

# Analysis of an undocumented network protocol

Jean-Baptiste Bédune  
Fred Raynal



November 7, 2008

- 1 Overview and first clues
  - Architecture
  - Administration interface - Configuration file
  - Binaries classification
- 2 Protocol analysis
  - Make the program speak
    - Verbose mode
    - Debugging mode
    - First intuitions
  - First streams, first intuitions
  - Packets dissection
  - Finalisation of the analysis of the format
- 3 Short security analysis
  - Development of a client
  - Vulnerability research
  - Exploiting the protocol

# Introduction

- Analysis of a network protocol: methodology?
- Substantial software: how to make analysis easier?
- Vulnerability research:
  - looking for “classic” flaws
  - ... or exploiting design errors?

# Architecture

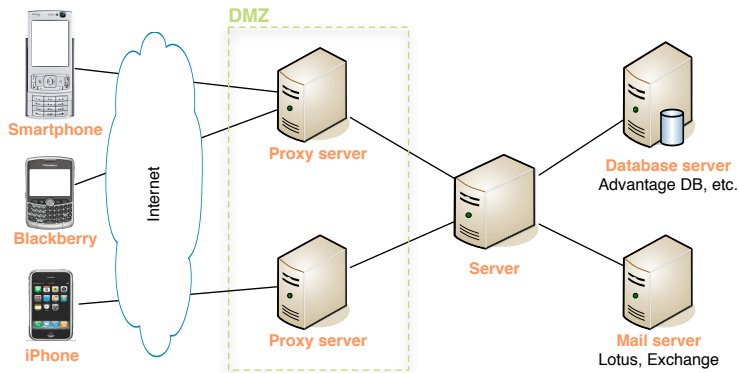
## General architecture

- A central server, one or more proxies opened on the internet
- Communication with mail servers (Lotus, Exchange) and DB servers
- Proprietary client on numerous platforms (Palm, RIM, Smartphone, etc.)

## Features

- Sending and receiving e-mails
- Retrieve Groupware data
- Manipulating files, registry
- Sending device configuration, etc.

# Global scheme



# Administration interface

- Done using a single tool
- Action list for each type of device:
  - Reception of Groupware data: contacts, e-mails, calendar, etc.
  - Time synchronization
  - Operations on registry, files
- Ability to handle groups (“Profiles”)
- Several authentication schemes: LDAP, Notes, Radius, others (SDK available)

Configuration stored in plaintext XML, in the Data directory

No information about a specific user in this file

# Binaries classification (1)

## Binaries classification

Goal: identify central binaries, in order to restrict analysis to a minimal amount of code.

All the binaries are in the `Bin` directory

- 20 executables
- 180 libraries

→ Complex understanding, no way to analyse all the modules.  
Classification from libraries information: filename, metadata, exported functions, etc.

## Binaries classification (2)

5 categories:

- Runtime : C++ runtime, system libraries;
  - **Communication: between the server, mail servers and databases;**
  - **Message processing;**
  - **Authentication;**
  - File conversion (to read Office documents on mobile devices).
- 2/3 of the files won't be analyzed.

# Verbose mode

## Make the program speak

Goal: retrieve information about the program internals by activating “indiscreet” modes, often present in substantial softwares for debugging purposes.

- Documented options
- ... or not
- Reading logs: logs are displayed in *Log Viewer*. Useful for administration, but no information for protocol comprehension.
- Trace options: *Trace Viewer* shows status, state changes, events on the server. More options are displayed when adding a given field in an XML configuration file (documented).

# Trace : example

```
20070726T120937 * * Startup: Computer - 'JB'. Successfully initialized data tables
20070726T120937 * * Admin settings change request processed* * Logging started
20070726T121019 * * Startup: Computer - 'JB'. Successfully initialized data tables
20070726T121019 * * Admin settings change request processed* * Logging started
20070726T151256 * * Startup: Computer - 'JB'. Successfully initialized data tables
20070726T151257 * * Admin settings change request processed
20070726T151306 jb b7e2226aaf4041f9a6a3ed06de94082a ProcessTMStartMessage:
Action started for pre 4.5 client. Client version: (null)
20070726T151309 jb b7e2226aaf4041f9a6a3ed06de94082a ProcessTMFinishMessage:
Actionfinished for pre 4.5 client. Client version: (null)
20070726T151309 jb b7e2226aaf4041f9a6a3ed06de94082a ProcessTMStartMessage:
Action started for pre 4.5 client. Client version: (null)
20070726T151312 jb b7e2226aaf4041f9a6a3ed06de94082a ProcessTMFinishMessage:
```

Action finished for pre 4.5 client. Client version: (null)

→ Once more, not really useful.

# Seeking debugging modes

## Debugging mode

- Used by developers to test their product
- Often present in final versions, yet hidden
- Generally easy to find (command line options, etc.)
- Give **a lot of** information about the program

Here, adding:

```
<msglog trace="y" />
```

in file `routerconfig.xml`.

All packets sent and received by the server are sorted and stored in the file `routertrace.txt`.

Option found when analysing `router.dll`

# Debugging mode: example

Block for a stored message:

## routertrace.txt

```
Time: 2007/07/25 20:33:09:453
Source: tm://0bdbd28e5eded64f86c1c7c2b919acbf?devid=b7e2226aaf4041f9a6a3ed06de94082a
&userid=jb
Destination: prof://?ID=-803747957&devid=b7e2226aaf4041f9a6a3ed06de94082a
&devtype=windows&gid=StarterGroup&sessionid=0bdbd28e5eded64f86c1c7c2b919acbf
&starttime=20070725T183309Z&tmdest=server&userid=jb&v=4

MessageBodyFile: tm-prof-026.xml
```

## tm-prof-026.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<ProfileMgr Revision="1185360030" Type="profreq">
<Profile Current="Y" Name="Plain" Revision="1185360030"></Profile>
</ProfileMgr>
```

In addition, routerconfig.xml is very interesting:

### routerconfig.xml

```
<handler schema="notes" lib="notesaddon.dll" func="HandleMessage"/>
<handler schema="notesadapter" lib="notesadapter.dll" func="HandleMessage"/>
<handler schema="noteslistener" lib="noteslistener.dll" func="HandleMessage"/>
<handler schema="playlist" lib="playlistmgr.dll" func="HandleMessage"/>
<handler schema="prof" lib="prof.dll" func="HandleMessage"/>
```

...

(44 entries)

router.dll seems to route all the messages received by the server, sending them to the various libraries lib via the fonction func.

# First intuitions

## Content of messages

- Clients are identified by:
  - a login `userid` ;
  - a session identifier `sessionid` ;
  - a device identifier `devid` ;
  - a device type `devtype` ;
  - a group `gid`.
- Actions are identified by:
  - a source URI;
  - a destination URI;
  - data (here in XML format).

Message are routed by `router.dll` according to the destination URI

# Dumping traffic

## Capturing exchanges

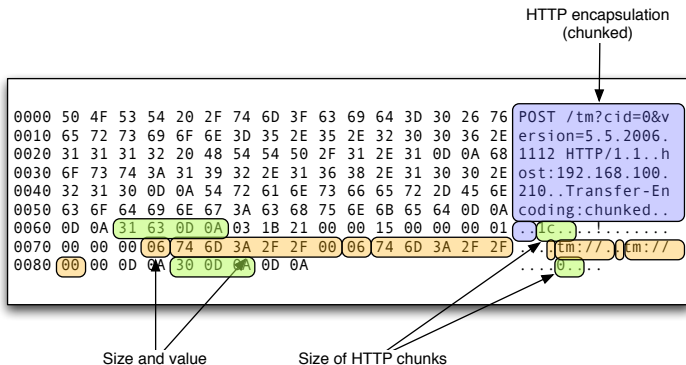
Captures are done using `tcpdump` or `Wireshark`

- without action: simple authentication, exchange is as simple as possible
- with various actions: make actions vary to analyze differences: number of packets exchanged, content, etc.

Modifying security options to simplify the analysis:

- SSL connections are an option
  - No SSL
- Communications are encrypted by default
  - No encryption

## Following your intuition



→ Some characteristic structures can already be recognized.

# Packets dissection

## Packets format

Packets are:

- built before being sent;
- dissected once received.

## Method of analysis

- hooking `send` and `recv` to retrieve construction and dissection functions
- reverse engineering key functions
- tests, intuitions to guess the role of unidentified functions

# Reverse engineering

## Tedious task

High level language (C++). Many constructors, destructors, virtual methods, etc.

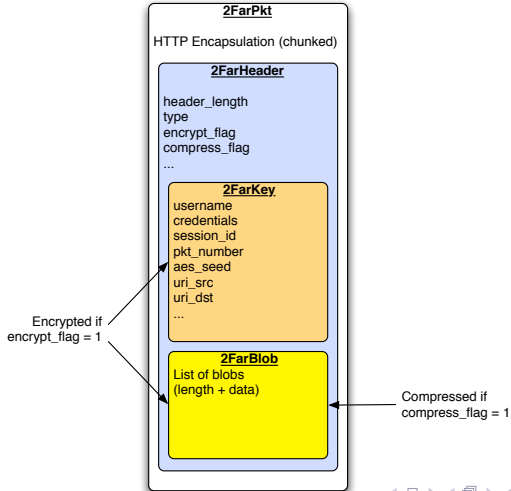
→ a line-by-line analysis is tedious.

## Graph visualization

Advantages:

- Viewing the path executed while a packet is processed
- Forcing a function to enter a path it never takes to determine its role

# Simplified packets structure



# Compression and encryption

## Compression

Use of `zlib`

Signatures creation, and calls location

## Encryption

Use of RSA BSAFE Crypto-C

Signatures creation, and calls location

- Key establishment: RSA OAEP
- Packet encryption: AES CFB
  - Key derived from a seed extracted from the packet just received (`aes_seed`)

Reverse engineering on Crypto-C functions to develop compatible functions

# Packets routing

Message = source URI, destination URI and data (OneBlob).  
Each URI starts with `tm` (*Terminal Manager?*)

→ `tm.dll` processes these messages, and sends them to other modules.

## File transfer

- Destination URI:

```
tm://ft/download?ftid=11223344&sessionId=xxxxxxx&tmcmd=start&tmdest=server
```

- Intern routing:

```
<handler schema="tm" lib="tm.dll" func="HandleMessage"/>
```

```
<handler schema="ft" lib="filetransfer.dll" func="HandleMessage"/>
```

→ Processed by `tm.dll` which rewrites the message, then routed towards `filetransfer.dll`.

# Creation of a client

## Python client

Goal: “talk” with the server, and send it packets that can be easily manipulated, to understand and exploit the protocol.

## Method

- Use of scapy, packet manipulation program written in Python.
- Definition of classes to build and dissect exchanged packets.
- Class as flexible as possible to be able to correct possible errors.

→ Design takes more time at first, but finally saves a considerable amount of time.

## “Classic” vulnerabilities

### Programming errors

- Not our main goal.
- Problems detected while examining the protocol.

### Pre-auth denial of service

- Chunked HTTP encapsulation: chunks with negative sizes are not correctly handled.
- A thread is created for each new connection.
- Newly created thread crashes, the server service notices it, logs its state and shuts down.

# “Classic” vulnerabilities

## Programming errors

- Not our main goal.
- Problems detected while examining the protocol.

## Pre-auth denial of service

- Chunked HTTP encapsulation: chunks with negative sizes are not correctly handled.
- A thread is created for each new connection.
- Newly created thread crashes, the server service notices it, logs its state and shuts down.

## “Classic” vulnerabilities (2)

### Post-authentication denial of service

- A very big compressed buffer is sent.
- Denial of service during decompression. Server load raises to 100%.
- Works only with some buffers, we don't try to see why.

### Post-authentication denial of service

- The size of some fields is encoded in VLQ (*Variable Length Quantity*) format.
- Bytes representing the size are copied in a fixed size buffer until their higher significant bit is set.
- No code execution, but creating an infinite loop is simple.

## “Classic” vulnerabilities (2)

### Post-authentication denial of service

- A very big compressed buffer is sent.
- Denial of service during decompression. Server load raises to 100%.
- Works only with some buffers, we don't try to see why.

### Post-authentication denial of service

- The size of some fields is encoded in VLQ (*Variable Length Quantity*) format.
- Bytes representing the size are copied in a fixed size buffer until their higher significant bit is set.
- No code execution, but creating an infinite loop is simple.

# Protocol exploitation

## How handlers work?

- Connection profile loaded by the server, sent to clients.
- Information on loaded connections in a proprietary database, open to the server and its handlers.
- Profiles are not stored in the database, and not open to handlers.
- Handlers are totally disconnected from user profiles: they only verify if the user is correctly logged by requesting the database.

→ Handler = feature. What is the security policy at a handler level?

# Example: file transfer

## Administration options

- Transfers from/to the server
- Files to include/exclude according to their extension
- Source and destination of files

## Security?

- Configuration sent to the clients... but only as a rough guide.
- No check at the server side, all the files accessible by the server (which has SYSTEM rights) can be downloaded.

# Example: file transfer

## Administration options

- Transfers from/to the server
- Files to include/exclude according to their extension
- Source and destination of files

## Security?

- Configuration sent to the clients. . . but only as a rough guide.
- No check at the server side, all the files accessible by the server (which has SYSTEM rights) can be downloaded.

# Example: Trojan (1)

## Method

- A new handler is developed. It will be uploaded in the server exploiting the file transfer handler.
- The router configuration (`routerconfig.xml`) is retrieved. Our handler is added.
- Actions are sent by our client: code execution, and system control.

## Example: Trojan (2)

### Features

- Retrieve server private keys: they are located in the registry, encrypted with a fixed key.
- Dump of database, to retrieve `sessionid` of other logged on users, and the hash of their credentials.
- Traffic listening, SAM, remote shell, etc.

Presentation

Introduction

Overview and first clues

Protocol analysis

Short security analysis

Development of a client

Vulnerability research

Exploiting the protocol

# Questions?