How hackers grind an MMORPG: by taking it apart!

An introduction to reverse engineering network protocols

Rink Springer

rink@rink.nu

https://rink.nu

https://github.com/zhmu

Outline

- Introduction: Runes of Magic
- Motivation
- Capturing and analysing packets
- Improving analysis: romdump
- Improving capturing: romproxy
- Improving manipulation: openrom
- Legal side of things
- Q&A

Thanks!

- Technical advice: luxtau
- FDB_Extractor2: McBen
- 31C3 “Cyber Necromancy”: Joseph Tartaro and Matthew Halchyshak
- Legal advice: Arnoud Engelfriet
- IDA Freeware: Ilfak Guilfanov
- OllyDbg: Oleh Yuschuk

Wait, 'MMORPG'? What?

- **M**assively **M**ultiplayer **O**nline **R**ole-**P**laying **G**ame
- Interactive online game, generally fantasy-themed
- Subscription-based or free-to-play
- Goal is to improve your virtual character (levels, gear, quests)
- Socially involved: lots of team-play ('guilds')
- Designed to suck up a lot of your time [*]

[*] Though usually by playing the game

Runes of Magic (1)

- Free-to-play MMORPG, launched in 2009
- Taiwanese developer (Runewaker Entertainment)
- German publisher (GameForge, formerly Frogster)

Images from http://board.us.runesofmagic.gameforge.com/board272-runes-of-magic-us/board276-community/board321-erebos-pve/76815-grats-noto-on-killing-first-boss-of/ and https://howtoloseyourlifetoanmmorpg.wordpress.com/

Runes of Magic (2)

- 3 races, 10 classes, dual-class system
- 9 crafting professions
- 35+ instances
- Extensive item customization
- Pet system, elite skills, house

In other words, a lot to figure out...

Runes of Magic (3)

Images from the game, slightly edited

Motivation

- Curiosity – what makes an MMORPG game tick ?
- Privacy/security – it runs on my PC, on my network …
- The time is now – much harder once the servers are offline
- … much more fun than actually *playing* the game!

Capturing packets (1)

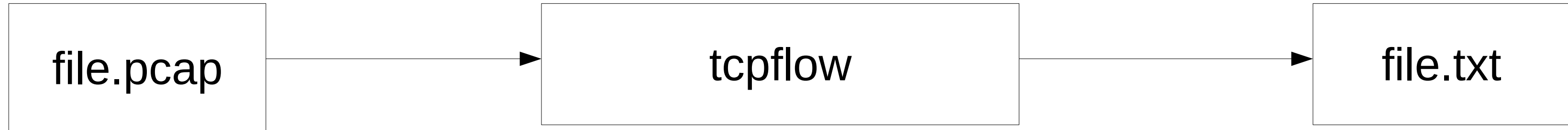
- Local NAT host with 2 network interface cards
- tcpdump to capture
- tcpflow to decode TCP streams

Not ideal, but good enough when starting

Capturing packets (2)

```
# tcpdump -i eth1 -w file.pcap 'net 198.51.100.0/24'
```

```
$ tcpflow -cDr file.pcap 'not port 80' > file.txt
```




```
$ cat /tmp/file.txt
```

```
198.51.100.40.21002-203.0.113.70.48137:
000: 1600 0000 034f 67ff 0000 0000 200e 64ca 149e 2722 d5d3       .....Og..... .d...'"..

203.0.113.70.48137-198.51.100.40.21002:
000: 5401 0000 0255 ad00 0100 0000 0200 0000 2121 2121 0000 0000   T....U..........!!!!....
...

198.51.100.40.21002-203.0.113.70.48137:
000: 1400 0000 02a0 b700 0100 0000 0400 0000 2500 0000             ................%...
```

Packet dumps (1) – initial login capture

```
                          198.51.100.40.21002-203.0.113.70.48137:
000: 1600 0000 034f 67ff 0000 0000 200e 64ca 149e 2722 d5d3        .....Og..... .d...'"..

                          203.0.113.70.48137-198.51.100.40.21002:
000: 5401 0000 0255 ad00 0100 0000 0200 0000 2121 2121 0000 0000   T....U..........!!!!....
018: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
030: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
048: 0000 0000 0000 0000 f7f4 42f8 f7f3 f3f7 f4f5 f4f2 f0f0 44f4   ..........B...........D.
060: 44f3 f346 f8f0 43f4 f6f6 f344 43f5 45f5 0000 0000 0000 0000   D..F..C....DC.E.........
078: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
090: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
0a8: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
0c0: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
0d8: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
0f0: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
108: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
120: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
138: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000   ........................
150: 0101 e3ee                                                      ....

                          198.51.100.40.21002-203.0.113.70.48137:
000: 1400 0000 02a0 b700 0100 0000 0400 0000 2500 0000             ................%...
```

Packet dumps (2) – login 'aaaa' / 'aaaa'

Server -> Client (22 bytes)

```
@00: 16 00 00 00 03 4f 67 ff 00 00 00 00 20 0e 64 ca
@10: 14 9e 27 22 d5 d3
```

Client -> Server (340 bytes)

```
@00: 54 01 00 00 02 55 ad 00 01 00 00 00 02 00 00 00
@10: 21 21 21 21 00 00 00 00 00 00 00 00 00 00 00 00
@50: f7 f4 42 f8 f7 f3 f3 f7 f4 f5 f4 f2 f0 f0 44 f4
@60: 44 f3 f3 46 f8 f0 43 f4 f6 f6 f3 44 43 f5 45 f5
```

Server -> Client (20 bytes)

```
@00: 14 00 00 00 02 a0 b7 00 01 00 00 00 04 00 00 00
@10: 25 00 00 00
```

Packet dumps (3) – login 'aaaa' / 'bbbb'

Server -> Client (22 bytes)

```
@00: 16 00 00 00 03 1c 34 ff 00 00 00 00 03 71 d4 24
@10: af e6 37 66 c4 7a
```

Client -> Server (340 bytes)

```
@00: 54 01 00 00 02 7e d6 00 01 00 00 00 fe 00 00 00
@10: 5f 5f 5f 5f 00 00 00 00 00 00 00 00 00 00 00 00
@50: 32 33 3e 3f 38 34 2f 43 30 2f 44 32 44 3e 31 31
@60: 2d 2d 43 37 30 3f 33 3e 31 42 37 43 32 42 38 30
```

Server -> Client (20 bytes)

```
@00: 14 00 00 00 02 83 9a 00 01 00 00 00 04 00 00 00
@10: 63 00 00 00
```

Logging in (1) - could it be… MD5 ?!

ASCII recap

`'a' = 61, 'A' = 41, '0' = 30, '4' = 34, '7' = 37`

Password seems to be

```
f7 f4 42 f8 f7 f3 f3 f7 f4 f5 f4 f2 f0 f0 44 f4
44 f3 f3 46 f8 f0 43 f4 f6 f6 f3 44 43 f5 45 f5
```

Let's just give it a try

`MD5('aaaa') = 74 b8 73 37 … 66 3d c5 e5`

Logging in (2) – big surprise... + XOR

Key could be: <u>`20`</u> `0e 64 ca 14 9e 27 22 d5 d3`

Assuming: `f7 f4 42 f8 → 37 34 42 38` and `61 → 21`

```
20: 0010 0000   0010 0000
f7: 1111 0111 → 0011 0111
f4: 1111 0100 → 0011 0100
42: 0100 0010 → 0100 0010
f8: 0110 0001 → 0010 0001
```

However… we know that `00` encodes to `00` …

```
f7 + 20 xor 20 = 37
f4 + 20 xor 20 = 34
42 + 20 xor 20 = 42
```

Packet dumps (4) – noticing patterns

Key packet

```
16 00 00 00 03 e7 ff ff 00 00 00 00 ...
```

Data packets

```
54 01 00 00 02 28 80 00 01 00 00 00 ...
58 02 00 00 02 8a e9 01 02 00 00 00 ...
58 02 00 00 02 16 77 02 03 00 00 00 ...
58 02 00 00 02 82 e5 03 04 00 00 00 ...
...
58 02 00 00 02 11 7e 08 09 00 00 00 ...
58 02 00 00 02 b9 28 09 0a 00 00 00 ...
58 02 00 00 02 c8 2f 00 0b 00 00 00 ...
58 02 00 00 02 94 fd 01 0c 00 00 00 ...
```

So we know the length, flags, key number and sequence
OllyDbg / IDA learns there are two checksums (header and data)
… both so trivial we could have done it ourselves

Packet dumps (5) – what about these?

The following sequence keeps showing up at seemingly random intervals:

Server → Client

`10 00 00 00 04 13 26 ff 00 00 00 00 00 00 00 00`

Client → Server

`10 00 00 00 08 17 2e ff 00 00 00 00 00 00 00 00`

Simple keepalives

Combined with sequence number, perhaps they intended to use UDP?

Packet dumps (6) – recognising numbers

Assuming we're moving around in the game

Client → Server

```
0000: 82 00 00 00 11 00 ff ff 2c 00 00 00 18 00 00 00
0010: af 07 01 00 5f c3 7c c5 f6 80 71 43 1b 88 01 c6
0020: 1a 62 57 42 06 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 48 42 00 00 00 00
```

Little endian makes floats easy to recognize

```
5f c3 7c c5 = -4044.21
f6 80 71 43 =   241.50
1b 88 01 c6 = -8290.03
1a 62 57 42 =    53.85
00 00 48 42 =    50.00
```

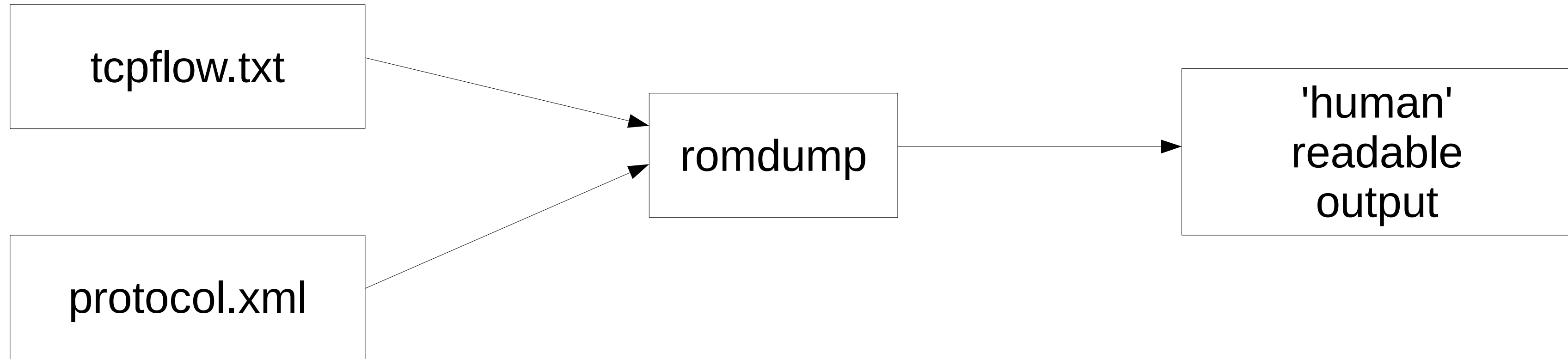
Decoding packets

After lots of staring:

- Packet header understood
- Identified types of packets
- This protocol uses subtypes
- Manually decoding things is boring (and error-prone)

So, let's write a tool to help us!

romdump (1) - basics



- Parses a tcpflow output file
- Decrypt the contents (knows protocol basics)
- Works on a per-packet basis
- Apply definitions from XML
- Supports output filtering

romdump (2) – XML definitions

```
<packet name="Login" source="client">
  <field type="u32" name="type" fixed_value="0x02" />
  <field type="string" name="accountname" length="64" />
  <field type="string" name="password" length="256" />
  <field type="u32" name="unknown" />
</packet>
```

```
<packet name="LoginFailure" source="server">
  <field type="u32" name="type" fixed_value="0x04" />
  <field type="u32" name="error" />
</packet>
```

```
>>> 2: 198.51.100.40:49436 -> 203.0.113.70:21002 len 328 flag 0x2 key 0 seq 1
packet 'Login'
 'type': 0x2
 'accountname': 'aaaa'
 'password': '74B87337454200D4D33F80C4663DC5E5'
 'unknown': 0x2e230101
```

```
@000: 02 00 00 00 61 61 61 61 00 …
@044: 37 34 42 38 37 33 33 37 34 …
@144: 01 01 23 2e
```

```
>>> 3: 203.0.113.70:21002 -> 198.51.100.40:49436 len 8 flag 0x2 key 0 seq 1
packet 'LoginFailure'
 'type': 0x4
 'error': 0x65
```

```
@00: 04 00 00 00 65 00 00 00
```

romdump (3) – subpackets

Definition

```
<packet name="ClientRequest" source="client">
  <field type="u32" name="type" fixed_value="0x82" />
  <field type="u16" name="target" />
  <field type="u16" name="unknown" fixed_value="0xffff" />
  <field type="length" name="length" />
  ...
  <subpacket name="PlayerMove">
      <field type="u32" name="type" fixed_value="0x18" />
      <field type="u32" name="objectid" />
      <field type="float" name="x" />
      <field type="float" name="y" />
      <field type="float" name="z" />
      <field type="float" name="angle" />
      <field type="u32" name="move_type" />
      <field type="float" name="dist_x" />
      <field type="float" name="dist_y" />
      <field type="float" name="dist_z" />
      <field type="u32" name="unknown1" />
  </subpacket>
  …
</packet>
```

romdump output

```
packet 'ClientRequest'
 'type': 0x82
 'target': ? <0x11>
 'unknown': 0xffff
 'length': 30 subpacket 'PlayerMove'
  'type': 0x18
  'objectid': 0x107af
  'x': -4044.21
  'y': 241.50
  'z': -8290.03
  'angle': 53.85
  'move_type': 0x6
  'dist_x': 0.00
  'dist_y': 0.00
  'dist_z': 50.00
  'unknown1': 0x0
```

Packet

```
@00: 82 00 00 00 11 00 ff ff 2c 00 00 00 18 00 00 00
@10: af 07 01 00 5f c3 7c c5 f6 80 71 43 1b 88 01 c6
@20: 1a 62 57 42 06 00 00 00 00 00 00 00 00 00 00 00
@30: 00 00 48 42 00 00 00 00
```

romdump (4) – features

Roughly in order of implementation:

- Hex-dump of packets
- Filtering by packet type

Definitions for:

- Constants
- Enumerations
- Structured types
- Arrays (only for primitive types)
- Transformations
- Static annotations (quest id, item id)
- Dynamic annotations (object id)

Wait… quest and item ids?

- Link an item or quest and see what is inside
  http://runesofmagic.gamepedia.com/ItemLink

- Item database websites
  http://www.runesdatabase.com (offline)
  http://www.rom-welten.de/database/

- Extract from the datafiles yourself
  https://github.com/McBen/FDB_Extractor2
  https://github.com/zhmu/romdb

  Make decoding things much easier as you'll recognize them sooner.
  Hex dumps of data help a lot!

Recognizing compressed data

- One packet looked like to contained player info
- … yet, name was partially present, same goes for item id's
- Data looked like random gibberish
- *random* … usually compressed...

- Used OllyDbg to set read breakpoint; turned out it *was* compressed!
- Used IDA, C and a lot of patience to learn the algorithm
- Implemented as transformation in romdump ('rompack')
- Yet no idea about ~99% of the 100kB structure…

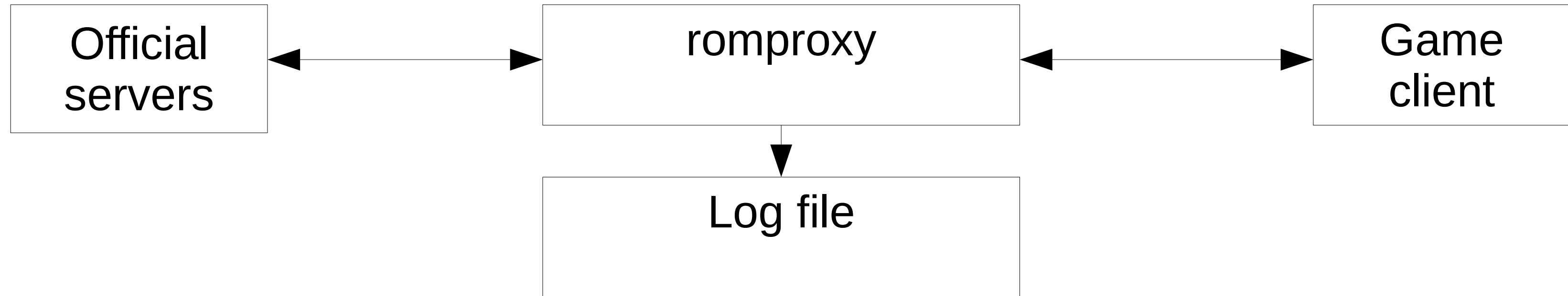
Advanced capturing

- Tcpdump isn't ideal [*]
- Tcpflow isn't ideal either [*]
- MMORPG's tend to have multiple servers or backends
- Redirect to another server, etc
- Packet dump text file becomes a mess

So… let's write a proxy!

[*] … but are both great tools!

romproxy



- Proxies between client and real servers
- Game lets you override main server in INI file, so easy to use
- Logs traffic as desired
- Rewrites redirects to itself
- Handles keepalives internally
- “Undetectable”
- Could do other interesting things [*]

[*] We're not interested in those things. We want to learn the protocol!

You can only go so far...

- The protocol is pretty chatty
- Logging in and entering game world is ~400 kB of data!
- How do commands interact?
- What can be left out?
- There's a lot of unknown data fields...
- Observation: you need to influence data

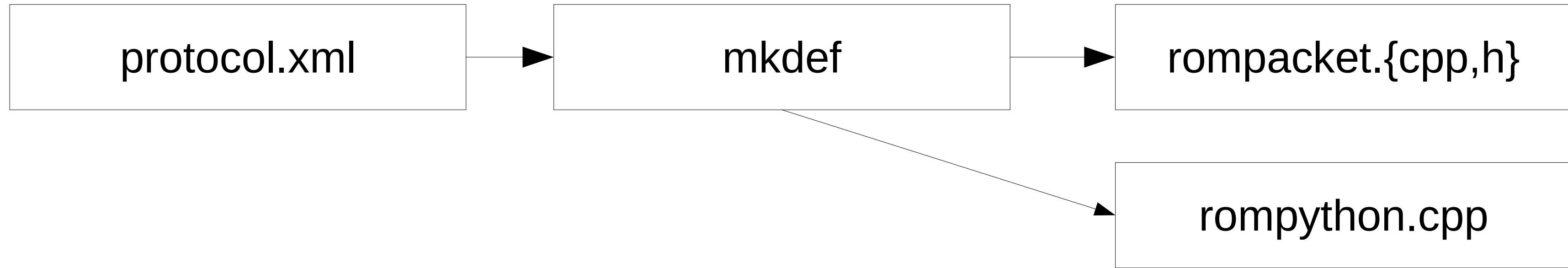
Idea: let's write our own server!

openrom (1)

- Acts as a Runes of Magic server
- C++, SQLite
- Logging capabilities
- Embedded Python
- Telnet console to execute scripts

But, `Broadcast ( [ 12, 34, 56, 78, … ] )` wouldn't be very helpful, now would it?

openrom (2) – wait, we've got protocol.xml



- Use protocol.xml to generate code to send packets!
- … and why not use it to parse them as well
- Heck, let's generate Python bindings

openrom (3) – Python to the rescue

```
<packet name="ServerResponse" source="server">
  ...
  <subpacket name="DisplayYellowText">
    <field type="u32" name="type" fixed_value="0xa1" />
    <field type="u32" name="objectid" annotation="objectid" />
    <field type="u32" name="unknown2" />
    <field type="u32" name="unknown3" />
    <field type="u32" name="unknown4" />
    <field type="string" name="message" min_length="1" length="65536" />
  </subpacket>
```

```
$ telnet 0 12345
rompacket.ServerResponse_DisplayYellowText(objectid=0x4010755, unknown2=1, unknown3=0xffffff80, unknown4=0x12, message='Hello world')
```

Makes deciphering commands *fun*!

openrom (4) – meanwhile, in the real world

- Patience, small steps, start by replaying data
- Observation: timing of response matters
- Buggy client doesn't really help

Initially, just get it working. No need to make it perfect [*]

[*] If you've ever played this game, you'll understand

openrom (5) – the demo

```
<struct name="object_appearance">
  <field type="u8" name="unk1" />
  <field type="u8" name="unk2" />
  …
  <field type="u8" name="unk31" />
  <field type="u8" name="unk32" />
</struct>
```

```
<packet name="ServerResponse" source="server">
  <field type="u32" name="type" fixed_value="0x87" />
  <field type="u16" name="target" enumeration="targets" />
  <field type="u16" name="unknown" fixed_value="0xffff" />
  <field type="length" name="length" />
  ...
  <subpacket name="ObjectAppearance">
    <field type="u32" name="type" fixed_value="0x1e" />
    <field type="u32" name="objectid" annotation="objectid" />
    <field type="object_appearance" name="appearance" />
  </subpacket>
</packet>
```

- Works with the latest official client [*]
- Will demonstrate how the scripting is used to learn about the game
- Bear with me while we get things started…

[*] But won't be demonstrated here

Runes of Magic protocol 101 (1)

- Servers: login → portal → game
- … almost everything has a specific command!
  Often, an event results in a lot of replies
- >200 packet types (about ~150 understood)
- Object-based: {Create,Delete,Move}Object
- Player characters: Object{Apperance,GearLook}
- Positioning, fighting, ... all have commands

Runes of Magic protocol 101 (2)

- Large 'character info' packet which mainly influences the UI (game world is completely object-based)
- Client knows which quests are pending
- Client security doesn't exist [*]
- Some information gathering (MAC address, GPU, OS)

[*] … I didn't dare bother with the server!

Now what? The legal stuff...

- I've spoken with several lawyers on how to proceed
- Most say: keep everything to yourself
- *However*, Arnoud Engelfriet was very helpful

- rom{dump,proxy} useful tools
- Protocol definitions help greatly to explain them
- Yet… own server is another matter [*]

- Advice: release dump/proxy/protocol to illustrate their usefulness
- Don't bother with server, can only cause problems (and how useful is it?)

[*] Even considering the state it is in

Thank you!

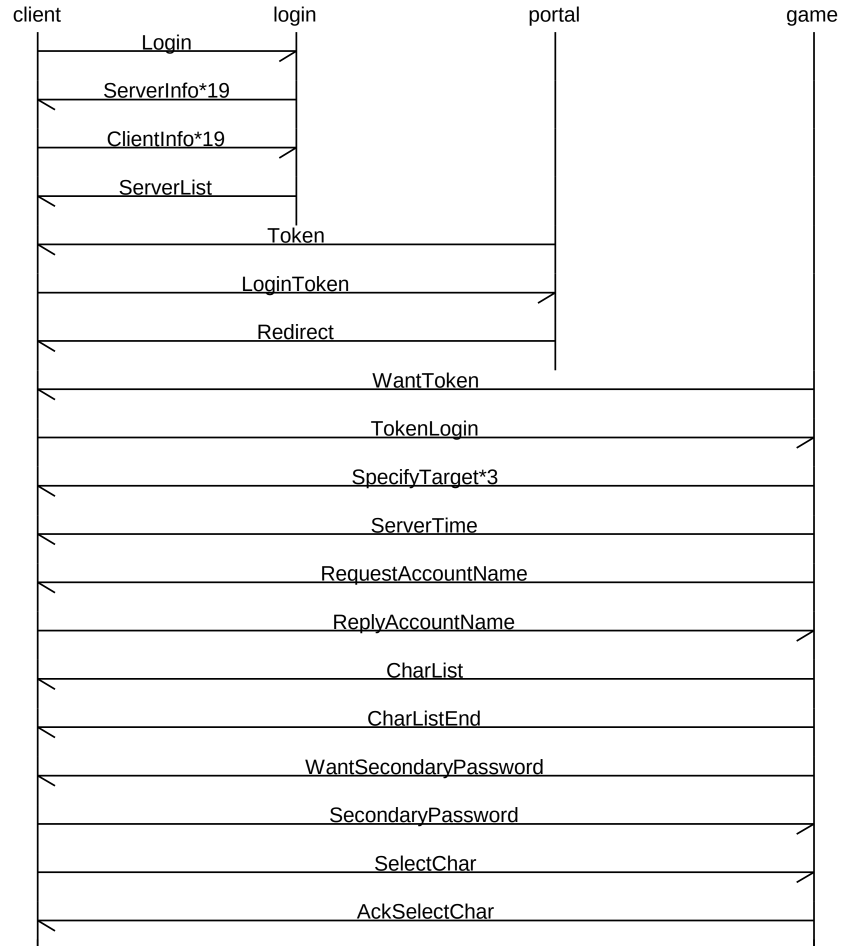
All things released so far: https://github.com/zhmu/openrom/

Any questions?

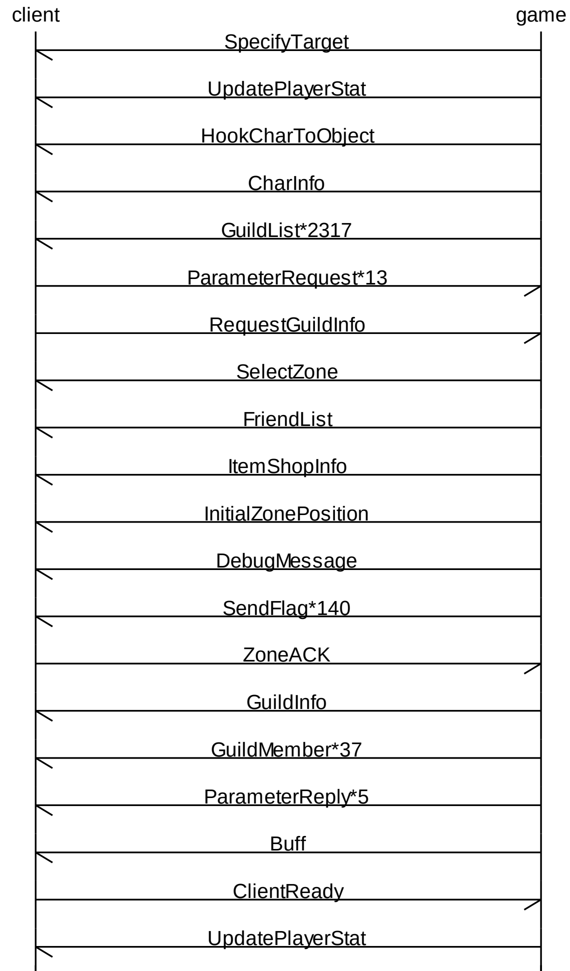
Or contact me after the talk:

Rink Springer
rink@rink.nu
https://rink.nu

The protocol (1) – logging in



The protocol (2) – game initalisation



Packet dump (5) – kinda, it's + XOR!

Key

`03 71 d4 24 af e6 37 66 c4 7a`

Account name

`5f 5f 5f 5f`

`5f + 03 xor 03 = 61 ('a')`

Password

`32 33 3e 3f …`

`32 + 03 xor 03 = 36 ('6')`
`33 + 03 xor 03 = 35 ('5')`

`MD5('bbbb') = 65 …`