Reverse-Engineering A Proprietary Game Server With Erlang

Erlang, the fear of game developers...

Loïc Hoguin - @lhoguin
Erlang Cowboy and Nine Nines Founder

Background

Why Reverse-Engineer?

- Curiosity
- Research the game
- Build your own server
  - Because you want a challenge
  - Because the official server is discontinued

Frowned Upon

- Few game companies understand
- Data mining is cheating
- Reverse-engineering is cheating
- Even if you don't take advantage of your knowledge

Anticheat Technology

- Online games feature anticheat technology
- Often instead of real security
- You have to bypass it, undetected
- It's OK, anticheat systems are full of flaws

Windows

- Online games usually only run on Windows
- Wine doesn't work because of anticheat programs
- A few steps require a Windows box
- If you're doing any gaming you probably have one

Phantasy Star Universe (PSU)

- SEGA game released in 2006
- US version shutdown in early 2010
- JP version still running
- Protected by GameGuard (check files, memory, cheat tools...)
- TCP for patch server, SSL for login/game servers
- My example for this talk

Packet Logging

Undetected Logging

- Methods available can vary depending on the game
- Common methods:
  - Snooping
  - Hooking a function on packet receive/send
  - Man in the middle

Breaking Through PSU's SSL

- Find the client SSL certificate
- Try connecting to the server from Erlang
- Use hosts file to redirect the client to localhost
- Make the client connect to Erlang and redirect the packets to the server
- You just built a proxy for the game
- Packets going through the proxy are readable

Tee

- A command that redirects input to both standard output and file
- Make the proxy save to a file at the same time as redirecting

Proxy Hint

- You can always send the server whatever you want now
- You can read, modify, filter packets
- You can send any packet anytime
- You get more control than you would through the client
- Though you need to know the protocol first

Packet Analysis

Protocol

- First we need to figure out the general framing by hand
- Open the log file with a hex editor and find packet boundaries
- We got a packet size followed by a command number and the packet
- The protocol is 32bit

Spreadsheet Help

- Parse the file and aggregate the info to CSV files
- Figure out the field boundaries
- We got 8bit, 16bit, 32bit data, 32bit floats, ASCII and UCS2 strings
- Always take notes of what you are doing

Aggregate The Field Values

- Get a clear view of what values a field can take
- Sometimes the value never changes
- Knowing the values allow you to guess the field purpose
- Example: player ID, player level...
- It's actually not that hard to figure out most of them

Game Mechanics

- A few key values hide a lot of secrets
- Player ID identifies a player account
- Quest, zone, map and entry IDs identify the map you play
- Target ID identifies the player object in the zone
- Client and file analysis helps figure some of these out

Extracting Files

What Files?

- Quest and zone files
- Quest files identify the mission played
- Zone files define the scripts and objects in a set of area
- If we are to write a server, we must understand those

File Extraction

- The files aren't always named or identified
- Extract and tag them as properly as possible
- Ignore duplicates
- Get a good SSD for this, it can represent GBs of data

Files Analysis

Client Files Too

- Analyze both client and server files
- They share file formats
- PSU has an offline mode where more missions can be found
- Sometimes file formats can be found on Google, often not

Start With A Debugger

- First, remove the anticheat technology
- Load the game up to the game title
- Break at all file loadings (find them with a dissassembler)
- Press Enter, loading the login screen
- You now have the ASM for loading the file

Step By Step

- Tediously advance step by step to find the interesting functions
- The main archive format was encrypted using a blowfish variant
- It was also compressed using a custom LZE algorithm
- We got through their "security", let's extract the files

Concurrent Extraction

- Use Erlang to concurrently extract all the files you have
- This can take some time, but less than if you had to do it on 1 core
- Chances are your extraction code is wrong and doesn't handle edge cases

Continue With An Hex Editor

- Hex editing allow you to isolate values and group of values
- Problem: some files are just structs and arrays with pointers
- Pointers get converted to real memory addresses on load
- We still need to use a debugger to figure these out

File Parser

- We now have enough info to write a parser for all files
- We should make sure the parser gets values in the right range
- Pattern matching allow us to crash on unexpected values
- Also crash on values that don't seem to change

Concurrently Check Our Assumptions

- Parse all files concurrently with range checking
- If all files pass, then all our assumptions are verified
- Bonus: convert the files to readable formats

Prototype Server

Validating Protocol Assumptions

- Using the proxy would be too limited
- We need a valid implementation checked against the client

PSU's Protocols

- Patch, login and game servers
- Patch is a very simple TCP protocol
- Login and game are the same SSL protocols
- Login just redirects to the game server on successful auth

First Implementation

- Make use of the previously logged packets
- Take one log and just send all the packets unmodified
- Reach in-game and stop there
- Figure out the packet order
- Try modifying values and check that nothing went wrong

Trial And Error

- Figuring out values and testing them is a trial and error process
- We're developers, we're used to do this
- It gets easier when we properly reach in-game

Trial And Error

In-Game

- Open the menus, move the character, enter rooms
- In other words: make the client send packets!
- Note what packets are sent when you do something
- Find in the logs what is replied when it happens

Responses

- Same as before, start sending a logged packet
- Then figure out the values and test things out
- Write a function that does it for you for next times

Shell Testing

- You don't have to wait for client actions
- Use the shell to send packets directly from the server
- Make your character warp around!
- Test things out thoroughly

Warping Is Good

- Make sure to write a quick command to warp around
- Changing areas allow you to unstuck yourself
- The client doesn't do everything asynchronously

Feedback Loop

Lengthy Process

- We need early feedback
- We must not make the client disconnect
- Reconnecting makes us lose at least 1 minute!

Reloading

- Code reloading allows us to test fixes right away
- Data files can be reloaded too
- Client can be forced to reload an area through warping

Don't Crash In The Network Layer

- We must not kill the socket
- When something bad happens, print the error in the console!
- If a packet can't be parsed, print its hex representation!
- Tips: also print when something is parsed properly
- You can always crash after you finished working on the server

Still Trial And Error

- Someone can help by figuring out values in the client
- But this is still mostly trial and error
- Although it's much better thanks to Erlang

Demo

Questions?