A Monster Emerges from the Chrysalis
(Experiences reverse-engineering the Luna CA3)

Mike Bond

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Computer Laboratory
10th February 2004
Contents

• Security API attacks
• Introducing the Luna CA3
• Reverse engineering with IDA
• The cloning protocol
  – Stage 1: Finding it
  – Stage 2: Understanding it
  – Stage 3: Breaking it
• Implementing host side interface
• Lessons learned
This was a team effort!

Many many thanks to:
  – Steven Murdoch
  – Dan Cvrcek

Also thanks to:
  – Richard Clayton, IH, Stephen Lewis, Jolyon Clulow
  – and many more…
What is a Security API?

- A command set that uses cryptography to control processing of and access to sensitive data, according to a certain policy.
Security API Attacks

- APIs for HSMs have evolved to support more and more transactions and sophisticated features – but they are getting too complex now
- Use the permitted commands of the interface in an unusual sequence to trick a device into revealing secret key material

- Are simpler, quicker and more effective than going in by the ‘front door’?
- Or are they?
Simple

U→C : PAN
C→U : \{ PAN \}_TC

U→C : \{ PAN \}_TC , \{ PMK1 \}_TMK
C→U : \{ PAN \}_PMK1
Not So Simple?

Data Key

A^B

Data Key Part

A

Set of Data Keys

Test Pattern 0

Set of Data Key Parts

XOR

Set of Test Vectors

ENCRYPT

Data Key Part B

Data Key A^B

Set of Test Vectors

EXPORT

Set of Exporter Key Parts

Exporter Key Part X

Exporter X^Y

MIMCRACK

Exporter Key Part Y

DECRYPT

Valuable Key Material

Exporter Key X^Y

Exported Valuable Key Material
The Luna CA3

• PCMCIA token, for secure storage of private keys for Certification Authorities
• Manufactured by Chrysalis-ITS (Toronto), acquired by Rainbow, acquired by SafeNet
• Became popular during the rise of PKIs in the dot com boom (Verisign exclusively uses Chrysalis kit for key storage)
• Uses the PKCS#11 API (through an internal proprietary ‘Luna API’)}
Luna Dock
The Cloning Protocol

- Used for backup and availability
- Initialise a new token into the same domain (you need the **RED** key)
- Log on to source and destination tokens (with **BLUE** security officer key)
- Select an object and call `CA_ClonePrivateKey` to transfer between source and destination. The devices exchange public keys then set up a session key for the transfer.
Luna CA3 – Pin Entry Device (PED)
Luna CA3 – Datakeys
Primary Goal

Develop a way to extract all PKCS#11 keys in the clear from the Luna token, with the co-operation of the security officer.
Motivations

• Break customer lock-in – help the market
• Learn about internal HSM architecture
• Find implementation faults (buffer overflows?)
• Find new Security API attacks?
• Learn useful skills along the way
  – Reverse-engineering
  – Assembler
  – Particular disassembly tools
A Simple Plan

• Open up the card
• Reverse-engineer the flash chip
• Discover the cloning protocol
• Extract device keys
• Use keys to impersonate token in cloning protocol
Stage 1: Finding the Protocol

- Get the ARM code
- Get a reverse engineering tool
- Familiarise and Mark-up ARM code
- Identify Command Despatcher
- Annotate Commands
- Intercept and Decode PCMCIA Bus
- Locate and Decode Cloning Protocol
Luna CA3 – Depackaged

- Mystery FPGA
- PCMCIA Controller?
- Stuff
Luna CA3 – Depackaged

Flash 1

Flash 2

StrongARM
The Luna Flash File – AM29.BIN

- Two 1/2MB flash chips, holding half words
  - ~300KB code
  - ~500KB data
  - ~200KB blank

- Complexity
  - 1035 subroutines
  - ~1700 pages of assembler (on this screen)
IDA – The Interactive Disassembler

• Made by ‘Datarescue’ – one man consultant went commercial with the tool he developed for himself. Cost ~$700 for 2 year licence.

• Beautiful windows GUI and navigation system. Rename functions and variable names on-the-fly and the new information propagates through the disassembly listing
Reverse-Engineering Golden Rules

Conventional wisdom is one rule...

• Figure everything out for yourself!
Reverse-Engineering Golden Rules

My wisdom...

• If you don’t know what to do, instead, *do what you can.*

• Give *everything* a name.

if you get stuck...
http://www.babycakesinternational.com/100topbabnam.html
or use movies, friends, books
Make every letter in a name count!

Group C1 type functions into larger clumps
Pay special attention to most called functions

`memcpy` 327 calls

Start propagating type information
  - (memcpy arg 2 is length, args 0 and 1 pointers)
Finding the Command Despatcher

• Search for the biggest case switches…
  – 45 switch statements in total
  – ranging between 0x17 and 0x5 ways
  – no idea what the command encoding was

```
ADDLS PC, PC, R0, LSL#2 ; switch 0xC ways
```

• Two pages from back of policy document listing the Luna API commands categorised by module was all we had.
## APPENDIX C. Session And Login States Required For Luna Token Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>To</th>
<th>Times</th>
<th>Key</th>
<th>Session</th>
<th>Open, No.</th>
<th>Login</th>
<th>SO</th>
<th>Logged On</th>
<th>User</th>
<th>Logged On</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Token Management Commands</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNA SERNOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAWithType</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNASER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPENACCESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSEACCESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session Manager Commands</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPENSESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSESESSION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPENACCOUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSEACCOUNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Module Commands</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPENUSER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPENUSERNAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPEN Group</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPENGroupUser</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSEUSER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Management Module</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPEN OBJECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSE OBJECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPENOBJECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSEOBJECT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Management Module</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAOPENOBJECTSET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNACLOSEOBJECTSET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key Management Module</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAGENERATEKEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAGENERATEKEYWVALUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAGENERATEKEYWVALUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAGENERATEKEYWVALUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAGENERATEKEYWVALUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUNAGENERATEKEYWVALUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Unrestricted*
The Command Dispatcher

Raw command ID (single byte)

\[\text{C1}_30\text{SER\_MODULE\_DESPATCHER\_LUCY}\]

Switch 0xC ways

\[\text{C1\_xlate\_cmd}\]

Switch 0x9 ways

\[\text{L3\_C1\_LUCY\_moduleA}\]
\[\text{C1\_30SER\_BSW\_KEYMANAGE}\]

Switch 0x8 ways

\[\text{L3\_C1\_MAIN\_MODULE}\]
\[\text{L3\_C1\_CRYPTO\_MODULE}\]

Switch 0x7 ways

\[\text{L3\_C1\_USER\_MODULE}\]
\[\text{C1\_RANDOM\_NUM\_GEN\_MODULE}\]

45 switch statements in total…

\[\text{L3\_C1\_OBJECT\_MANGER}\]
\[\text{C1\_SESSION\_MGR\_MODULE}\]
Intercepting the PCMCIA Bus
Interception the PCMCIA Bus
## Bus Intercepts: Cloning Protocol

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>TARGET</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUNA_FIND_OBJECTS</td>
<td>LUNA_FIND_OBJECTS</td>
</tr>
<tr>
<td>LUNA_GET (SLOT 0xE)</td>
<td>LUNA_DESTROY_OBJECT</td>
</tr>
<tr>
<td>LUNA_GET (SLOT 0xF)</td>
<td>LUNA_FIND_OBJECTS</td>
</tr>
<tr>
<td>LUNA_CLONE_AS_SOURCE</td>
<td>LUNA_GET (SLOT 0xE)</td>
</tr>
<tr>
<td>LUNA_GET (SLOT 0xE)</td>
<td>LUNA_GENERATE_KEY</td>
</tr>
<tr>
<td></td>
<td>LUNA_SET_UP_MASKING_KEY</td>
</tr>
<tr>
<td></td>
<td>LUNADESTROY_OBJECT</td>
</tr>
<tr>
<td></td>
<td>LUNA_GENERATE_KEY_W_VALUE</td>
</tr>
<tr>
<td></td>
<td>LUNA_CLONE_AS_TARGET_INIT</td>
</tr>
<tr>
<td></td>
<td>LUNA_CLONE_AS_TARGET</td>
</tr>
<tr>
<td></td>
<td>LUNA_GET (SLOT 0xE)</td>
</tr>
</tbody>
</table>
Luna Key Cloning Protocol

**SOURCE**

- **LUNA_GET (SLOT 0xF)**
  - \(\{K_S\}K_{chrys}^{-1}\)

- **LUNA_CLONE_AS_SOURCE**
  - \(K_S^{-1}, K_{chrys}\)
  - \(K_X = N_T \oplus N_S \oplus K_D \oplus C\)

**TARGET**

- **LUNA_CLONE_AS_TARGET_INIT**
  - \(\{K_T\}K_{chrys}^{-1}\)

- **LUNA_CLONE_AS_TARGET**
  - \(K_T^{-1}, K_{chrys}\)
  - \(K_X = N_T \oplus N_S \oplus K_D \oplus C\)

Legend:
- \(\{\text{REQ} \ , \ N_T\}K_S\)
- \(\{\text{REP} \ , \ N_S\}K_T\)
- \(\{\text{APPKEY}\}K_X\)
Stage 2: Understanding the Protocol

- We knew *what* the cloning routine did, but not *where* the key material came from.
- The encrypted key material came from LEELA, the decryption key from JADE.
- We could see encryption and decryption, but not exactly *how*... had to mark-up the crypto routines called by the cloning code.
  - Identify which algorithms are used.
  - Identify algorithm parameters, key lengths.
  - What about IVs?
The Luna Mysteries

- To understand the protocols we needed to discover the purpose of some puzzling functions
  - C4_crypt0_action_mechsw
  - LEELA
  - JADE
  - 'EDAFLU'
C4_crypt0_action_mechsw

• Seemed to be the central function for symmetric crypto – called by…
  C25_C_ACTION_0_ENCRYPT
  C27_C_ACTION_0_DECRYPT

• Called C5_do_BlockEncrypt_CBC, and called lots of crypto-like routines, but the two seemed unlinked.

• Evidence of software DES was found (key-schedule), but the block encrypt function called HIFN (a DES accelerator manufacturer) IO functions. Yet there was no HIFN chip in the token. How and where was the DES done?
C4_crypt0_action_mechsw (2)

• Solution: a well hidden table jump inside the CBC loop, once discovered made the code make sense
• There were 3 function tables – one for preparing key schedule, one for encrypt and one for decrypt
• DES key schedule was calculated in software, then uploaded into accelerator chip (this upload was mistaken for the full DES calculation)
• Why was DES done as a composite in H/W and S/W? To claim ‘hardware accelerated DES in marketing brochure’? Space was too limited in FPGA?
Hunting LEELA

- Official name: C68_LEELA_load and C35_LEELA_save

- The token private key came from LEELA slot 0xF, but where did the slot live? The code used memcpy to pluck it from unusual address, but we only had rough idea of the memory map. Could they be special secure memory inside FPGA?

- Eventually: discovered that LEELA slot save code looked like flash file update code: became convinced that slots lived on 1MB flash image.

- Wrote script to scan flash for linked list of pointers as theorised from reader code. Success! Found LEELA slots at 0x88000 in AM29.BIN
Finding JADE

• JADE, officially:
  
  C12_JADE_prep_cryptolstruct_entryA  
  C4_JADE_entryB

• JADE takes no arguments, and returns a cryptolstruct, containing a DES key or a 3DES key used for decrypting the contents of a LEELA slot.

• Problem: JADE walks through data structure in RAM to find keys – how can we locate code that set up keys in data structure?
Finding JADE (2)

• Solutions:
  – Take a guess. Look in login routines – maybe JADE keys come from physical datakeys
  – Observe class of error code in JADE functions, and search for functions exhibiting similar error codes

• Success: `C3_LOGINOUT_setup_auth_contexts_JADE` was found. In fact, key material in JADE slots came from a decrypted version of the data structure inside a LEELA slot.

• But where did the encryption key come from? The datakey? And if so, which?
Finding JADE (3)

- Problem: So how can the keys be stored in encrypted form when the token is uninitialised? – there is no blue key
Datakeys Revisited

- Security Officer
- KCV Domain
- User
- M-of-N User Keys (optional)
The Luna PED Protocol

• PED talks to token be reusing high address lines from PCMCIA spec as bidirection communications channels
• Three lines: RESET, DATA, and DATA_VALID
• However, DATA_VALID was clocked in an unpredictable erratic way. Reason: Luna token implements serial communications protocol in software, and cycle time of loop was data dependent.
• Used a datakey reader to make an independent observation of data on keys, and try to observe this on the bus.
The ‘EDAFLU’ Story

- During initialisation of a token, there is a special requirement: insert the mystery ‘grey key’
- Grey key not mentioned at all in documentation, or release notes
- Contained 64 bytes, mainly zeroes, save for one interesting constant… more 0xDEADBEEF?

```
00 00 01 00 00 30 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 65 64 61 66 6C 75 00 74 00 00 00
```
The ‘EDAFLU’ Story (2)

Datakey reader had wrong half-word endian!
Extracting the Token Private Key

- LEELA slot contained encrypted private key of token, in two forms, encrypted under grey key and under current blue key.
- Key material from data key retrieved
- JADE decrypts slot and puts clear keys in RAM
- We re-implemented decryption of LEELA slot using hash of ‘default’ key.
- Unfortunately…FAILURE
- Need to emulate ARM code and try again, or switch to another plan
Stage 3: Breaking the Protocol

- Find the protocol in the code stack
- Familiarisation and mark-up of PKCS#11 DLL code in CRYST201.DLL
- Follow data flow inside DLL
- Intercept and change data flow
- A change of plan: CVKs
The Luna Code Stack

Luna Enabler
163KB Application

CRYST201.DLL
287KB Library Code

LUNACR0
141KB Device Drivers
LUNANT
65KB
LUNAVPN

Luna Controller
~256KB Hardware

Luna Dock
~256KB

Luna CA3
1MB
Inside CRYST201.DLL

• Usual PKCS#11 entry points exported, but some extra vendor-specific ones of interest
  CA_SetCloningDomain
  CA_SetTokenCertificateSignature
  CA_ClonePrivateKey (and many more...)

• DLL written in mix of C++ and C. PKCS#11 entry points called C++ methods of object hierarchy representing different models of Luna token (Luna 1, Luna 2, Luna CA3, Luna RA etc.)

• These methods called ‘SOLAR API’, which corresponded closely (but not exactly) to Luna API intercepted on PCMCIA bus. SOLAR API called C stub functions, which called I/O methods of C++ class hierarchy representing different device drivers.

• To summarise: a real mess inside
Inside CRYSST201.DLL

PKCS#11 API

JT_CMDSET_MAIN

JT_SOLAR_API

C
C++

CA_SetTokenCertificateSignature

D1_100h_MAIN_set_token_cert_sig

D4_SOLAR_1F8_LUNA_LOAD_CUST_VERIF_KEY

ETHAN GOAT
Write DWORD

NIOBE FISH
Write block

DOZER
Send cmd

WORM
Get data buf

SKUNK

GOAT (A-D)

FISH (A-D)

CAT

ZAK

AUSTIN4 LLCMDS

AUSTIN TOKENIO

WORM

SKUNK

GOAT (A-D)

FISH (A-D)

CAT

ZAK

DRV40
get numsl

DRV00
get tokpr

DRV20
get insct

DRV08
read wind

DRV04
execute

DRV48
reset
Customer Verification Keys

**SOURCE**

LUNA_GET (SLOT 0xF)

\[\{K_S\}K_{\text{chrys}}^{-1}\]

LUNA_CLONE_AS_SOURCE

\[K_S^{-1}, K_{\text{chrys}}, K_{\text{cust}}\]

\[K_X = N_T \oplus N_S \oplus K_D \oplus C\]

**TARGET**

LUNA_CLONE_AS_TARGET_INIT

\[\{K_T\}K_{\text{chrys}}^{-1}\]

LUNA_CLONE_AS_TARGET

\[K_T^{-1}, K_{\text{chrys}}\]

\[K_X = N_T \oplus N_S \oplus K_D \oplus C\]
Cloning to Clear

1. Generate known $K_{cust}$ and $K_{cust}^{-1}$

2. Load CVK

2. Send chosen $N_T$

3. Generate known $K_T$ and $K_T^{-1}$

4. Sign certificate authorising chosen $K_T$

5. Receive source nonce under chosen $K_T$

5. Combine nonces with KCV and decrypt APPKEY

LUNA_LOAD_CVK

$K_{cust}^{-1}$

LUNA_CLONE_AS_SOURCE

$K_{chrys}$

$K_{cust}$

$K_x = N_I \oplus N_S \oplus K_D \oplus C$

$\{REQ, N_T\}K_S$

$\{K_T\}K_{cust}^{-1}$

$\{REP, N_S\}K_T$

$\{APPKEY\}K_X$
Making the Key Cloning Vector

RAW KCV

xor

64 bytes

MD5

0x3CC3A596
0xDEADBEEF
0x01234567
0x89ABCDEF

xor

Hashed KCV + C

16 bytes
Making the Key Cloning Vector (2)

Hashed KCV + Constants → var_80 → 16 bytes

→ constant 0x1 → SHA1 → var_98 → 16 bytes

→ 4 bytes → 20 bytes → 40 bytes

→ MD5 → 16 bytes

→ xor → 20 bytes → 24 bytes
Making the Key Cloning Vector (3)

Hashed KCV + C

constant 0x2

16 bytes  4 bytes  16 bytes

36 bytes

SHA1

20 bytes

xor

24 bytes

3 key 3DES

K1        K2        K3

24 bytes

24 bytes

24 bytes
Lessons Learned

• Going in the front door (reverse-engineering) is tough, but it is a skill that can be learned, and done again much more quickly

• Choice of tools, and knowledge of tools is vital to chances of success

• It’s easy to drown in a sea of maybes and unknowns and give up. The golden rules of reverse engineering can help
  – “do what you can”, and “name everything”
Lessons Learned (2)

• Legacy code is much better camouflage than obfuscation to slow reverse engineering.
• 0xDEADBEEF hinders reimplementation of crypto code as it has to bit-for-bit perfect.
• A new defence – stupidity! If the programmer understands his task poorly, the reverse engineer will have an even worse time.
• Beware of undocumented features in your API. Chrysalis didn’t let on about the CVK, what are other manufacturers hiding?
Lessons Learned (3)

• The Luna CA3 API is secure, but the architecture has accumulated too much baggage – if it is pushed much further, it may break completely.

• If the Luna CA3 is anything to go by, HSM code is no better than O/S code.

• Even if your architecture is not exploited by a Security API attack, it may still be used in an unexpected way.
IDA Strengths

• Excellent navigation interface design, once familiarisation done
• Excellent cross-referencing comment system
• Good auto-analysis and support for standard libraries
• Strong use of colours and graphics to help spot patterns
• Good extensibility, supporting scripts and plugins
IDA Weaknesses

• No graphing of conditional jumps or calculated jumps
• Poor support for stack variables on ARM
• Poor documentation – many features discovered late
• Non-standard look and feel
• Some cosmetic defects
Weak Spots in the Luna CA3

• Application Key Integrity
  – During transport, cipher was 3-Key 3DES in CBC with fixed IV, 32-bit CRC with custom polynomial used for ‘integrity’

• Buffer, integer overflows?
  – Will take a brief look shortly

• Cryptographic Algorithms
  – “BRUNO C.” (to be explained…)
Question: How do you encrypt data that doesn’t fit to a block boundary?
“BRUNO C.”

- Question: How do you encrypt data that doesn’t fit to a block boundary?

**Problem:** Not enough 0xDEADBEEF!
• Question: How do you encrypt data that doesn’t fit to a block boundary?
Luna CA3 users, don’t worry...
Luna CA3 users, don’t worry...

YOU STILL NEED THE BLUE KEY
More Information

http://www.cl.cam.ac.uk/~mkb23/research.html

Technical Report coming April 2004

CL:  Possible reverse-engineering mini course coming soon