Model Checking Cryptoproductors
(or “Why I Like the British Museum”)

Mike Bond

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Computer Laboratory
12th November 2002
Contents

• The Problem : Analysing Security APIs
• Protocol Analysis Tools
• The Formalisation Step
• Experiments with SPASS
• The “MIMSEARCH” Tool
• Implementation Tour
• Results & Conclusions
What is a Security API?
Security APIs

- Found at trust boundary of tamper-resistant processors which use cryptography to control processing of and access to sensitive data.
Why Automate API Analysis?

- APIs are getting more complex – more human effort required, and few skilled people
  - VSM Banking API ’89 – 80 pages
  - CCA Banking API ’02 – 458 pages
- Can make finding attacks quicker
- Can spot stupid mistakes at once
- Might one day find an attack of its own accord?
- Can search for all instances of a known attack
- Operating tool can help build intuitive knowledge
Protocol Analysis Tools

CAPSL

Casper

CSP

N-PATRL

NRL Analyser

Inductive Method

Isabelle SPASS Spin SMV FDR Prolog

Theorem Provers Model Checkers Search Tools

Mona Kronos

UPPAAL HyTech
Formalising APIs

1. Read specification (or instruction manual)
2. Decide on primitives required
3. Choose analysis tool supporting primitives
4. Formalise each command
5. Test against known attacks
6. Patch to prevent known attacks
7. Search for unknown attacks
Data_Key_Import

Data_Key_Import (CSNBDDKM)

<table>
<thead>
<tr>
<th>Feature/Product</th>
<th>O/S</th>
<th>AX</th>
<th>NT</th>
<th>OS/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM 4738-1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The Data_Key_Import verb imports an encrypted, source DES DATA key and creates or updates a target internal key token with the master-key encrypted source key. The verb can import the key into an internal key token in application storage or in key storage. This verb, which is authorized with a different control point than used with the Key_Export verb, allows you to limit the export operations to DATA keys as compared to the capabilities of the more general verb.

Specify the following:

- An external key token containing the source key to be imported. The external key token must indicate that a control vector is present; however, the control vector is usually valued at zero.
- Alternatively, you can provide the encrypted data key at offset 16 in an otherwise all X'00' key token. The verb will process this token format as a DATA key encrypted by the importer key and a null (all zero) control vector.
- An IMPORTER key-encrypting key under which the source key is deciphered.
- An external or null key token. The internal key token can be located in application data storage or in key storage.

The verb builds the internal key token by the following:

- Creates a control vector for a DATA key type in the internal key token, if the control vector in the external key token is zero. If the control vector is not zero, the verb copies the control vector into the internal key token from the external key token.
- Multiplies (decrypts) the key under the keys formed by the exclusive-OR of the key-encrypting key (identified in the importer_key_identifier) and the control vector in the external key token, then multiplies (decrypts) the key under keys formed by the exclusive-OR of the master key and the control vector in the internal key token. The verb places the key in the internal key token.
- Calculates a token-validation value and stores it in the internal key token.

This verb does not adjust the key parity of the source key.

Restrictions

None

Format

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSNBDDKM</td>
<td>Output</td>
</tr>
<tr>
<td>return_code</td>
<td>Integer</td>
</tr>
<tr>
<td>reason_code</td>
<td>Output</td>
</tr>
<tr>
<td>exit_data_len</td>
<td>Input/Output</td>
</tr>
<tr>
<td>ext_data</td>
<td>Input/Output</td>
</tr>
<tr>
<td>source_key_token</td>
<td>Input</td>
</tr>
<tr>
<td>importer_key_identifier</td>
<td>Input</td>
</tr>
<tr>
<td>target_key_identifier</td>
<td>Input</td>
</tr>
</tbody>
</table>

Parameters

For the definitions of the return_code, reason_code, exit_data_len, and exit_data parameters, see "Parameters Common to All Verbs" on page 1-10.

source_key_token

The source_key_token parameter is a pointer to a 64-byte string variable containing the source key to be imported. The source key must be an external key.

importer_key_identifier

The importer_key_identifier parameter is a pointer to a 64-byte string variable containing the (IMPORTER) transport key used to decipher the source key.

target_key_identifier

The target_key_identifier parameter is a pointer to a 64-byte string variable containing a null key token, an internal key token, or the key label of an internal key token or null key token record in key storage. The key token receives the imported key.

Required Commands

The Data_Key_Import verb requires the Data Key Import command (offset X'0109') to be enabled in the hardware.
Command Formalisation


The following transactions are used to translate PINs, that is, to decrypt an encrypted PIN block and re-encrypt it under another key. During the process, the PIN block format itself may be changed if desired.

These transactions are used on interchange networks. For example, if a foreign cardholder uses your ATM, the ATM typically makes his PIN up into a PIN block, encrypts it under a local key (a terminal PIN key) and sends it to you. You then use 'CA' to translate this PIN block from the local key to the acquiring bank's PIN key currently in use with the switch in question.

Note that you don't use a communication key to encrypt PINs on the ATM link, since then one of your application programmers could write code to determine the PINs of other institutions' cardholders - precisely the type of attack the system is designed to thwart.

TRANSACTIONS CA/CC - TRANSLATE A PIN

Host Message:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>4 Alpha</td>
<td>Returned unchanged</td>
</tr>
<tr>
<td>Trancode</td>
<td>2 Alpha</td>
<td>'CA' - translate from a local key to an interchange key</td>
</tr>
<tr>
<td>Source key</td>
<td>16 Hex</td>
<td>'CC' - translate from one interchange key to another</td>
</tr>
<tr>
<td>Destination key</td>
<td>16 Hex</td>
<td>CA: the local key encrypted under keys 14 &amp; 15</td>
</tr>
<tr>
<td>Destination PIN block</td>
<td>16 Hex</td>
<td>CC: any interchange key encrypted under keys 6 &amp; 7</td>
</tr>
<tr>
<td>Destination PIN format</td>
<td>2 Decimal</td>
<td>The interchange key (eq ANK) encrypted under keys 6 &amp; 7</td>
</tr>
<tr>
<td>Account no.</td>
<td>12 Decimal</td>
<td>'ZC' - the working key from the switch, encrypted under the ZCKX</td>
</tr>
</tbody>
</table>

Response Message:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>4 Alpha</td>
<td>Returned unchanged</td>
</tr>
<tr>
<td>Trancode</td>
<td>2 Alpha</td>
<td>'FA'</td>
</tr>
<tr>
<td>Source PIN block</td>
<td>16 Hex</td>
<td>Encrypted under master keys 4 &amp; 5</td>
</tr>
<tr>
<td>Key (ZCKX)</td>
<td>16 Hex</td>
<td>The working key from the switch, encrypted under the ZCKX</td>
</tr>
</tbody>
</table>

The rest of the transactions in this chapter are for translating various keys.

TRANSACTION FA - RECEIVE A WORKING KEY FROM A SWITCH

Host Message:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>4 Alpha</td>
<td>Returned unchanged</td>
</tr>
<tr>
<td>Trancode</td>
<td>2 Alpha</td>
<td>'FA'</td>
</tr>
<tr>
<td>Source PIN block</td>
<td>16 Hex</td>
<td>Encrypted under master keys 4 &amp; 5</td>
</tr>
<tr>
<td>Key (ZCKX)</td>
<td>16 Hex</td>
<td>The working key from the switch, encrypted under the ZCKX</td>
</tr>
</tbody>
</table>

Response Message:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>4 Alpha</td>
<td>Returned unchanged</td>
</tr>
<tr>
<td>Trancode</td>
<td>2 Alpha</td>
<td>'FA'</td>
</tr>
<tr>
<td>Return code</td>
<td>2 Decimal</td>
<td>One of the N3000MN return codes</td>
</tr>
<tr>
<td>PIN format</td>
<td>2 Decimal</td>
<td>'Destination PIN format' above</td>
</tr>
</tbody>
</table>

Note: the PIN formats in brief are 01: ANSI: 02: Document II: 03: IBM/ Document II: FMU, SSM and VISA currently use format 03.
Command ‘CC’ Formalised

TMK/PIN → WK

ZCMK_I

(TMK_I → WK_I)

(RAND) → TC

(CLEAR) → TC_I

LP

ZCMK
Command ‘CC’ Formalised

\[
\begin{align*}
U \rightarrow C & : \{ D \}_{WK1} , \{ WK1 \}_{WK} , \{ WK2 \}_{WK} \\
C \rightarrow U & : \{ D \}_{WK2}
\end{align*}
\]

\[
\text{formula: } \forall [X,Y,Z], \quad \Rightarrow ( \text{public}(X), \text{public}(Y), \text{public}(Z)) , \text{public}( \text{enc}(\text{enc}(\text{i}(wk),Z),\text{enc}(\text{i}(\text{enc}(\text{i}(wk),Y)),X)))
\]

\[
A = \! < ( \{ X \}Y , \{ Y \}WK , \{ Z \}WK ) > . \ ?(x).[ x = \{ X \}Y ] \\
B = ?(x) . \ \text{case } x \ \text{of } ( \{ w \}y , \{ y \}WK , \{ z \}WK ) \ \text{in } \! < \{ w \}z > . \ 0
\]
Command ‘CC’ Formalised

Cmd "CC_Data_Translate_Between_Interchange_Keys"
   Input ENC(ANY,ANY)
   Input ENC(WK,ANY)
   Input ENC(WK,ANY)
   Output ENC( DEC(WK,TWO) , DEC( DEC(WK,ONE) , ZERO ) )
End_Cmd
What else needs Formalising?

<table>
<thead>
<tr>
<th>Protocol-Speak</th>
<th>API-Speak</th>
</tr>
</thead>
<tbody>
<tr>
<td>already…</td>
<td>Command Definitions</td>
</tr>
<tr>
<td>Protocol Messages</td>
<td>Initial Knowledge</td>
</tr>
<tr>
<td>also…</td>
<td>N/A</td>
</tr>
<tr>
<td>Environment</td>
<td>Initial Goals</td>
</tr>
<tr>
<td>Attacker’s Abilities</td>
<td>Search Depth</td>
</tr>
<tr>
<td>Security Requirements</td>
<td></td>
</tr>
<tr>
<td># of Runs of Protocol</td>
<td>N/A</td>
</tr>
<tr>
<td># of Concurrent Runs</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Experiments with SPASS

• SPASS used in its capacity as a first order logic (FOL) theorem prover
• Predicate ‘public’ used to define commands and knowledge e.g.
  \[
  \text{public(input)} \Rightarrow \text{public}(f(input))
  \]
• Try to prove assertion \text{public(a_secret)}
• Sit back and wait…
SPASS Output...

---------------------------- SPASS START ------------------------
PING PING PING PING PING PING PING PING PING
PING PING PING PING PING PING PING PING PING
PING PING PING PING PING PING PING PING PING
PING PING PING PING PING PING PING PING PING
PING PING PING PING PING PING PING PING PING
PING PING PING PING PING PING PING PING PING

(and so on...)

--------------------------- SPASS END ---------------------------
Why SPASS is Unsuitable

- Insufficient runtime feedback
- Insufficient documentation
- Too many (unexplained) parameters to tweak – over 90 command line options
- SPASS correctly rediscovered every step and pair of steps of an attack, but could not discover the attack all in one go
- Unclear how to best re-express the problem
Other Tools?

- Wide choice (15-20 tools)
- Quality of documentation variable; many concentrate upon the tool’s application to a particular specialised problem
- Decided to learn from building my own tool …

“The only way to understand the wheel is to reinvent it”
Mimsearch Tool : Goals

- Learn about strengths and weaknesses of model checkers and theorem provers through comparison with a well understood example
- Improve ability to effectively use existing tools through better understanding of their internal working
- Determine the minimum complexity of models that can capture all known API attacks
- Create a tool powerful enough to reason about financial APIs, especially those using control vectors (ie. XOR)
Mimsearch Tool: Non-Goals

• Produce a tool more powerful than tool X
• Produce a better documented tool than tool X
• Produce an implementation for public release
Core Idea: Meet-in-the-Middle

Initial Knowledge

Hash Table

Hash Table

Lookup

Lookup

Initial Goals
“The British Museum”
“The British Museum”
Core Idea: Exploring the Museum
Core Ideas: Summary

- Attack the state space from both directions
- Minimise the number of heuristics: “Intelligent” search damages state space, and reduces chance of finding new attacks
- Accurately measure the state space: more accurate bounds mean more accurate assessment of security
- Native support for XOR and cryptographic primitives
- Understand the search: proper diagnostics should be available to all users
Implementation Tour

- Problem Specification
- Symbolic Term Manipulation Engine
- Search Threads
- Reverse Execution
- Hash Tables
- Distributed Computing Support
- Diagnostics
Problem Specification

Begin_Transaction_Set("4758-testset")

Begin_Cmd_List
End_Cmd_List

Begin_Reverse_Cmd_List
End_Reverse_Cmd_List

Begin_Atom_List
End_Atom_List

Begin_Attack

Begin_Initial_Knowledge_List
End_Initial_Knowledge_List

Begin_Initial_Goal_List
End_Initial_Goal_List

Search_Depth 3

End_Attack

Begin_Wat_List
End_Wat_List

End_Transaction_Set
Problem Spec Parser

- Transaction set language parsed at compile time using ugly mess of C preprocessor and compiler.
- Minimal effort put in because there are few transaction sets, they change rarely, and parsers are difficult.
Command Representation

\[ \text{U} \rightarrow \text{C} : \{ \text{D} \}_{WK1}, \{ \text{WK1} \}_{WK}, \{ \text{WK2} \}_{WK} \]
\[ \text{C} \rightarrow \text{U} : \{ \text{D} \}_{WK2} \]

Cmd "CC_Data_Translate_Between_Interchange_Keys"

  Input ENC(ANY,ANY)
  Input ENC(WK,ANY)
  Input ENC(WK,ANY)

  Output ENC( DEC(WK,TWO) , DEC( DEC(WK,ONE) , ZERO ) )

End_Cmd
More Example Commands

Begin_Cmd_List

// Ability_XOR
Cmd "Ability XOR"
Input ANY
Input ANY
Output XOR(ZERO,ONE)
End_Cmd

// Key_Part_Import
Cmd "Key_Part_Import"
Input ENC(XOR(KM,CV_IMP_PART),ANY)  // X, kek_part_token
Input ANY  // Y, new XOR value
Output ENC(XOR(KM,CV_IMP),XOR(DEC(XOR(KM,CV_IMP_PART),ZERO),ONE))
End_Cmd

// Key_Import
Cmd "Key_Import"
Input ENC(XOR(ANY,ANY),ANY)  // W, ext_token
Input ENC(XOR(KM,CV_IMP),ANY)  // X, kek_token
Input ANY  // Y, claimed_type
Output ENC(XOR(KM,TWO),DEC(XOR(DEC(XOR(KM,CV_IMP),ONE),TWO),ZERO))
End_Cmd

// Encrypt (NOT Ability_Encrypt)
Cmd "Encrypt"
Input ENC(XOR(KM,CV_DATA),ANY)
Input ANY
Output ENC(DEC(XOR(KM,CV_DATA),ZERO),ONE)
End_Cmd

End_Cmd_List
Symbolic Term Manipulation Engine

- Terms represented as trees of objects
- reduce, rehash, substitute and pattern match methods
- subtree hashes stored to speed up pattern matching
- This part is most sensitive to bugs – wrong manipulations will invalidate analyses
- Worked hard to remove bugs, but conservative implementation is not optimised for speed
Search Threads

- Search threads for both forward and backward search
- Pseudo-Random Number Generator seeded with strong(ish) random number representing each path, then called by all random decision making code owned by that thread
- Plausible command and argument selected (optionally according to likely reduction filters)
- Command executed as substitutions followed by a reduction to normal form
- Resulting term checked for match against hash table from search in the other direction
- Resulting term added (temporarily) to initial knowledge, and registered with hash tables
Reverse Execution Logic

TC1 \rightarrow O_{\text{Enter\_Clear\_TC}} \rightarrow \{TC1\}_{TC}

\text{enc}(tc,X)

\{TC1\}_{TC} \rightarrow \text{Reverse}_{O_{\text{Enter\_Clear\_TC}}} \rightarrow TC1

\text{dec}(tc,X)

chosen \rightarrow \text{calculated}
Reverse Execution: Dual Inputs

\[
\{TC_1\}_{TC} \xrightarrow{\text{Encrypt}} \{\text{data}\}_{TC_1} \\
\text{data} \xrightarrow{\text{Encrypt}} \{\text{data}\}_{TC_1}
\]

\[
\text{enc}(\text{dec}(tc,X),Y)
\]

\[
\{TC_1\}_{TC} \xrightarrow{\text{Reverse_Encrypt}} \text{data} \\
\{\text{data}\}_{TC_1} \xrightarrow{\text{Reverse_Encrypt}} \text{data}
\]

\[
\text{dec}(\text{dec}(tc,X),Y)
\]

- guessed
- chosen
- calculated
Problems with Reverse Execution

\[
\{TC1\}_{TC} \xrightarrow{\text{guessed}} \{\text{data}\}_{TC1} \xrightarrow{\text{guessed}} \text{dec(\text{dec(tc,X)},Y)} \xrightarrow{\text{calculated}} \text{data} \xrightarrow{\text{Faulty logic!}} \{\text{data}\}_{TC1} \xrightarrow{\text{calculated, and checked against original goal}} \text{calculated.}
\]
Entropy Limited Term Invention

\[ \text{enc}(\text{xor}(\text{km}, \text{data}), \text{key}) \]

(need to specify burn probability at different depths)
Hash Tables

• First implementations small sized (20MB), but current implementation uses 400MB per machine
• Windows 2000 behaves unpredictably when high demands are made on memory – caused lots of difficulty.
• What should the hash table store?
  – single bit markers (chosen)
  – partial storage of seed
  – storage of whole hash
• Birthday paradox makes false collisions (i.e. different terms with the same hash) very likely. Collisions require human intervention, so hash table must be as big as possible if system is to stay up unattended for more than a few minutes
Distributed Computing Support

- Manual logon of 50 machines takes about 15 mins
- Client/Server architecture between control machine and search machines using persistent TCP/IP connections
- Control machine connects to searchers, and collates diagnostic and result information (main communications workload), and routes it on to the graphical user interface.
- Control machine will later take responsibility for routing information for distributed hash tables. Searchers will probably form fully connected mesh
Diagnostics

- Watch display for progress towards known attack (total hits & hit rate)
- Hash table growth statistics
- Statistics combined across machines
- Complexity & search rate reports
- Runtime command line interface (esp. for debug)
## Watch Display

<table>
<thead>
<tr>
<th>Function</th>
<th>(1) Hits</th>
<th>Rate</th>
<th>(2) Hits</th>
<th>Rate</th>
<th>(3) Hits</th>
<th>Rate</th>
<th>(4) Hits</th>
<th>Rate</th>
<th>ET</th>
<th>WC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forens - Test</td>
<td>123</td>
<td>3</td>
<td>174</td>
<td>2</td>
<td>126</td>
<td>2</td>
<td>741</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Test</td>
<td>156</td>
<td>1</td>
<td>163</td>
<td>1</td>
<td>167</td>
<td>1</td>
<td>159</td>
<td>1</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Test</td>
<td>204</td>
<td>1</td>
<td>163</td>
<td>1</td>
<td>159</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Reverse - Goal

<table>
<thead>
<tr>
<th>Function</th>
<th>(1) Hits</th>
<th>Rate</th>
<th>(2) Hits</th>
<th>Rate</th>
<th>(3) Hits</th>
<th>Rate</th>
<th>(4) Hits</th>
<th>Rate</th>
<th>ET</th>
<th>WC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reverse</td>
<td>252</td>
<td>2</td>
<td>323</td>
<td>2</td>
<td>244</td>
<td>2</td>
<td>182</td>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Reverse</td>
<td>340</td>
<td>1</td>
<td>273</td>
<td>1</td>
<td>20</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reverse</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Reverse</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Reverse</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Statistics
Control & Configuration
## Results

<table>
<thead>
<tr>
<th>Attack</th>
<th>Complexity</th>
<th># Commands</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSM XOR to null key</td>
<td>$2^{66.1}$</td>
<td>5</td>
<td>5 mins</td>
</tr>
<tr>
<td>VSM type-cast</td>
<td>$2^{38}$</td>
<td>3</td>
<td>&lt;1 sec</td>
</tr>
<tr>
<td>4758 CCA type-cast</td>
<td>$2^{57}$</td>
<td>5</td>
<td>30 secs</td>
</tr>
</tbody>
</table>
Scalability and Future Limits

• Should be scalable up to about $2^{80}$ search path space
  – requires several terabytes for hash table
  – equivalent to two $2^{40}$ searches in each direction
  – relies upon continued success with reverse command execution

• Larger computer cluster could be used – over 1,000 machines in entire PWF, only 50 in practical laboratory

• FPGA hardware technology? Compile transaction set into hardware search machine?
Conclusions

• Security APIs are amenable to analysis for several sorts of attack
• The British Museum algorithm is alive and well
• Birthday attacks are an extremely useful tool
• Many more interesting problems can be brought within range of current formal analysis techniques by applying engineering know-how
• We need to expend more effort measuring the difficulty of problems
  – Question: Can the complexity bounds of a random search through an API be narrowed in polynomial time?"
• We need to develop instinctive understanding of complexity consequences as new transaction sets are written, or existing ones are formalised (complexity theoretic editor?)
More Information

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

Academic paper by Feb 2003?