All this text wasn’t written by me (Tiago a.k.a TigaxMT).

All of it was transcribed by me from HackerOne video lessons.

All credits and thanks go to them.

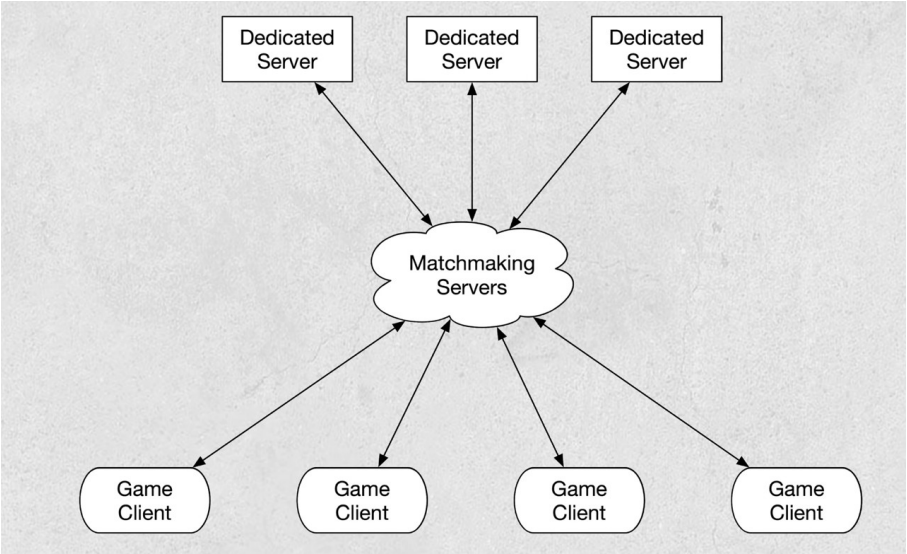
Game Hacking Basics

NETWORKING

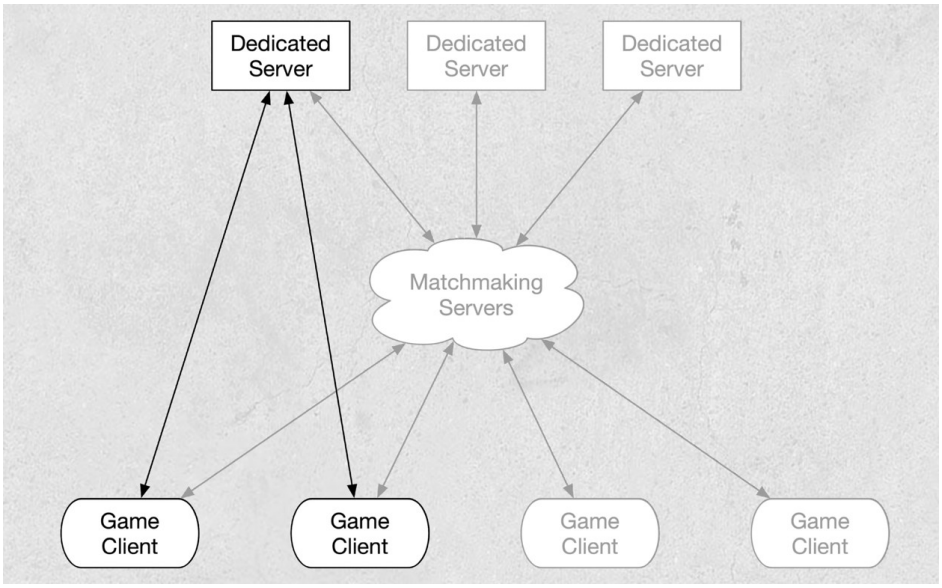
TERMINOLOGY

- Packets – Discrete chunks of data sent over the network
- Game Client – Either the game binary connected to a server or the computer on which it's running
- Server – A host to which either a game client or another server is connecting

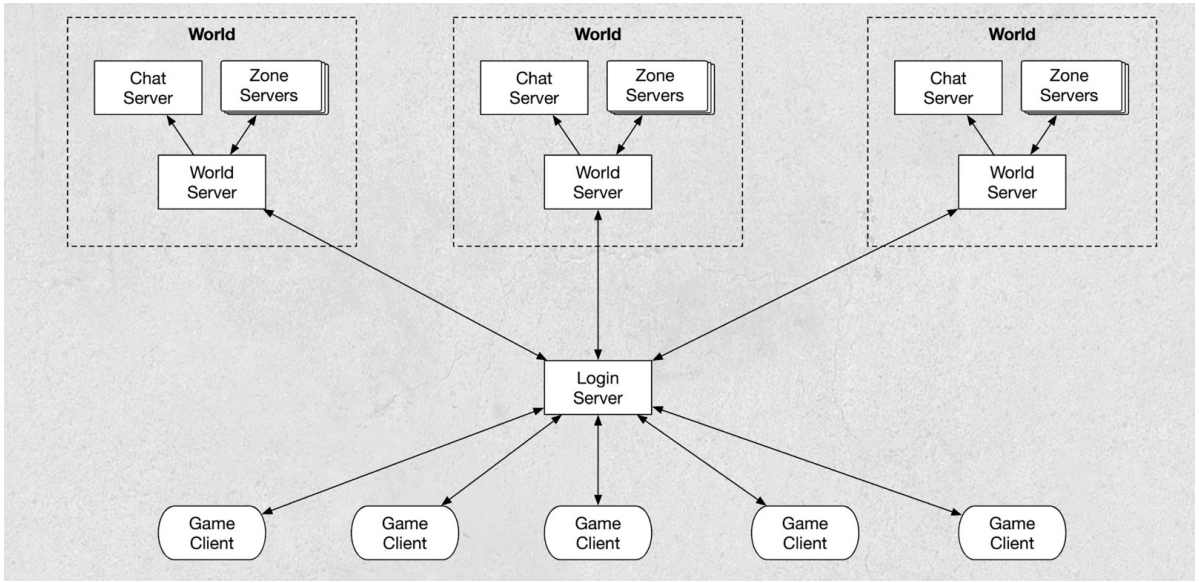
MATCHMAKING TOPOLOGY



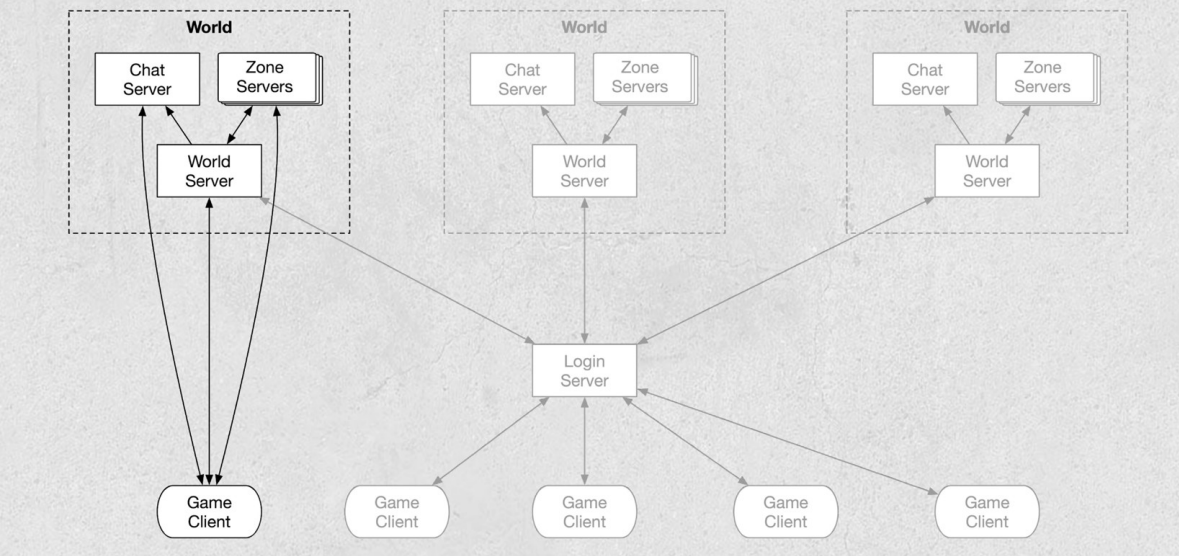
When the game started a dedicated server is chosen for a certain amount of players



MMO TOPOLOGY



When the player choose a world to play, the client disconnects to the Login Server and connects on the World Server



TCP VS UDP

Most of your traffic is intended to be received in the order it's sent, reliably retrying if a packet fails to be received by the other side. In most cases, this is done by building your protocol on TCP, the Transmission Control Protocol.

TCP carries a lot of overhead, though, and for a real-time game you may not want a a packet to be retried if it's been too long. UDP – User Datagram Protocol – doesn't ensure order, retries, or any of the things TCP provides.

There are not even any connections.

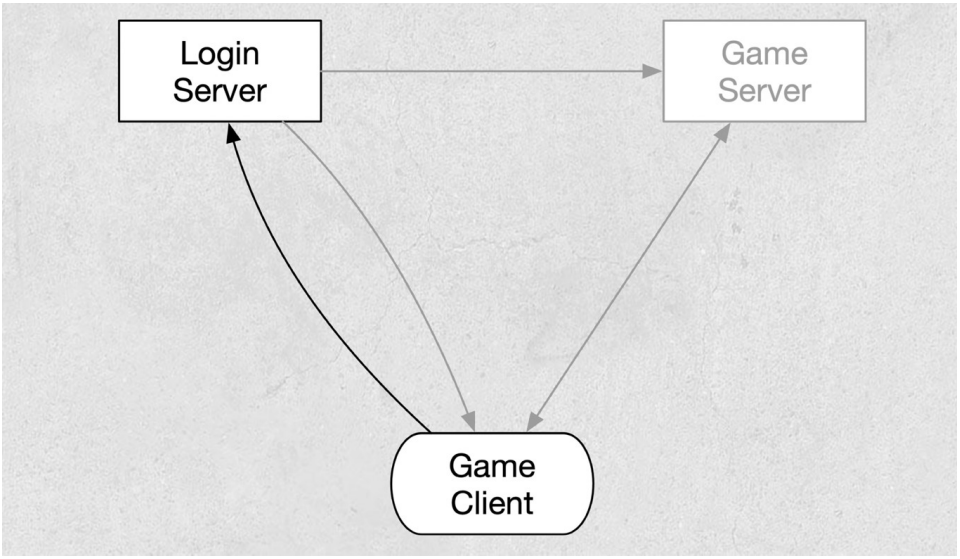
Fast-paced games utilize custom protocols built on top of UDP to ensure critical data is prioritized and makes it to its destination, without also carrying along any extraneous data.

PROTOCOL ENCRYPTION

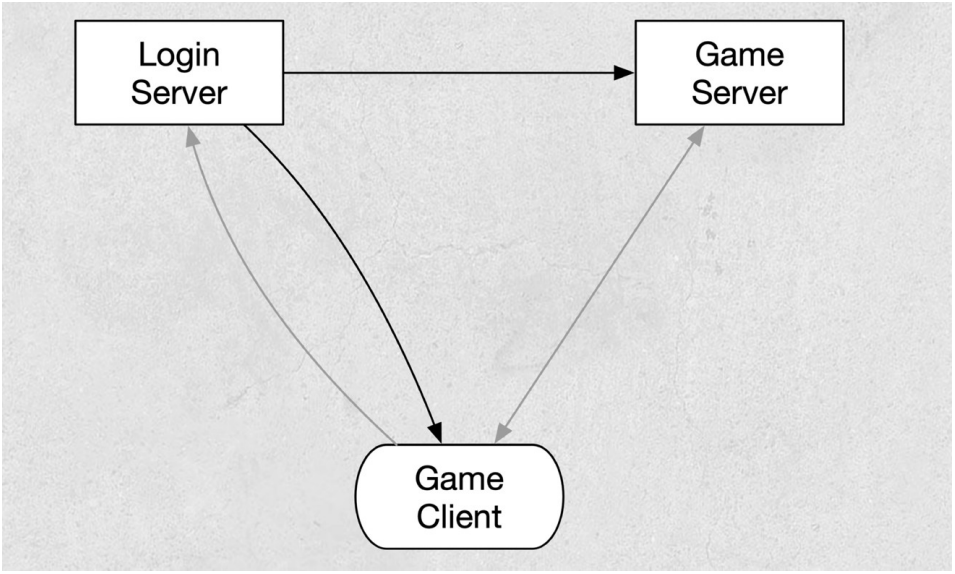
Games often include encryption for their network traffic, both to prevent sniffing or tampering by potential attackers and to make cheating more difficult.

This is usually very simple to keep overhead low and reliability high.

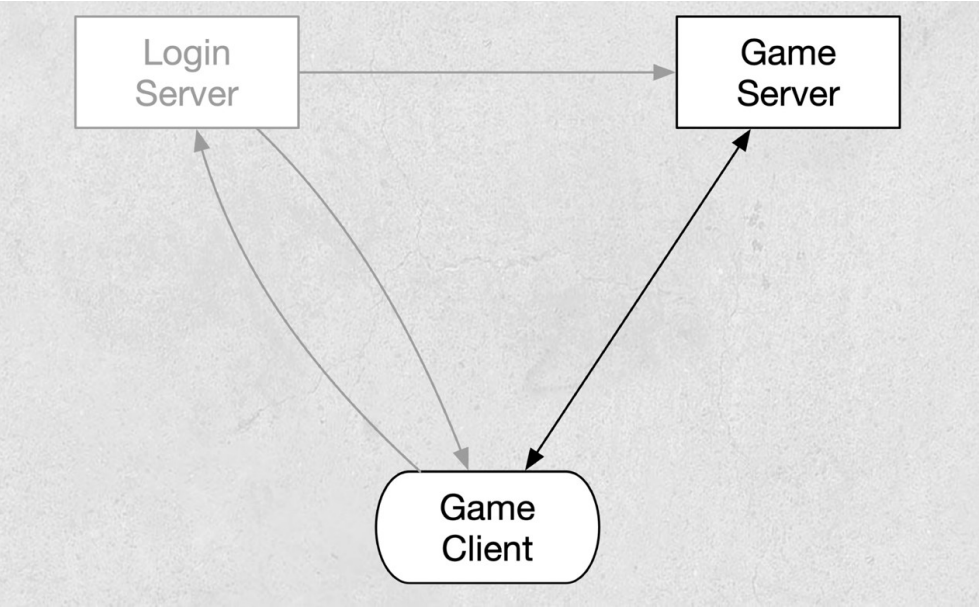
The game client connects to the login server and if the authentication is made a public key of the server



Then the login server sends an encryption key to both game client and game server



And finally the game client connects to the game server, each packet is encrypted with the share key that login server provided



REVERSE ENGINEERING

GAME BINARY ANALYSIS

- Find any encryption routines
- Dissect the protocol handlers
- Reverse-engineer asset loaders
- Defeat anti-debugger protections

Anything that comes from a server you could potentially control is something you should analyze. Each of these inputs could be vulnerable to attack.

SERVER BINARY ANALYSIS

The same steps from client binary analysis apply here. There are some key differences:

- There are typically no anti-debugger mechanisms
- There's less code to inspecting
- There are typically more inputs that go directly from clients to servers than vice versa

PROXIES

Custom proxies make testing easier enabling you to intercept traffic, inject payloads for testing, etc.

These require a high degree of understanding and a large amount of development time, but can lead to great bugs.

POTENTIAL TARGETS

MATCHMAKING

During matchmaking in many games, a server you control is able to provide a name, description, IP + Port, and other details.

This data if often sent directly to clients where it may not be handled properly.

This exact scenario led to an amazing bug and report by *vinnievan* and *0xacb*:

#470520 – RCE on Steam Client via buffer overflow in Server information

$1800 bounty

CHAT

In-game chats usually have many inputs:

- Standard player messages
- Death notifications
- Taunts/actions
- Etc

These may all be handled with one routine, but often they're each handled independently. These each give different potential exploitation paths.

ASSETS

Games have to load a large number of assets – maps, textures, UI layouts, etc – and this code is often built with the assumption that assets are controlled by the studio.

However, many games have facilities for loading assets sent by a server, particularly first-person shooters.

If there's a way you can manipulate assets loaded by the client remotely, this is a huge attack surface that is often only lightly tested.

Report #351016 by *chippy* shows such a bug.

This bug persisted for 20 years and could be exploited remotely.

EMBEDDED BROWSERS

The browsers embedded into many modern games open up new possibilities for exploitation.

XSS in a page embedded in game may allow you to access internal APIs.

These internal APIs can allow everything from loading new assets to taking over accounts.

Once you have control over the browser via standard web bugs, you can explore these internal APIs via enumeration:

```
for(var key in window)
    console.log(key)
```