- Developed a tool for creating arbitrary cryptographically valid firmware updates

Display Board: Configuration Web Server

- Hardcoded, universal credentials for all accounts including web superuser
- Credentials embedded in plaintext in service binary

Password (redacted)	Description
*****	Basic access: Shuffler label, time, deck library, history and restart.
********	Basic access + network configuration options.
*****	Basic access + network + Card ID Deck calibration, images, error details and restore default settings.
******	All previous access + Advanced Options and Advanced Calibration of motors, cameras and sensors.

Control Board System Review

- No Secure Boot Implemented
- Code Read Protection not enabled (ISP/JTAG possible)

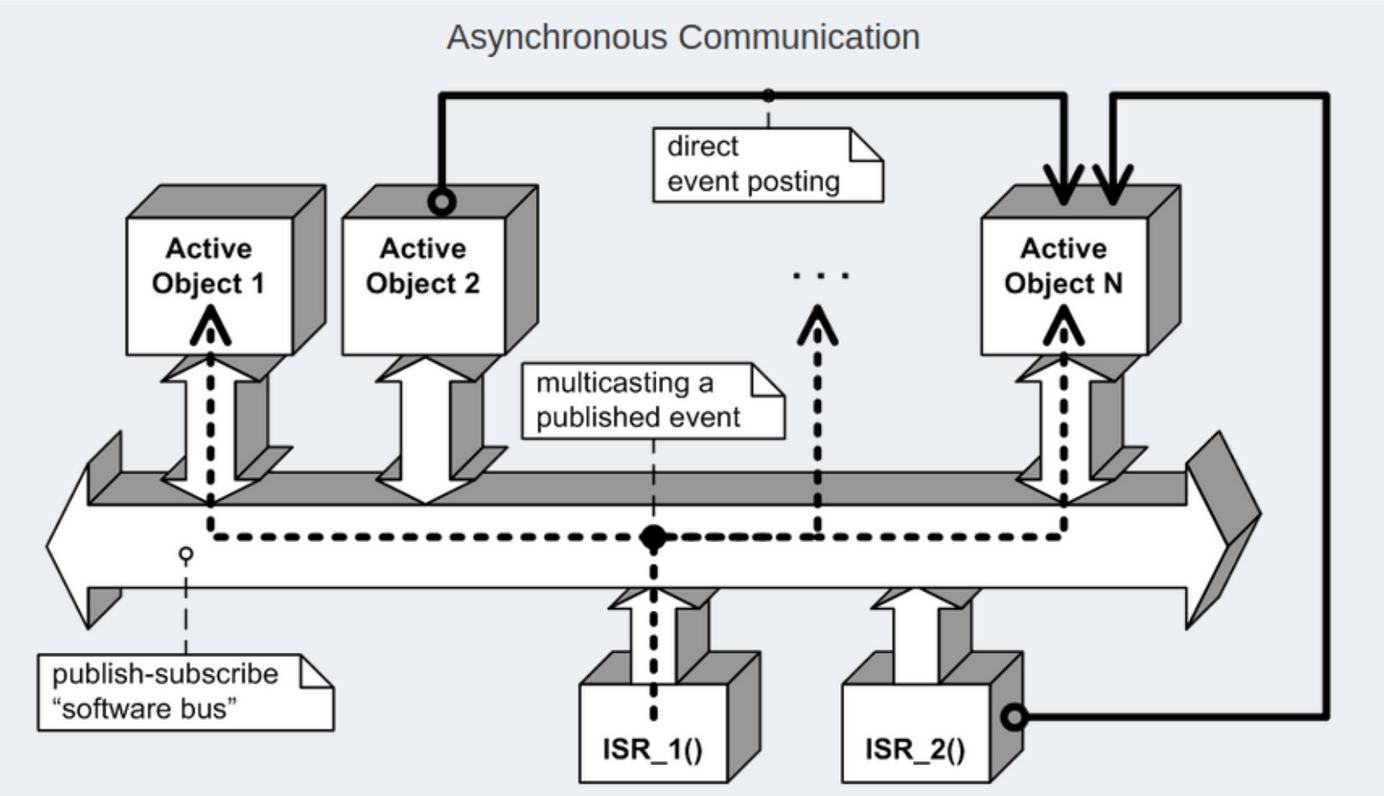
```
000002c0  45 f2 2c 53 c0 f2 05 03-5b 5c 42 ea 03 03 da b2
000002d0  7b 68 1a 70 07 f1 0c 07-bd 46 80 bc 70 47 00 bf
000002e0  80 b5 82 b0 00 af 78 60-0b 46 fb 70 7a 68 f9 78
000002f0  45 f2 b4 53 c0 f2 05 03-5b 5c 03 f1 01 03 db b2
```

Table 568. Code Read Protection options[1]

Name	Pattern programmed in 0x000002FC	Description
CRP1	0x12345678	Access to chip via the JTAG pins is disabled. This mode allows partial flash update using the following ISP commands and restrictions: • Write to RAM command can not access RAM below 0x10000200. This is due to use of the RAM by the ISP code, see Section 32.3.2.7. • Read Memory command: disabled. • Copy RAM to Flash command: cannot write to Sector 0. • Go command: disabled. • Erase sector(s) command: can erase any individual sector except sector 0 only, or can erase all sectors at once. • Compare command: disabled This mode is useful when CRP is required and flash field updates are needed but all sectors can not be erased. The compare command is disabled, so in the case of partial flash updates the secondary loader should implement a checksum mechanism to verify the integrity of the flash.
CRP2	0x87654321	This is similar to CRP1 with the following additions: • Write to RAM command: disabled. • Copy RAM to Flash: disabled. • Erase command: only allows erase of all sectors.
CRP3	0x43218765	This is similar to CRP2, but ISP entry by pulling P2.10 LOW is disabled if a valid user code is present in flash sector 0. This mode effectively disables ISP override using the P2.10 pin. It is up to the user's application to provide for flash updates by using IAP calls or by invoking ISP with UART0. **Caution: If CRP3 is selected, no future factory testing can be performed on the device.**

Control Board Architecture – QP/C and Events

- QP/C: Real Time Embedded Framework
  - "Active Object" model of Computing
  - Event-based
  - **Open source**




```
//${QV::QActive::start_} .....................................................
//! @public @memberof QActive
void QActive_start_(QActive * const me,
    QPrioSpec const prioSpec,
    QEvt const * * const qSto,
    uint_fast16_t const qLen,
    void * const stkSto,
    uint_fast16_t const stkSize,
    void const * const par)
{
    Q_UNUSED_PAR(stkSto);  // not needed in QV
    Q_UNUSED_PAR(stkSize); // not needed in QV

    QF_CRIT_STAT
    QF_CRIT_ENTRY();
    Q_REQUIRE_INCRIT(300, stkSto == (void *)0);
    QF_CRIT_EXIT();

    me->prio  = (uint8_t)(prioSpec & 0xFFU); // QF-priority of the AO
    me->pthre = (uint8_t)(prioSpec >> 8U);   // preemption-threshold
    QActive_register_(me); // make QF aware of this active object

    QEQueue_init(&me->eQueue, qSto, qLen); // init the built-in queue

    // top-most initial tran. (virtual call)
    (*me->super.vptr->init)(&me->super, par, me->prio);
    QS_FLUSH(); // flush the trace buffer to the host
}
//$enddef${QV::QActive} ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Identifying Active Objects

```
int main() {
    QF_init();  // initialize the QF framework
    BSP_init(); // initialize the Board Support Package

    // instantiate and start QP/C active objects...
    Blinky1_instantiate();
    static QEvt const *blinky1QSto[10]; // Event queue storage
    QACTIVE_START(
        AO_Blinky1,          // AO pointer to start
        Q_PRIO(1U, 1U),      // QF-prio/pre-thre.
        blinky1QSto,         // storage for the AO's queue
        Q_DIM(blinky1QSto),  // queue length
        (void *)0,           // stack storage (not used)
        0U,                  // size of the stack [bytes]
        BSP_getWorkEvtBlinky1(0U)); // initialization event

    Button2a_instantiate();
    static QEvt const *button2aQSto[8]; // Event queue storage
    QACTIVE_START(
        AO_Button2a,         // AO pointer to start
        Q_PRIO(2U, 3U),      // QF-prio/pre-thre.
        button2aQSto,        // storage for the AO's queue
        Q_DIM(button2aQSto), // queue length
        (void *)0,           // stack storage (not used)
        0U,                  // size of the stack [bytes]
        (QEvt const *)0);    // initialization event -- not used

    return QF_run(); // run the QF application
    // NOTE: in embedded systems QF_run() should not return
}
```

```
int32_t main() __noreturn

    char var_9 = 1
    initialize_system_features()
    sub_3b554()
    global_list_of_items[2].__offset(
    global_list_of_items[3].__offset(
    sub_16c8()
    sub_94c(0x10001af0, 0xcc)
    QF_poolInit(&data_10002150, 0x320
    char var_9_1 = 2
    QActive_start_(&AO_Controller, 1,
    char var_9_2 = 3
    QActive_start_(&AO_ShuffleCards,
    char var_9_3 = 4
    QActive_start_(&AO_Calibrate, 3,
    char var_9_4 = 5
    QActive_start_(&AO_UIRoot, 4, &UI
    char var_9_5 = 6
    QActive_start_(&AO_Diagnostic, 5,
    char var_9_6 = 7
    QActive_start_(&AO_Homing, 6, &Ho
    char var_9_7 = 8
    QActive_start_(&AO_Utility, 7, &U
    char var_9_8 = 9
    QActive_start_(&AO_JamRecover, 8,
    char var_9_9 = 0xa
    QActive_start_(&AO_FeedElev, 9, &
    char var_9_10 = 0xb
    QActive_start_(&AO_Platform, 0xa,
    char var_9_11 = 0xc
    QActive_start_(&AO_Rack, 0xb, &Ra
    char var_9_12 = 0xd
    QActive_start_(&AO_Unloader, 0xc,
    char var_9_13 = 0xe
    QActive_start_(&AO_PickOff, 0xd,
    char var_9_14 = 0xf
    QActive_start_(&AO_SpeedUp, 0xe,
    char var_9_15 = 0x10
    QActive_start_(&AO_Packer, 0xf, &
    char var_9_16 = 0x11
    QActive_start_(&AO_CardWeight, 0x
    QF_run()
    noreturn
```

Pattern Matching

- main identified
- Calls to QActive_start_ are passed ActiveObject references
- xrefs lead to *_initial* functions for each object
- *_initial* functions contain event subscriptions and *Root_events* function pointer

- 16 Active Objects identified

Random Number Generation

Questions to Answer:

- Hardware or Software?
- How is entropy sourced?
- What seed?
- What PRNG algorithm?

```
int32_t initialize_system_features()

    CPU_clock_configuration_mb()
    initialize_SSP0_mb()
    initialize_SSP1_mb()
    configure_pins_and_perform_ssp_transmission_and_enable_interrupt()
    initialize_ADC_mb(0xf4240)
    initialize_RIT_repetitive_interrupt_timer_mb()
    configure_gpio_pins()
    initialize_uarts_and_various_objects()
    initialize_device_info_struct(&device_info)
    initialize_RNG(&RNG, print_description, get_RITIMER_COUNTER, &device_info)
```

Random Number Generation

Entropy

- RITimer -> Repetitive Interrupt Timer
- 32 bit counter, counts from 0 to 0xffffffff
- Configurable tick rate, division of system clock
- By default, equals clock rate
- NXP LPC1769 clock max @ 120MHz

A single poll of the RITIMER counter value not sufficient for entropy – timing may be constant if *SeedRNG* called at fixed time after boot.

SeedRNG *seed_status* maintaned across multiple calls, and timer queried twice. Delay between calls variable, dependent on whims of QP/C Scheduler

Delays on the order of tens of nanoseconds will affect the final seed value

```
void SeedRNG(struct rng_struct* rng)

    if (rng->seed_status != SEEDED) {
        if (rng->seed_status != IN_PROGRESS) {
            RITimer_Counter = rng->get_RITIMER_COUNTER_fp()
            rng->seed_status = IN_PROGRESS
        } else {
            int32_t current_timer_value = rng->get_RITIMER_COUNTER_fp()
            set_item_value(&rng->seed_obj, RITimer_Counter * current_timer_value)
            rng->seed = get_item_value(&rng->seed_obj)
            if (rng->seed != 0) {
                rng->print_description(0xb, "SeedRNG: seed complete")
                rng->seed_status = SEEDED
            } else {
                rng->print_description(0xc, "SeedRNG: unable to update seed")
            }
        }
    }
```

Random Number Generation

PRNG Algorithm

- Magic constants *0x19660d* and *0x3c6ef35f*
- Parameters found in *Numerical Recipes* by D. Knuth and H. W. Lewis, in common use
- Used as *multiplier* and *increment* for Linear Congruential Generator

```
int32_t get_next_seed(struct rng_struct* rng_obj)

    rng_obj->seed = 0x19660d * rng_obj->seed + 0x3c6ef35f
    return rng_obj->seed
```

Linear Congruential Generator Security

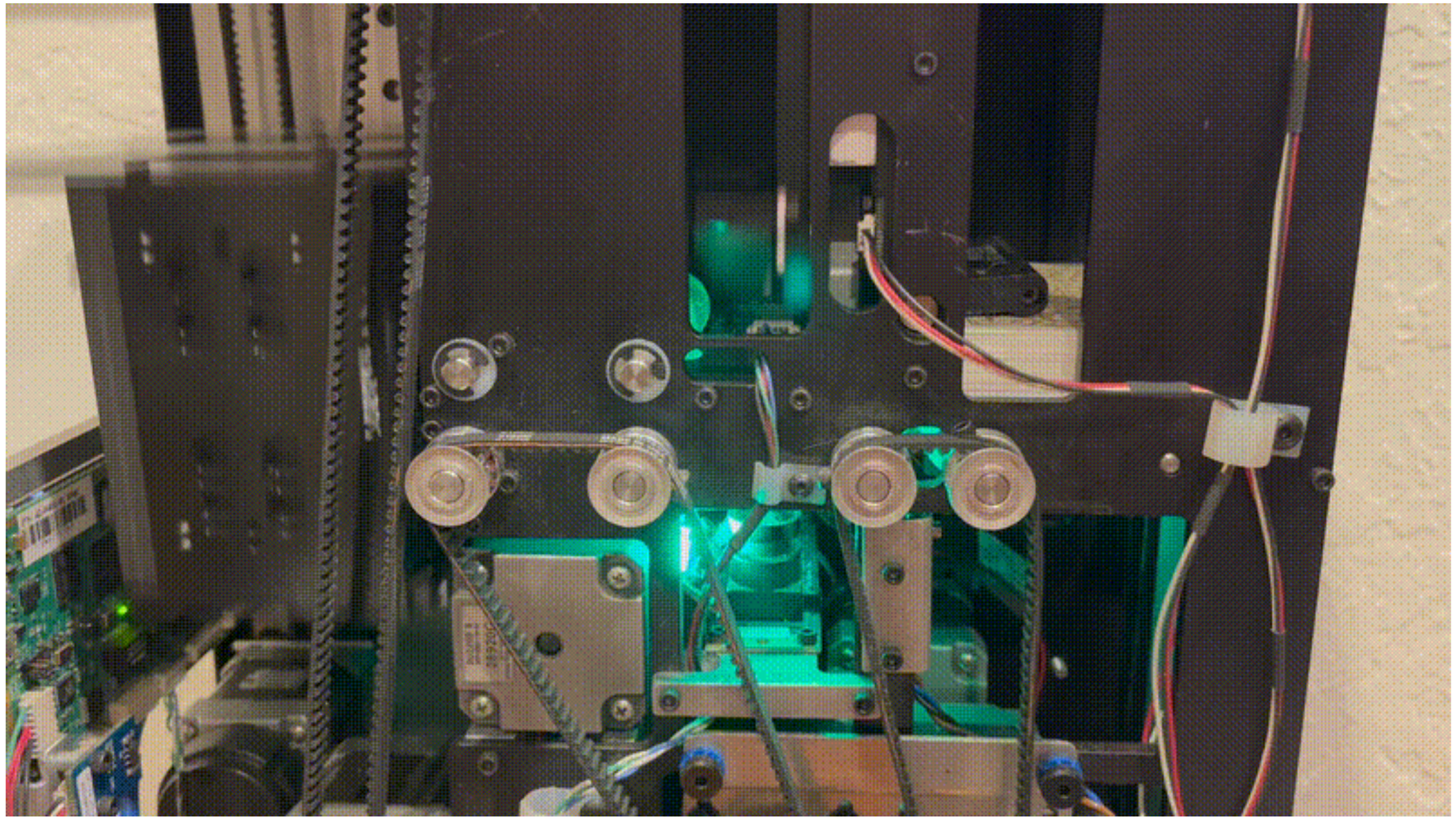
- LCG output considered to be sufficiently random for non-cryptographic applications
- Acceptably unpredictable for this specific application, without knowledge of initial seed and iteration count
- Same PRNG as the Deck Mate 1

Shuffling Algorithm

- Constructs an array *target_positions* equal to size of inserted deck
- Each index in array represents a card in the unshuffled deck
- Populates this array with *target position* values
- Each card in unshuffled deck at position *i* is placed at position *target_positions[i]* in the shuffled deck
- Similar to Deck Mate 1 randomization

```
int
RandomizeCards(struct RNG *rng, short total_cards,
               short top_pos, short bottom_pos,
               short *target_positions, short rng_buf_size)
{

    // populate ordered list of card positions, 0
    // to 51 in normal case
    for (int i = 0; i < total_cards; i++) {
        rng->ordered_positions[i] = i + bottom_pos;
    }

    short max_slot = top_pos - bottom_pos;
    int read_position = 0;

    for (; read_position < total_cards - 1; read_position++, max_slot--) {

        // generate random position, then set the target positions at the current
        // offset into the shuffled deck to the card position found in the ordered
        // cards array at the random position
        int rand_slot = generate_random_int_in_range(rng, max_slot, 0);
        target_positions[read_position] = rng->ordered_positions[rand_slot];

        int curr_slot = rand_slot;
        int next_slot = rand_slot + 1;

        // remove the already used position from the ordered positions array
        // so it can't be reused, allowing for duplicate rand_slots
        // to not produce duplicate positions
        for (; next_slot < max_slot; curr_slot++, next_slot++) {
            rng->ordered_positions[curr_slot] = rng->ordered_positions[next_slot];
        }
    }
    // put the last remaining position in the ordered positions array into the final slot
    // of the shuffled cards. This is fine and still not predictable, since
    // the fact that this position was not selected was a result of calls to generate
    // random function happening to miss it.
    target_positions[read_position] = rng->ordered_positions[0];
    return 1;
}
```

Physical Shuffling Mechanism



Shuffler Mode

- Multiple modes supported
- Normal Shuffle
- Sort: multiple modes for different suit orders

- Sort mode reads card data from camera for placement information.
- Normal Shuffle reads order information from virtual deck, card values in physical deck irrelevant (though still read and recorded).

```
enum shuffler_mode mode
bool is_randomized
if (zx.d(me->deck_is_randomized) == 0) {
    mode = get_item_value(&ShufflerMode)
    if (mode != SORT) {
        is_randomized = false
    }
}
if (zx.d(me->deck_is_randomized) != 0 || (zx.d(me->deck_is_randomized) == 0 && mode == SORT)) {
    is_randomized = true
}
```

Cheating with DM2

Cheating with DM2: Deck Order Manipulation

- Repurpose the DM2 Camera to identify each card and place it in a target location
- This allows for "sort" mode, where cards are placed in a specific order
- Modifying Control Board firmware allows for cheater-specified sort order
- Dealer will usually cut deck, disrupting intended order
- Deck orders which allow for the cheater to win consistently are suspicious
- Requires cheater to be in a specific seat

Cheating with DM2: Exfiltrating the Deck

- Use the Camera to read the current card being shuffled and exfiltrate it.
- Control Board firmware can be modified so that this information is reported to the Display board over UART
- Cheater-controlled device connected to the Display Board extracts this data
- Order of deck post-shuffle can be transmitted to the cheater
- Deck order is not modified and thus avoids suspicion
- Deck cut can be accounted for
- Does not attempt to force a win but rather increase odds for cheater, thus no specific seat or game configuration is necessary

Cheating with DM2: Proof of Concept

Attack Scenario: Cheating player

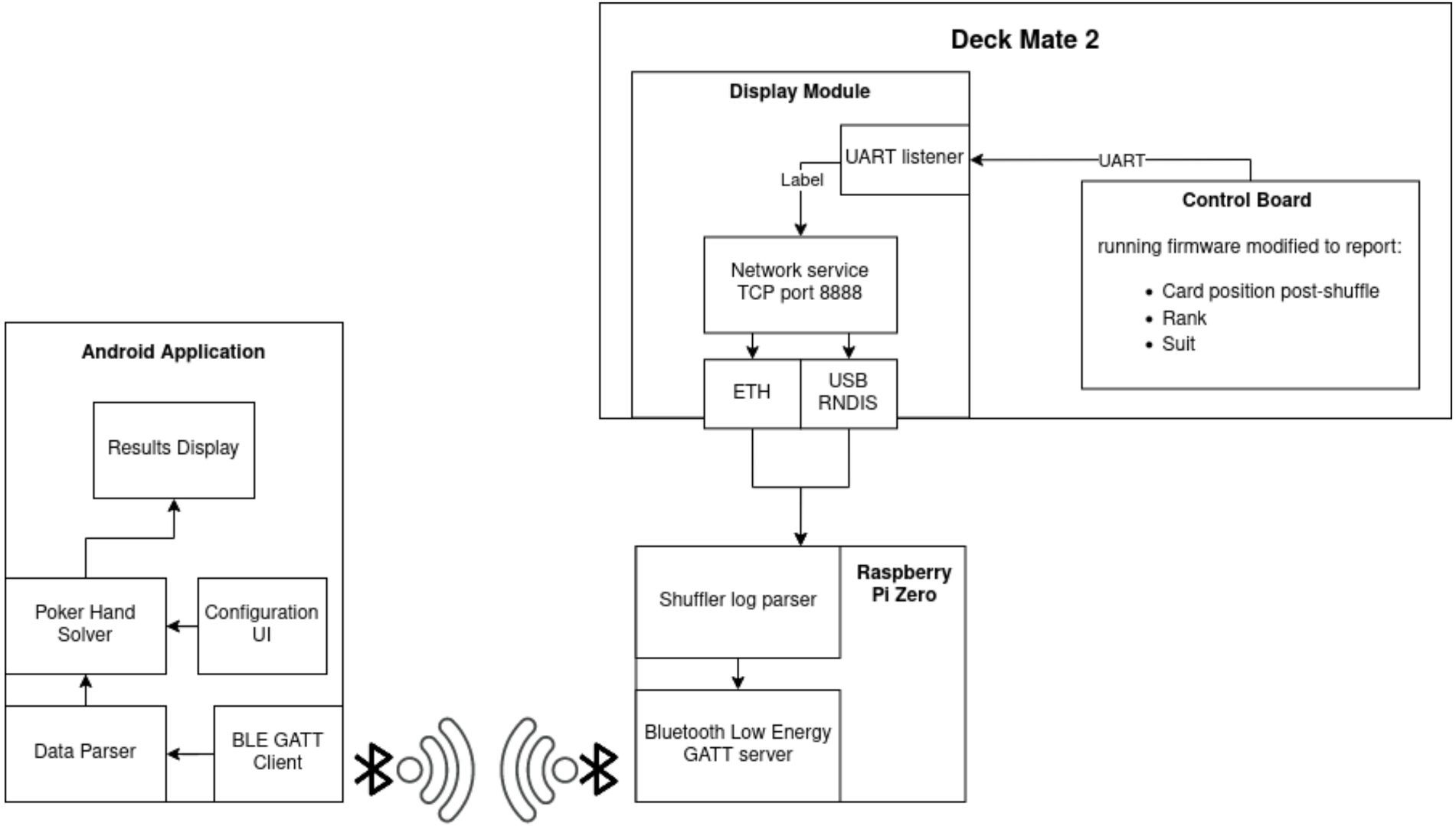
Vulnerabilities Leveraged

- SSH exposed over USB/Ethernet/Cellular
- Hardcoded Display Board root credentials
- Incomplete GAT implementation
- Lack of firmware update security for Control Board
- Lack of secure boot for Control Board
- Lack of filesystem integrity protections for Display Board
- Inadequate physical security for Deck Mate 2 device and enclosure
- Inadequate physical access restrictions/monitoring in common deployment

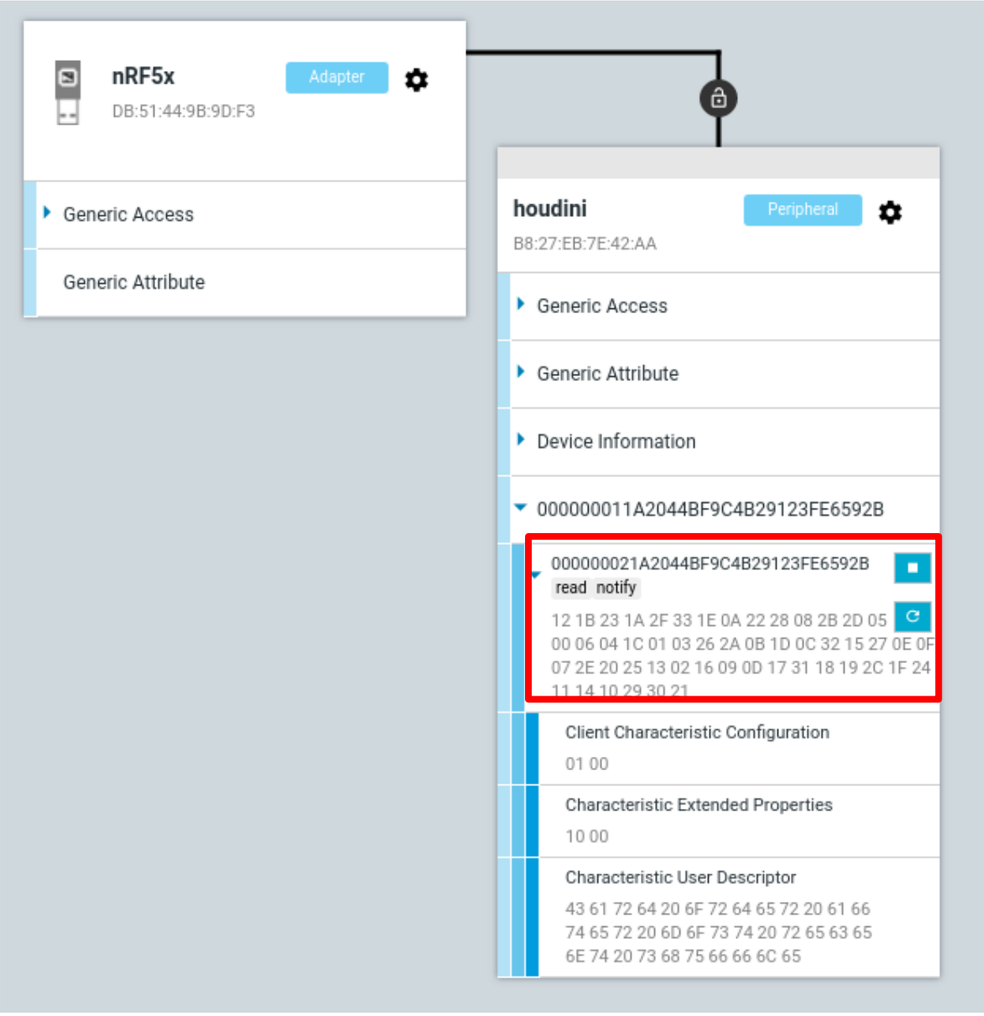
Equipment

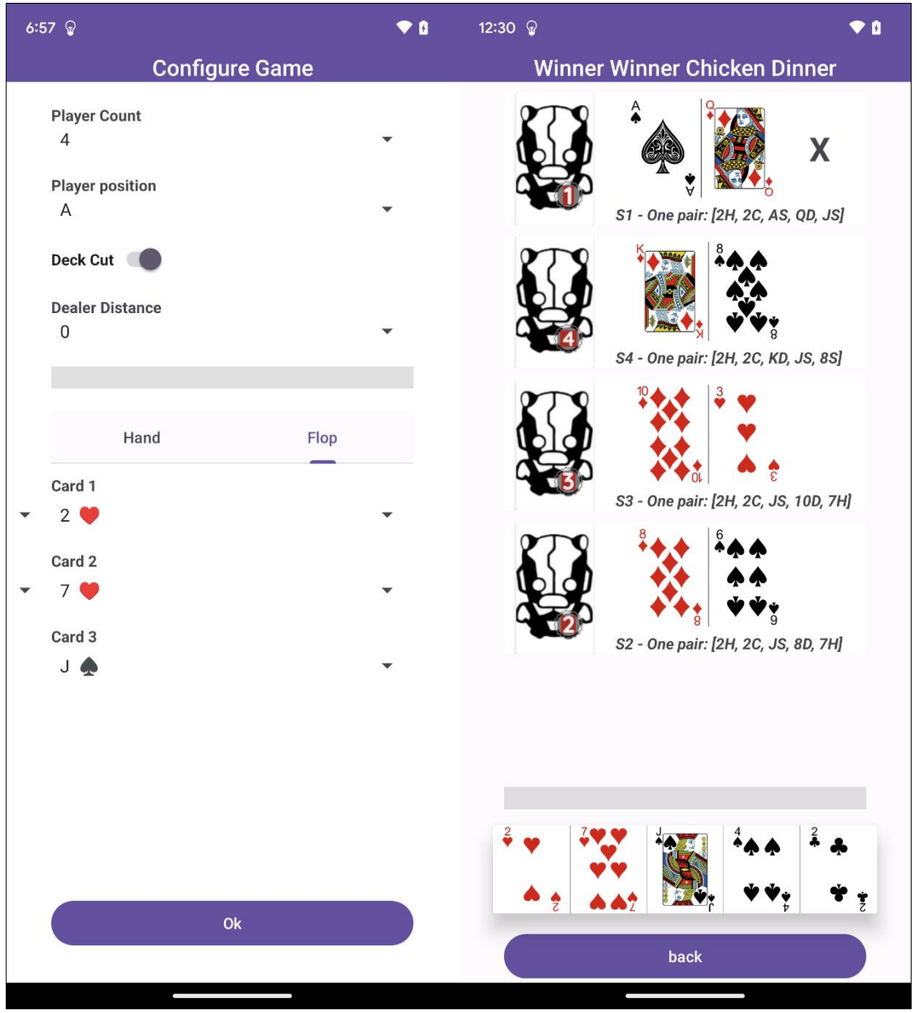
- Raspberry Pi Zero W
- Android Phone

Cheating with DM2: Exfiltrating Deck Information via Bluetooth



Cheating with DM2: Exfiltrating Deck Information via Bluetooth





Conclusions

Impact

- Automated shufflers and gaming standards sport surprisingly weak security given the high-stakes nature of their purpose
- Research focused on Poker, but similar shufflers are used in other table games such as Blackjack and Baccarat and incur losses to gaming operators
- Overall, cheating scenarios like this affect players trust in the integrity of the game, without trust there is no game

Recommendations

Gaming Operators

- Implement physical restrictions on access to exposed ports
- Leverage relationship with manufacturer to directly address concerns

Players

- Ultimately boils down to your trust in the operator/game

Recommendations

Doug Polk @DougPolkVids · May 26, 2022

I have heard of MANY different occasions where this tactic was used to cheat players. In home games, in clubs, and even at least once at a major casino. There are countless times this has been used to swindle people out of their money.

1 35

Doug Polk @DougPolkVids · May 26, 2022

It is far more likely that players use the knowledge of deck order to cheat than sort the deck into rigged hands. Cutting the deck does nothing, as once you know 1 card location you know where it was cut and thus the entire order.

3 1 33

Doug Polk @DougPolkVids · May 26, 2022

The main ways that this information is relayed to in game players is via ear piece or their mobile device. I dont know the exact specifics of that transmission.

Wanted to clear up these common misconceptions I'm seeing regarding using the shufflers to cheat.

9 37

Doug Polk @DougPolkVids · May 26, 2022

Oh 1 last thing.

If you are ever worried ask for the dealer to riffle a few times at the end. With a few riffles its basically impossible to use this technique to cheat.

21 4 91

https://twitter.com/DougPolkVids/status/1529976301536280576

Questions?

A detailed whitepaper will be available in the new few weeks

Thank you

IOActive.