Analyzing Complex Systems
The BlackBerry Case

FX of Phenoelit
Step 1

Getting the big picture
Why Big Picture?

- You might not know every aspect of the target
  - WYSIWYG is an intuitive but poor choice
  - WYSIWYG is probably where the focus of the defending side was

- The bigger the picture (system), the more clearly you need to identify the promising attack vectors
  … unless your organization has a three letter acronym and you got unlimited time on your hands
Why Big Picture? II

- You might not know what resources you will need
  - Hardware
  - Software
  - Infrastructure & Accounts
  - Tools
- Getting what you need might take time
- Trying to get it might have other consequences
  - Can you afford to invest money? How much?
  - Can you afford to cross legal lines?
  - Can you afford your target to know it’s under attack?
  - Do you care?
Before anything else, make sure you identified all components in the game. Don't miss non-obvious components.
Big Picture II

- Abstraction of the big picture helps to identify key areas to look at
- Split the picture into it’s major components
Break down the primary components of the system you are looking at:

- Handheld devices
- Mobile Network (i.e. GSM)
- RIM Network
- Internet based communication
- BlackBerry Enterprise Server
- BlackBerry Enterprise Server Connectors
- BlackBerry Management Tools
Reclassify the key elements in common terms:

- Handheld devices = Embedded system, proprietary hardware, RTOS, Java
- Mobile Network = 2.5/3G GSM style infrastructure
- RIM Network = unknown, likely IP based
- Internet based communication = Proprietary IP based Protocols
- BlackBerry Enterprise Server and Connectors = Windows based server software, closes source
- BlackBerry Management Tools = Windows based client/server software
Big Picture V – Accessibility

- Accessibility of the components
  - Handheld devices
    → doable, $666 per device
  - Mobile Network
    → hard, illegal
  - RIM Network
    → doable, illegal
  - Internet based communication
    → doable, requires access to a working installation
  - BlackBerry Enterprise Server and Connectors
    → easy, see IDA
  - BlackBerry Management Tools
    → easy, see IDA
Estimate the impact of a successful attack

- Handheld devices
  - Information disclosure, potentially remote control of single user

- Mobile Network
  - Redirection of communication endpoints

- RIM Network
  - Full control over the infrastructure, being RIM

- Internet based communication
  - Impersonation of RIM or BlackBerry Server, brute force attacks

- BlackBerry Enterprise Server and Connectors
  - Code execution on host OS, owning of a centrally placed server system in corporate networks

- BlackBerry Management Tools
  - Modification of policies, sending messages to everyone, may be installing software on handhelds (see Handheld devices)
Big Picture VII

Ease of access

Impact

little

BES Mgmt
Handheld
IP Protocols

Massive

BES
RIM Network
Mobile Network

Phenoelit
Step 2

Getting the details right
Diving into Details

- When you got the big picture completed, the details are what matters most

- The details decide:
  - How hard it will be to find an attack
  - What you need
  - How feasible the attack is
  - How (il)legal the attack is
Handheld devices

- Simulation environment available
- Developer SDK available
  - Current version is for Java
  - Old version is for C
    - Obviously more interesting (no sandbox)
    - Only available for US and Canadian developers
- Desktop Software available
- Third party code available
  - What do the 3rd party products do?
  - What does this tell you about the powers of the API?
Protocols

- How many communication channels are used?
- Who initiates the communication, who can?
- What underlying protocols are used (i.e. are they connection oriented or connection-less)?
- How much encapsulation is used?
  - Multiple levels of encapsulation indicate a tree structure of code handling the payload.
  - Flat protocols indicate a single massive protocol parser.
- How variable is the protocol design?
Server Software

- How is the software designed?
  - User-land, Service or Kernel?
  - Security Context and required privileges?

- What building blocks is the software made of?
  - Which handle user input?
  - How is the user input transformed before handled by this component?
  - Who developed the component?
  - What coding style was used?
  - What programming language was used?
  - Where is the interesting stuff stored?
Things to look at for details:

- **History**
  - How old is the component compared to the overall scenario?
  - Where does this component come from? What did the first release do, what does the latest?
  - Was there any major rewrite?
  - Check the press releases.

- **Documentation**
  - What are the setup requirements in administration guides?
  - What are the troubleshooting procedures recommended?
  - What are the troubleshooting procedures people actually use?

→ Take what you read in publications, press releases, documentation and forums as a hint, not a fact!
Step 3

Work
Work...

Plop

Beer!

This sucks!

!$@*#$
BlackBerry!

Phenoelit

int i,j,k;
i=atoi(in);
j=i<<2;
k=i/j;
Step 4

Results: The Handheld
First things first: strip it!

7290 naked (back view)
First things first: strip it more!

7290 naked (front view)
Handhelds

- Used to be 386, turns out it’s an ARM (C SDK fairly useless since it’s for 386)
- Different RTOS Kernels, some run KADAK AMX 4, others run RIM proprietary code. Every model is different.
- Binary images with hardware near code
- Loadable modules as PE/COFF DLLs linked against the RIMOS.EXE main binary
Handheld JVM

- Java Virtual Machine loaded as largest binary module (jvm.dll)
  - CDLC 1.1, MIDP 2.0
  - Java Vendor is RIM
- Limited set of J2ME classes
  - Reflection API missing 😞
- Device control via RIM classes
  - Java applications are almost useless without RIM class support
Code Signing

- **Java Application signature**
  - To use RIM classes
  - Signs a hash of the JVM binary (.jar)
  - $100 to be paid by credit card
  - Suspicion: Collection of a list of all platform binary’s hashes in case they become malware
  - News Flash: Stolen Credit Cards exist
  - Replacing the class loader doesn’t work 😞

- **Firmware image signature**
  - Checked in Loader (see your debugger 😊)
  - Something is checked while device is loading 😞
It’s not a Siemens, but …

- Browser Issue when parsing .jad Files:
  - long name for MIDlet-Name or -Vendor
    - Exception thrown by the dialog
    - Uncaught, modal dialog left over
    - Browser toast, everything else still works
    - Soft- or Hard-Reset don’t work
      (solution: denial all power to the device)
- RIM says it’s fixed in 4.0.2
Other things not tried yet

- Find the JTAG connectors
- Bluetooth on BlackBerry
- JVM bugs
- Reversing Images
- Figuring out checksums
- Loader.exe should be able to read memory contents from the device as well (credit: mark@vulndev.org)
Step 5

Results: The Protocols
Server Relay Protocol

- Encapsulation protocol inside IPv4
  - Simple header
  - Multiple string or integer payload chunks in TLV (type, length, value) format
# Server Relay Protocol

## Header

<table>
<thead>
<tr>
<th>Byte</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Protocol Version</td>
</tr>
<tr>
<td>2</td>
<td>Function</td>
</tr>
<tr>
<td>3-6</td>
<td>Length of the entire message</td>
</tr>
</tbody>
</table>

## Chunk Format

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Byte</th>
<th>Value/Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>1</td>
<td>0x53 / type identifier</td>
</tr>
<tr>
<td></td>
<td>2-5</td>
<td>/ length of the string</td>
</tr>
<tr>
<td></td>
<td>6-x</td>
<td>/ content</td>
</tr>
<tr>
<td>Integer</td>
<td>1</td>
<td>0x49 / type identifier</td>
</tr>
<tr>
<td></td>
<td>2-5</td>
<td>/ value</td>
</tr>
<tr>
<td>Opcodes</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>RETURN</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>DISCONNECT</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>RECEIVE</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>STATUS</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>SEND</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>CONNECT</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>REGISTER</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>DATA</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>PAUSE</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>RESEND</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>CANCEL</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>STATUS_ACK</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SUBMITTED</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>DATA_ACK</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>RESUME</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>STATE</td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>RESET</td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>INFO</td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>CONFIG</td>
<td></td>
</tr>
<tr>
<td>FC</td>
<td>PING</td>
<td></td>
</tr>
<tr>
<td>FD</td>
<td>PONG</td>
<td></td>
</tr>
<tr>
<td>FE</td>
<td>SRP Error</td>
<td></td>
</tr>
</tbody>
</table>
Session Setup

1. Client → Server: System ID
2. Server → Client: Server challenge
   - Server Random seed + Random value + Ctime
3. Client → Server: Client challenge
   - Client Random seed + Random value + Service string
4. Server → Client: HMAC_SHA1 (Client challenge)
   - Transformed SRP Key used for HMAC_SHA1
5. Client → Server: HMAC_SHA1 (Server challenge)
6. Server → Client: init request
7. Client → Server: init data

Successfully implemented a Server and a Client in Perl
Gateway Message Envelope

- Encapsulation protocol for messaging
- Routing Information of the message
  - Source (Server Identifier or PIN)
  - Destination (Server Identifier or PIN)
  - Message ID
- Comparable to information in Email headers

Diagram:
- IPv4 Header
- IP Payload
- SRP Header
- SRP Encapsulation
- GME Header
- GME Payload
## Generic Message Encapsulation (GME)

The GME Format is as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol version</td>
<td>1 byte</td>
</tr>
<tr>
<td>Source</td>
<td>Type = 1 byte [0x10]</td>
</tr>
<tr>
<td></td>
<td>Length = 1 byte</td>
</tr>
<tr>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>Destination</td>
<td>Type = 1 byte [0x20]</td>
</tr>
<tr>
<td></td>
<td>Length = 1 byte</td>
</tr>
<tr>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>Terminator</td>
<td>1 byte = [0x00]</td>
</tr>
<tr>
<td>Message ID</td>
<td>4 byte</td>
</tr>
<tr>
<td>Application Identifier</td>
<td>Type = 1 byte [0x50]</td>
</tr>
<tr>
<td></td>
<td>Length = 1 byte</td>
</tr>
<tr>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>GME command</td>
<td>1 byte</td>
</tr>
<tr>
<td>Content length</td>
<td>Variable length integer</td>
</tr>
<tr>
<td>Terminator</td>
<td>1 byte = [0x00]</td>
</tr>
</tbody>
</table>
Application Layer

- Application layer identifier in clear text
  - CMIME = message
  - CICAL = calendar updates
  - ITADMIN = key updates, IT policies, etc.
- Email, calendar and others encrypted
- PIN messages in clear text
  - Documented behavior, but very hard to find
### Application Layer

#### CMIME Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption Type</td>
<td>1 byte</td>
</tr>
<tr>
<td>Key ID</td>
<td></td>
</tr>
<tr>
<td>Terminator</td>
<td>1 byte [0x00]</td>
</tr>
<tr>
<td>Session Key</td>
<td>32 Byte</td>
</tr>
<tr>
<td>Terminator</td>
<td>1 byte [0x00]</td>
</tr>
<tr>
<td>Message identifier</td>
<td>1 byte [0x19]</td>
</tr>
<tr>
<td>Message</td>
<td></td>
</tr>
</tbody>
</table>
Application Layer Payload

- AES or DES encryption
- Key ID in clear text
- Session Key encrypted with device key
- Message compressed and encrypted with session key
- Successfully implemented packet dump message decryption script with given key in Perl
A word about the crypto

- Crypto library is FIPS certified
- Phe-no-crypto-people
- Implementation looks good in the disassembly
- No obvious key leak problems when activating devices via USB
- Crypto may be re-Weis-ed (as in Rüdi)
Traffic analysis

- Traffic analysis based on header possible
  - Sender PIN known
  - Recipient PIN known
  - Message content type known
  - Timing known

- In combination with (il)legal interception of SMTP email traffic
  - Email address to PIN mapping
Protocol based attacks I

- SRP Session setup with someone else’s key and SRP ID
  - Legitimate key owner disconnected when modifying data in the session startup
  - New connection from either source results in the other one begin dropped
  - After 5 reconnects in less than a minute, the key is locked out. No BlackBerry service until RIM resolves the issue.

- RIM Authentication keys are not viewed as secrets by most companies
  - Slides and screenshots with keys can be found by your favorite search engine
Protocol based attacks II

- SRP String Type length field
  - Integer overflow leads to Access Violation when initially decoding packets
  - Negative value -5 causes infinite decoding loop
  - Affects at least router and enterprise server

```
.text:0042B11B     OR      eax, edx
  ; EAX is length field (now in Host Byte Order) after \x53
.text:0042B11D     LEA     edi, [eax+ecx]
  ; ECX is current position pointer in packet
.text:0042B120     CMP     edi, ebx
  ; position + length > overall_length ?
.text:0042B122     JG      short loc_42B19F
  ; jump to failure handling code if position + length points
  ; past the packet
```

Spam anyone?

- PIN messages not encrypted
  - Therefore, no crypto code needed
- SRP authentication key can be used to PIN message anybody, not only your users
  - Any legitimate or stolen SRP key can be used
- Simple Perl script sufficient to send messages to any PIN
  - Sequentially sending it to all PINs from 00000000 to FFFFFFFF?
  - Spoofing sender might be possible (no evidence that it is not) – turns out it is!
Step 666

Results: The Enterprise Server
BlackBerry Enterprise Server

- BES Architecture
- SQL Database
- The beauty of updates
- Code style and quality
- Interesting libraries
- Attachment Service Special
<table>
<thead>
<tr>
<th>BES Accounts</th>
<th>Logon Locally</th>
<th>Logon as Service</th>
<th>Local Admin</th>
<th>Exchange RO Admin</th>
<th>Exchange MailStore Admin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Account</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>Server Mgmt Account</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>User Admin Account</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
<td>✔️ ✔️ ✔️</td>
</tr>
</tbody>
</table>
SQL Database

- MS SQL Server with user authentication
  - No integrated authentication for Domino
- Tables for individual messages and mails
- Table with SRP Authentication Key
  - The most important secret between the BES and RIM stored in clear text
- Table with Device Keys
  - Previous, current and new/pending key
  - Can be used for traffic decryption
- Default account: SA / (no password)
The beauty of updates

- RIM updates the BES
  - Service Packs
  - HotFixes
  - Release and fix notes tend to be extremely entertaining

- Hackers should update BES
  - SABRE BinDiff
  - Free .pdb debug information files in some fixes. Many thanks to RIM.
Code style & quality

- Massive C++ code
  - By-the-book pattern implementations
  - Large classes
  - STL
  - Harder to reverse engineer

- Surprisingly good
  - STL helps a lot
  - “If in doubt, check again” approach
    - A.k.a. select, select, select, recv
  - But generally using signed integers, although mostly correct
Interesting Libraries – reverse engineered

- Microsoft IStream classes
  - Parsing of Microsoft Office documents
- Microsoft MSHTML4 engine
  - Parsing of HTML documents
- MSXML SDK
  - Installed, no idea what for.
  - MSXML used for Sync server.
- Arizan parsing product
  - Central parsing engine
  - Parsing of PDF and Corel WordPerfect
Interesting Libraries – reverse engineered

- **Zlib 1.2.1**
  - ZIP attachment handling is copy & paste contrib/unzip.c (almost binary equal)
  - Known bugs 😊
    - 1.2.3 is current

- **GraphicsMagick 1.1.3**
  - ImageMagick spin-off
  - Fully linked, including debug code and …
open source ➔ source audited

- ...supported and compiled in file formats in GraphicsMagick:
  - ART, AVI, AVS, BMP, CGM, CMYK, CUR, CUT, DCM, DCX, DIB, DPX, EMF, EPDF, EPI, EPS, EPS2, EPS3, EPSF, EPSI, EPT, FAX, FIG, FITS, FPX, GIF, GPLT, GRAY, HPGL, HTML, ICO, JBIG, JNG, JP2, JPC, JPEG, MAN, MAT, MIFF, MONO, MNG, MPEG, M2V, MPC, MSL, MTV, MVG, OTB, P7, PALM, PBM, PCD, PCDS, PCL, PCX, PDB, PDF, PFA, PFB, PGM, PICON, PICT, PIX, PNG, PNM, PPM, PS, PS2, PS3, PSD, PTIF, PWP, RAD, RGB, RGBA, RLA, RLE, SCT, SFW, SGI, SHTML, SUN, SVG, TGA, TIFF, TIM, TTF, TXT, UIL, UYVY, VICAR, VIFF, WBMP, WMF, WPG, XBM, XCF, XPM, XWD, YUV
Source audit: Use the Code Luke!

- GraphicsMagick ChangeLog:
  - “coders/avi.c, bmp.c, and dib.c: applied security patch from Cristy.”
  - “coders/tiff.c (TIFFErrors): Prevent possible stack overflow on error.”
  - “coders/psd.c (ReadPSDImage): Fix stack overflow vulnerability”
  - “coders/tiff.c (ReadTIFFImage): Fix overflow while computing colormap size.”
- Odd own format strings in arbitrary text fields of any image format
  - Expect image comment \texttt{100\%tonne} to become \texttt{100C:\Windows\temp\bbaAA.tmponne}
Reverse Engineering + Source results I

- Heap overflow in TIFF parser
  - Integer overflow in image data memory requirement allocation
  - Allocation of small (0) memory block for image data
Reverse Engineering + Source results II

- Heap overflow in PNG parser
  - `#define PNG_USER_WIDTH_MAX 1000000L` does not prevent integer overflows
- Overflow in memory allocation counter
- Allocation of small (1MB) memory block for image data decompression
More Open Source results

- Zlib museum in PNG parser
  - Paying attention?
    - Version 1.2.1 used, inclusive decompression bug
  - PNG image data is zip compressed
  - Heap overflow when decompressing image data
  - Your arbitrary BugTraq example works

- Interestingly enough, known libPNG bugs are fixed
BES Architecture Attack

- Internet
- RIM
- Router
- Dispatcher
  - MDS
  - Alert
  - Attachment Service
  - Policy Service
- Key
- Obtain Keys
- Connector
  - Notes / Exchange / GroupWise
  - MS SQL

(int i, j, k; i = atoi(in); j = i <= 2; k = 1/i;)

Phenoelit

[Diagram showing relationships between components]
BES Architecture must be
Separate Attachment Service issue

- Remote control
  - TCP port 1999
  - Unauthenticated XML
- Query
  - Version
  - Statistics
  - Number of processes
- Set number of processes
  - Recommended test values: 0, 20000
Step 7

Mopping up
Vendor communication

- Vendor and users of the system in question can greatly profit from the analysis done
  - Well planned analysis yields unique insights in the architecture and the effectiveness of fixes

- RIM
  - re-work of attachment image parsing

- RIM customers
  - Moving BES and Database in separate DMZ
  - Separation of the attachment service
Finalizing

- Print offensive T-Shirts
- Meet with everyone involved
- Get drunk
- Send greets to random people, such as:

Phenoelit, 13354, Halvar Flake & SABRE Security, THC, all@ph-neutral, hack.lu, Scusi, mark@vulndev.org, Frank Rieger, the Eschschloraque Rümpschrümp, mac, t3c0, trash, the darklab@darklab.org people and Ian Robertson from RIM