Contents

Chapter 1

Objective

Traffic analysis classifies network traffic using observable information. Commercial traffic analysis tools are used by governments to block access to websites that counter their current narrative. To prevent blocking, Pluggable Transports (PTs) were developed to obfuscate users' traffic patterns.

PT developers must ensure their tools are able to penetrate nation state firewalls while authoritarian governments must determine the optimal defense [18]. Some popular PTs simply wrap encrypted traffic with a new header to allow TLS traffic to pass through firewalls [20, 5]. At first, this may seem like an elegant solution, but it is simple to add another firewall rule to block this traffic. This is security through obscurity. We choose *Format Transformation Encryption (FTE)* [12] for our PT, which is a form of steganography that translates users' network traffic into a *host protocol* [1] that is not censored and blocked by the network provider of the users. The FTE used in Minecruft-PT is observation-based [22] and is different from the traditional FTE, which is regular expression-based. Accordingly, choosing the proper host protocol becomes critical to the success of FTE-based PTs. The goal of Minecruft-PT is to disguise users' network traffic as innocuous Minecraft game traffic that can be correctly interpreted and accepted by a real Minecraft game server.

[1]The host protocol refers to the protocol being mimicked.

Chapter 2

Introduction of Minecruft-PT

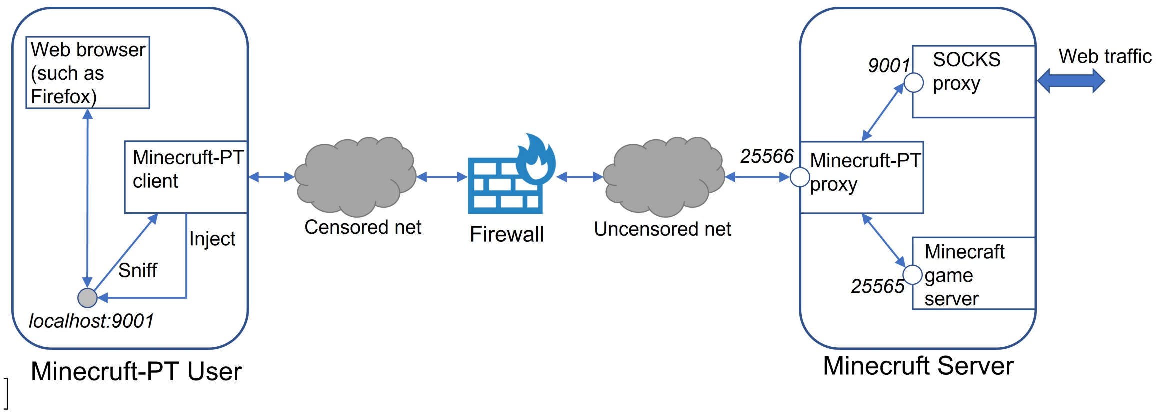
Our Minecruft-Pluggable Transport (Minecruft-PT) uses *Minecraft*[1] – one of the world's most popular video games video – as its host protocol. Minecruft-PT performs protocol mimicry by translating input traffic to Minecraft game traffic. There are several reasons Minecraft is particularly suitable in this context. Although game traffic is generally considered innocuous, many video games are blocked to Iranian players mainly due to the sanctions on Iran. Minecraft is open source under the LGPL (GNU Lesser General Public License). Anyone can set up their own game server and make it open. As a mater of fact, privately-owned Minecraft servers are the main way of the game. There are a number of Minecraft servers within Iran that are open to the public. We provide our users two different Minecraft packages: "Minecruft+SOCKS" and "Minecruft+iPerf" for different purposes.

2.1 Minecruft + SOCKS

Minecruft-PT consists of two components – *Mincruft client* and *Minecruf proxy*. A common application scenario of Minecruft-PT is illustrated in Figure 2.1, where the user uses Minecruft client to access the censored sites through the Minecruft server. The Minecruft server is located on the uncensored network and has three relevant programs running on it: *Minecruft proxy*, *SOCKS proxy* and *Minecraft game server*. Data sent by the user to the censored sites is processed as follows.

- On user side:
  - Minecruft client initiates a game session with the Minecraft game server through the Minecruft proxy.

[1] `https://www.minecraft.net/en-us`



]

Figure 2.1: The application scenario of "SOCKS+Minecruft".

- Minecruft client sniffs the user's web traffic on localhost and translates it to Minecraft game traffic by writing the sniffed data in the payload of various Minecraft packets using FTE-based encoding. These forged Minecraft packets are then inserted into the live game session.

• On Minecruft server side:

  - Minecruft proxy decodes the forged packets to recover the user's web traffic and forwards it to the SOCKS proxy, which then forwards the user's web traffic on to its intended destination.
  - Minecruft proxy forwards the received data directly to the Minecraft game server.

The data transmission of the opposite direction (i.e., from the sites back to the user) is processed similarly, only Minecruft proxy encodes the data while Minecruft client decodes the data. Minecruft-PT users can specify the Minecraft packet types their web traffic is disguised as. The Minecraft Wiki provides the complete list of Minecraft packet types at [17], which are divided into *serverbound* and *clientbound* depending on the data flow.

2.2 Minecraft + iPerf

Note that, not all Minecraft packet types can be used by Minecruft-PT for data encoding. Some of these forged Minecraft packets may be rejected by the Minecraft game server due to a number of reasons, e.g., invalid fields in the packets, incompatibility between forged packets or the order of the packets and current game logic,

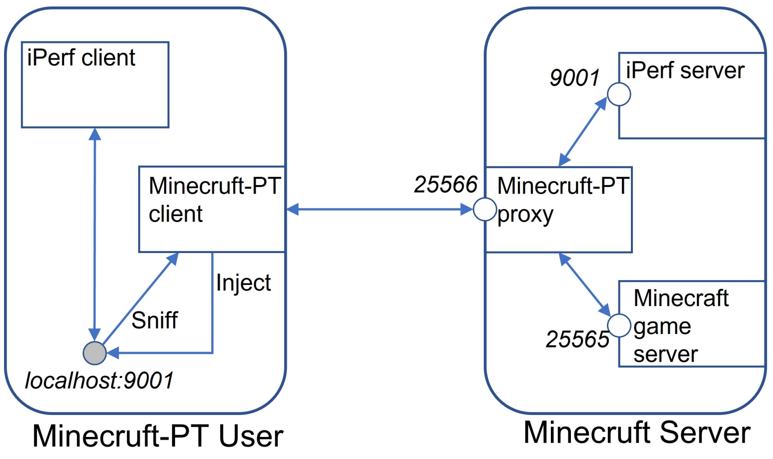


Figure 2.2: Basic configuration of "iPerf+Minecruft".

etc. The test results of currently working packet types are given in Section **??**. More studies and tests are needed to explore how to make optimal use of the available packet types listed at [17].

For testing simplicity, we provide the "Minecruft + iPerf" package, which allows users to test Minecraft without having to dealing with issues caused by any incompatibility between the program (e.g., web browser) using Minecraft for traffic tunneling and Minecraft, e.g., a session timeout caused caused by the delay introduced by Minecraft-PT. The basic configuration of "Minecruft + iPerf" is shown in Figure 2.2. We can see that "Minecruft + iPerf" is very similar to "Minecruft + SOCKS", except that the web browser is replaced by iPerf client and SOCKS proxy is replaced by iPerf server. Using the "Minecruft + iPerf" package, users can easily identify failures caused by the fact that the mixed data is rejected by the Minecraft game server.

Chapter 3

Runtime Environment Setup

Users are strongly recommended to test or modify Minecruft-PT in a safe and sandboxed environment, i.e., inside a virtual machine. The current version has been tested on Linux Mint VMs. The readers are recommended to use VirtualBox to create the Linux Mint VM [1]. The screenshot tutorial for setting up the Linux Mint VM can be found at [10]. Users should allocate at least 4 GB of memory and 20 GB of space to the Linux Mint VM for Minecruft-PT to run smoothly. To enhance the overall interactive performance and usability of the VM, it is recommended to install the *VirtualBox Guest Additions*, the installation guide of which can be found at [21]. Upon the completion of the Linux Mint VM, the following software need to be installed:

- *Minecraft* – Minecruft-PT is built on Minecraft computer edition (1.12.2, protocol 340). Users must install the right Minecraft edition to get Minecruft-PT work properly. The guide for installing Minecraft on Linux (Mint) can be found at [6, 19]. Minecraft requires a license. Users can purchase Minecraft Gift Code through Amazon [2]. Minecraft requires a Microsoft account to play since 2021. Users may go to create a Microsoft account.
- *Docker* – Figure 2.2 and Figure 2.1 show that setting up the runtime environment of Minecruft-PT involves running multiple programs (iPerf client/server, Minecruft client/proxy and Minecraft game server) interacting and exchanging data with each other. Using Docker allows these applications to be isolated into containers with instructions for exactly what they need to run that can

[1]Users are encouraged to test Minecruft-PT on other Linux distros and report issues to the authors.

[2]`https://www.amazon.com/Minecraft-Mac-Online-Game-Code/dp/B010KYDNDG/ref=sr_1_2?crid=4UH8ZNREZNRE&keywords=minecraft`

be easily ported from machine to machine. The guide for installing Docker on Linux Mint can be found at [2, 9].

- *Docker Compose* – Since "Minecruft + iPerf" is a multi-container application, Compose becomes the ideal tool for defining and running it, which uses a YAML file (minecruft_iperf/docker/iperf-test.yml) to configure all the services. The guide for installing Docker on Linux Mint can be found at [1].
- *Python* – Minecruft-PT is coded in Python. Although Minecraft game is written in Java, we selected Python because the implementation Minecruft-PT requires heavy use of *Quarry* – a Python library that implements the Minecraft protocol. It allows you to write special purpose clients, servers and proxies. The guide for installing Python 3.9 on Linux Mint can be found at [7].
- *iperf3* – iperf3 needs to be installed for testing Minecruft-PT.

Upon completion of the software installation, we strongly recommend users to export the Linux Mint VM as an appliance (.ova file) and keep a copy of it before testing Minecruft-PT on the VM. This can allow users to quickly set up the test environment on a different host or in case the current VM crashes during tests.

The dissection of Minecraft computer edition (1.12.2, protocol 340) can be found at [13]. The article covers the different data types of Minecraft traffic, as well as all the packet types.

Chapter 4: Measure Bandwidth using "Minecruft + iPerf"

In this chapter, we provide the step-by-step instruction on how to run "Minecruft + iPerf" on the Linux Mint VM. We also try to cover the necessary background for understanding Minecruft-PT.

The directory tree of the "Minecruft + iPerf" package is given in Appendix A.

The purpose of the "Minecruft + iPerf" package is to test the upload/download bandwidth of Minecruft-PT using different (or different combinations) of Minecraft packet types, without having to worry about issues introduced by the program (e.g., web browser) using Minecruft-PT, e.g., a session timeout caused by the delay introduced by Minecruft-PT. Or to put it differently, almost all the connection issues with "Minecruft + iPerf" are caused by the fact that the forged packets are rejected by the Minecraft game server.

4.1 Running "Minecruft + iPerf"

Running "Minecruft + iPerf" bascially involves running four dockers, one for each of the following applications as shown in Figure 2.2:

- Minecraft game server
- Minecruft-PT proxy.
- Minecruft-PT client.
- iPerf server.

The following box contains the step-by-step instruction of running "Minecruft + iPerf'. And we recommend users to go through the "common problems" listed after the box before running the test to reduce unnecessary trouble.

Open a terminal and navigate to the “Minecruft + iPerf” folder.

- **STEP 1: Firewall Configuration**
  Run the following command to set up the firewall rules needed for running the software. The following command ensures that TCP RST packets much be ignored on your local machine. Otherwise, services like Firefox will see that no service is listening on proxy port 9001 (see Figure 2.2) and will terminate the TCP session. Always run this command after each booting.

```
$ sudo utils/iptables_prestart
```

- **STEP 2: Start Minecraft Gamer Server**
  Run the following command in a separate terminal window/tab to start the Minecraft game server:

```
$ docker-compose -f iperf-test.yml up mserver
```

- **STEP 3: Start Minecruft Proxy and iPerf Server**
  Run the following command in a separate terminal window/tab to start the Minecruft-PT proxy and iPerf server:

```
$ docker-compose -f iperf-test.yml up --build testproxy
  iperfserver
```

- **STEP 4: Start Minecruft-PT Client**
  Run the following command in a separate terminal window/tab to start the Minecruft-PT client:

```
$ docker-compose -f iperf-test.yml up --build testclient
```

- **STEP 5: Start Speed Test**
  Once all the docker containers are up and running, run the following commands to measure the maximum bandwidths of Minecruft. The first command tests the maximum download speed and the second command tests maximum upload speed. The manual of *iperf3* can be found at [3].

```
$ iperf3 -c 127.0.0.1 -p 9001 -R -n 10K
$ iperf3 -c 127.0.0.1 -p 9001 -n 10K
```

- **STEP 5: Stop and remove containers**
  Run the following command in a separate terminal window/tab to stop and removes containers, networks, volumes, and images created by up.

```
$ docker-compose -f iperf-test down
```

∗ The order in which these dockers are started is on based on the functional dependency between these applications.

The following are some common issues users may have during the test. Users are encouraged to report any new issues they encountered to us.

- It may take a few minutes for each docker to start up for the first time because it needs to download all the required software packages.
- Users may need to run all the "docker-compose" commands with "sudo" to avoid running into permission problems.

4.2 Upstream Encoder and Downstream Encoder

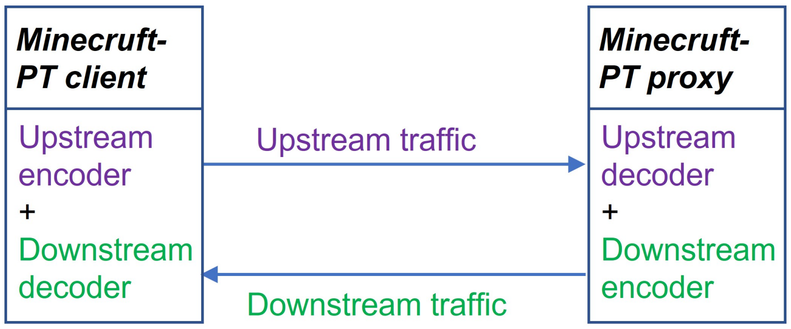
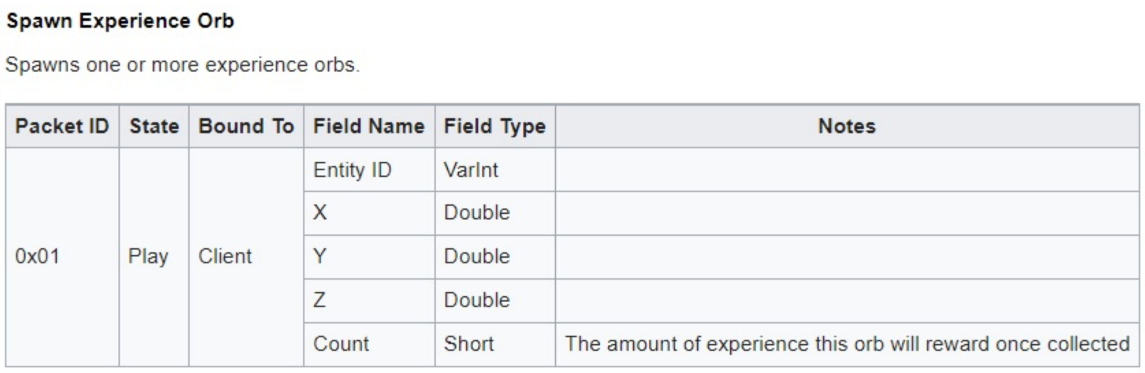

Minecruft-PT client
Upstream encoder + Downstream decoder
Upstream traffic
Downstream traffic
Minecruft-PT proxy
Upstream decoder + Downstream encoder

Figure 4.1: Upstream encoder/decoder + downstream encoder/decoder.

Minecraft uses TCP/25565. The *clientbound packets* [15] delivered from the game player to the Minecraft game server (a.k.a upstream traffic) are different from the *serverbound packets* [16] delivered from the Minecraft game server back to the player (a.k.a downstream traffic). This means different Minecraft packets are used by Minecruft-PT for upstream traffic encoding and the downstream encoding. As illustrated in Figure 4.1, the *upstream encoder* sits on the Minecuft-PT client and uses clientbound packets to encode data; and its corresponding decoder runs on the Minecruft-PT proxy. In contrast, the *downstream encoder* runs on the Minecuft-PT proxy and uses serverbound packets to encode data; and its corresponding decoder runs on the Minecruft-PT client.

4.3 Field-Wise Data Encoding

Encoding data in a Minecraft packet is basically encoding data in the selected fields of the packet. A Minecraft packet is a sequence of bytes divided into multiple *fields*,



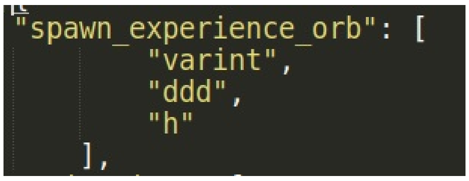
Spawn Experience Orb

Spawns one or more experience orbs.

Packet ID	State	Bound To	Field Name	Field Type	Notes
0x01	Play	Client	Entity ID	VarInt	
			X	Double	
			Y	Double	
			Z	Double	
			Count	Short	The amount of experience this orb will reward once collected

Figure 4.2: Disassembly of clientbound packet "`Spawn Experience Orb`"

each of which is of a particular Minecraft data type. Take the clientbound packet "`Spawn Experience Orb`" illustrated in Figure 4.2 for example. The packet has five fields. The data type of the first field is `VarInt`, followed by three `Double` fields, and the last field is `Short`. All the Minecraft data types are given at [14].



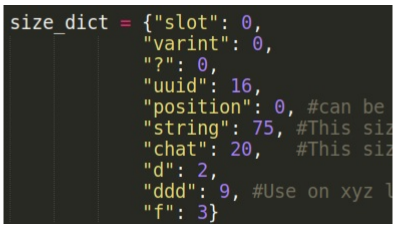
```
"spawn_experience_orb": [
            "varint",
            "ddd",
            "h"
        ],
```

Figure 4.3: Format specifier of clientbound packet "`Spawn Experience Orb`" defined in file "`client_decoder_actions_v12.2.json`"

The two `.json` files (see Section 4.4) contain the format specifiers of the Minecraft packets used for data encoding, one for each direction. The format specifier of a Minecraft packet lists the data types of each field of the packet. The format specifier of packet "`Spawn Experience Orb`" is given in Figure 4.3, which is defined in file "`client_decoder_actions_v12.2.json`". Since Minecruft-PT is witten in Python and uses the `struct` module, these format specifiers also adopt the the format characters used in `struct` module [4], which are inconsistent with the format characters used by Minecraft. So users must ensure the correct format characters are used when editing the `.json` files. Again, take "`Player Look`" for example. As shown in Figure 4.5a, packet "`Player Look`" contains three fields: '`f`', '`f`' and '`?`', which according to [4] are Float, Float and Boolean respectively.

4.3.1 Dynamic data encoding

Currently, both encoders uses the same encoding scheme for fields of the same data type for all Minecraft packets, and is known as *Dynamic Encoding*. The encoding function of different data types, called `encode_action()` is defined in file "`Mincruft_iperf/minecruft/DynamicMethodLoader.py`", where the data types supported by current version of Minecruft-PT is given in Figure 4.4. More encoding schemes using more Minecraft data types will be implemented.



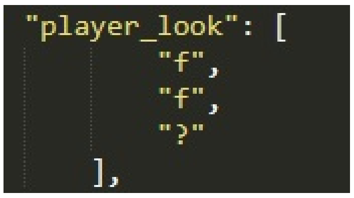
```
size_dict = {"slot": 0,
             "varint": 0,
             "?": 0,
             "uuid": 16,
             "position": 0, #can be
             "string": 75, #This siz
             "chat": 20,   #This siz
             "d": 2,
             "ddd": 9, #Use on xyz l
             "f": 3}
```

Figure 4.4: Data types supported by Minecruft-PT.

Note not all the fields of a selected packet can be used for data encoding. This really depends on if the encoded field value can be accepted by the other end of the communication, or rather if the encoded field value is considered *valid* to the receiver of the packet. There are many factors that jointly determine if the encoded field value is valid or not. For example, a `Float` field may have different ranges in different packets. If an encoded `Float` value is out of the given range of a field, the packet will be rejected by the receiver. Also, a encoded value may depend on or affect the values of other fields, as well as the current game context. For example, current encoders only use the three lowest bits (of the fractional part) of a `Float` field for user data encoding just to limit the range of the encoded float values. One way to improve current dynamic encoding scheme is to customize the encode function for each Minecraft packet type, also known as *Static Encoding*, which is explained in details in Section **??**. In short, all these make the encoding scheme very complex and more work needs to be done to the optimize the encoding of different fields of different packets.

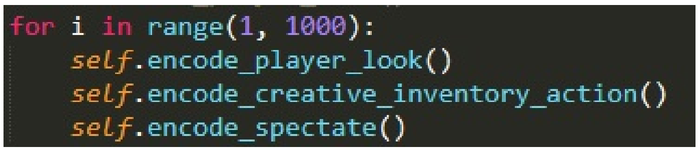
4.4 Add Minecraft Packets to Encoders

Minecruft-PT allows users to specify the Minecraft packets used to encode their network traffic for both directions. Multiple different Minecraft packets can be used in each direction. The default packets used for upstream traffic encoding are "`Player Look`", "`Creative Inventory Action`" and "`Spectate`" [16], all



```
"player_look": [
        "f",
        "f",
        "?"
    ],
```

(a) Snippet of “client_encoder_actions_v12.2.json”



```
for i in range(1, 1000):
    self.encode_player_look()
    self.encode_creative_inventory_action()
    self.encode_spectate()
```

(b) Code snippet of “MinecraftClientEncoder.py”

Figure 4.5: Upstream encoder using packet “Player Look”

of which are serverbound packets. Take “Player Look” for example. To use packet “Player Look” in the upstream encoder, users need to:

- Add the format specifier of packet “Player Look” to file “Minecruft_iperf/configs/client_encoder_actions_v12.2.json” (as shown in Figure 4.5a)
- Add the packet to function “encode()” defined in “Mincruft_iperf/minecruft/MinecraftClientEncoder.py” (as shown in Figure 4.5b)

To remove a packet from the upstream encoder, the user only needs to remove the packet (either delete or comment out the corresponding line) from file “MinecraftClientEncoder py”.

Similarly, the downstream “encode()” function is defined in file “Mincruft_iperf/minecruft/MinecraftProxyEncoder.py”. And the format specifiers of these packets are given in file “Minecruft_iperf/configs/client_decoder_actions_v12.2.json”.

Chapter 5

Developer Notes

The development of Minecruft-PT heavily uses *Quarry* – the Python Minecraft protocol library [11]. Quarry is a Python library that implements the Minecraft protocol. It allows you to write special purpose clients, servers and proxies.

5.1 Quarry's Packet Handlers

Quarry already defines functions for receiving every Minecraft packet type, and registers events for them when a user overrides these functions. For example, look at the at the serverbound "`Use Entity`" packet. If you wanted to print "hello world" every time one of these packets was sent you could overwrite the function `packet_upstream_use_entity` in the `MinecraftProxyEncoder.py` file within the `MinecraftProxyBridge` class. This would be the resulting code:

```
packet_upstream_use_entity(self, buff):
    self.logger.info("hello world")
```

Whenever a packet_EVENTNAME is registered from the minecraft protocol, Quarry will register that function (if EVENTNAME exists in the protocol and quarry's builtin csv listings) and will trigger a callback to the packet function if a minecraft action packet of EVENTNAME is received. This means event handling can be done transparently, and the developer can simply add desired functionality to each handler. To learn more, look at quarry's protocol and dispatcher classes/functions.

If you also wanted to forward the received packet to the minecraft server, you would also add

```
packet_upstream_use_entity(self, buff):
        self.logger.info(``hello world'')
        self.upstream.send_packet(``use_entity'', buff.read())
```

Look into Quarry's documentation about its buffer class to learn more (it is based on structs in python). Also look into Minecraft's data types that are used. For packets that end with minecraft NBT metadata, put the byte value 255 to signal the end of the metadata, otherwise the packet will be malformed and the minecraft server will not accept it.

5.2 Adding Static(Explict) Encoders/Decoders

Quarry's built-in packet handlers are used for defining decoders within Minecruft-PT. There are slight differences between decoders defined in the client ("`MinecraftClientEncoder.py`") and proxy ("`MinecraftProxyBridge.py`"). A static decoder within the client code should look like this:

```
var1 = buff.unpack(fmt)}
var2 = buff.unpack_varint()
......
self.incoming_decode_buffer.append(var1)
self.incoming_decode_buffer.append(var2)
......
update_incoming_buffer(self, self.downstream_factory.
                                    receiving_packet_queue, buff)
```

and a static decoder within the proxy will look something like this (note the additional save/restore functions):

```
buff.save()

var1 = buff.unpack(fmt)
var2 = buff.unpack_varint()
......
self.incoming_decode_buffer.append(var1)
self.incoming_decode_buffer.append(var2)
......
buff.restore()
self.downstream.send_packet(packet_name, buff.read())
update_incoming_buffer(self, self.downstream_factory.
                                    receiving_packet_queue, buff)
```

The encoders for the client and proxy also look slightly different. However, the entry point for handling encoder functions is the `encode()` function in both `MinecraftClientEncoder` and `MinecraftProxyBridge`. This function is registered to execute every game tick ($1/20^{th}$ second) in the `packet_player_position_and_look` (client) and `packet_downstream_player_position_and_look` (proxy) functions. A client encoder will look something like this:

```
# Add more bytes to the buffer if needed
# (typically this is wrapped in an if statement)
check_buff(self.forwarding_packet_queue, self.outgoing_encode_buffer)}

# This used to be get_byte_from_buff(self.outgoing_encode_buffer, num_bytes)
buff = ByteChopper(self.outgoing_encode_buffer)
consumed_bytes = buff.chop(num_bytes)

# Note that multiple self.packs for different data types will be
# included in call to send packet depending on the packet event name.
self.send_packet('packet_event_name', self.pack(fmt,
                 consumed_bytes[:somerange]), more packing calls...)
```

and a proxy encoder will look something like this

```
check_buff(self.forwarding_packet_queue, self.outgoing_encode_buffer)
buff = ByteChopper(self.outgoing_encode_buffer)
consumed_bytes = buff.chop(num_bytes)

# The difference here are the fact that bridges have upstreams
# (real minecraft server facing) and downstreams (client/bot facing)
self.downstream.send_packet('packet_event_name', self.pack(fmt,
                 consumed_bytes[:somerange]), more packing calls...)
```

Note that adding a new encoder/decoder requires looking at the required data types for that packet in the minecraft protocol wiki. Fortunately, by using a json defined version of the protocol (see [Python Minecraft Data Repository] [8]) encoders and decoders can be defined dynamically. However, defining functions dynamically will only allow packing based off fields and the fields types. A rule description can be added to the dynamic functions to enforce action specific rules, but a module that parses and executes these rules has not yet been implemented. If no specific rule is required, then it is simiplier to define dynamic rules.

Chapter 6

Minecruft Modules

6.1 DynamicMethodLoader Module

Bytearray parsing tools and Encoder/Decoder templates This chapter provides functions for dynamically adding encoder and decoder functions to the MinecraftProxyBridge and `MinecraftClientEncoder` (M) classes. It also includes the primary functions for parsing the incoming and outgoing byte buffers.

class `DynamicMethodLoader.`**`ByteChopper`**

Bases: `bytearray`

Simple class for chopping bytearrays

`chop``(`*`end`*`)` Chops from beginning of bytearray up to num_bytes. Stores the remaining bytes in self.

Returns num_bytes from the beginning of the bytes

Return type bytearray

`DynamicMethodLoader.`**`check_buff`**`(`*`fqueue`*`, `*`buff`*`)`

Checks when a buffer or queue is emptied. When a buffer is emptied, that means a full TCP packet has been transmitted to the client. When a full TCP packet is transmitted, it signals the Minecruft client with an encode_ enemy_look packet.

`DynamicMethodLoader.`**`create_decoder_function`**`(`*`cl`*`, `*`prefix`*`, `*`name`*`, `*`fmt`*`, `*`mode`*`)`

`DynamicMethodLoader.`**`create_encoder_function`**`(`*`cl`*`, `*`name`*`, `*`fmt`*`)`

Currently for special minecraft types, we will assume no bytes of data can be packed.

`DynamicMethodLoader.`**`decode_action`**`(`*`unpacker, incoming_decode_buffer, fmt`*`)`

`DynamicMethodLoader.`**`encode_action`**`(`*`self, packer, outgoing_encode_buffer, fmt`*`)`

Self expects the type to be a Minecraft Encoder (either Client or Proxy).

`DynamicMethodLoader.`**`pack_bool`**`(`*`packer`*`)`

A placeholder function so booleans are nott encoded

`DynamicMethodLoader.`**`pack_vint`**`(`*`packer`*`)`

A placeholder function so pack.

`DynamicMethodLoader.`**`safe_unpack`**`(`*`f, unpacker`*`)`

`DynamicMethodLoader.`**`unpack_bool`**`(`*`unpacker`*`)`

`DynamicMethodLoader.`**`unpack_slot_item`**`(`*`unpacker`*`)`

`DynamicMethodLoader.`**`unpack_varint_to_bytes`**`(`*`unpacker`*`)`

`DynamicMethodLoader.`**`update_incoming_buffer`**`(`*`encoder, encoder_queue, buff`*`)`

6.2 MinecraftClientEncoder Module

Python 3 module with core classes for establishing and maintaing Minecraft Client connections, encoding encrypted byte streams, and factory class as an interface to build different types of Minecraft Class Encoders.

`class MinecraftClientEncoder.`**`MinecraftClientEncoder`**`(`*`factory, remote_addr`*`)`

Bases: `quarry.net.client.SpawningClientProtocol`

The MinecraftClient Encoder is a class that represents a Minecraft Client connection and performs the FTE encoding of an encrypted bit stream.

`check_entity``(`*`data`*`)`

Checks an entity if its ID is over 20000. Should be depreciated due to how enitity IDs are assigned.

`encode``()`

Encode packet bytes as minecraft movements. Currently just encodes as creative mode inventory actions, but this can be expanded to other movement types. This is where any new encoder functions should be called.

encode_chat_message()

encode_player_position()
Encode and send original player's position from the start of the game.

encode_player_position_and_look()
Not currently used. Bytes can be injected in to the expected fields for the server such as x, y, z, yaw, and pitch.

packet_player_position_and_look(*buff*)
Receives the initial player position from the server and store the player's information for other turns.

update_player_full()
Sends a player's position to the server every 20 ticks (1 second).

class MinecraftClientEncoder.**MinecraftEncoderFactory**(*encoder_actions=None*, *decoder_actions=None*, *profile=None*, *f_queue=None*, *r_queue=None*)
Bases: `quarry.net.client.ClientFactory`
Factory for building Client Connections. Also serves as the interface for the two packet queues for encoded packets being set between the client and server of the Pluggable Transport.

protocol
alias of `MinecraftClientEncoder`.**MinecraftClientEncoder**

6.3 MinecraftPlayerState Module

class MinecraftPlayerState.**MinecraftPlayer**
Bases: `object`
Simple class for keeping track of a player's state
property position
update_position(*coords: list*)
Update the player's position coordinates for tracking position updates.

6.4 MinecraftProxyEncoder Module

`class MinecraftProxyEncoder.`**`MinecraftProxyBridge`**`(`*`downstream_factory, downstream`*`)`

Bases: `quarry.net.proxy.Bridge`

Class that holds references to both Upstream and Downstream. Waits for both client (Upstream) and server (Downstream) connections to be established before forwarding packets through the proxy.

`downstream_disconnected``(reason=None)`

Called when the connection to the remote client is closed.

`encode``()`

This is the main encode function that encodes packet bytes as minecraft movements. This is where any new encoder functions should be called.

`events_enabled = False`

`gen_rand``(`*`bound`*`)`

Uses os.urandom to generate random integer.

`get_byte_from_buff``(`*`buff`*`)`

Right now returns 256 for no bytes in buff. Later change to None to be more Pythonic.

`get_bytes_from_buff``(`*`buff, num_bytes`*`)`

Read num_bytes from a buffer. Returns a list of bytes.

`packet_downstream_entity_head_look``(`*`buff`*`)`

`packet_downstream_entity_look``(`*`buff`*`)`

`packet_downstream_player_position_and_look``(`*`buff`*`)`

`packet_upstream_creative_inventory_action``(`*`buff`*`)`

`packet_upstream_player_position``(`*`buff`*`)`

`quiet_mode = False`

send_packet(*name*, *buff*)

spawn_mobs(*player_position*)

Spawn mobs in a random position in a sqaure 255x255 blocks centered on the player's starting position. This function should not be used if the Minecraft server is meant to generate mobs.

upstream_factory_class

alias of `MinecraftProxyEncoder.`**`UpstreamEncoderFactory`**

class MinecraftProxyEncoder.**MinecraftProxyFactory**(*downstream_encoder_actions=None*, *upstream_decoder_actions=None*, *c_p_queue=None*, *s_p_queue=None*)

Bases: `quarry.net.proxy.DownstreamFactory`

Factory class for easy instantiation of new Bridge connections between client and server.

bridge_class

alias of `MinecraftProxyEncoder.`**`MinecraftProxyBridge`**

client_bound_buff = bytearray(b'')

connectionMade()

forwarding_packet_queue = None

motd = 'Proxy Server'

num_client_encoders = 0

num_waiting_encoders = 0

sync_buff(*oldbuff*)

Each Downstream must send the same data for enemy movement packets from the global buffer in order for the game sessions to be authentic.

class MinecraftProxyEncoder.**UpstreamEncoder**(factory, remote_addr)

Bases: `quarry.net.proxy.Upstream`

Inherits from Upstream Class. Represents the Client Connection to the Minecraft Server. Contains a buffer ¡Proxy_bound_buff¿ for holding packets destined for the Minecruft Client.

`assigned_enemy = 0`

`assigned_id = 0`

`class MinecraftProxyEncoder.`**`UpstreamEncoderFactory`**`(`*`profile=None`*`)`

Bases: `quarry.net.proxy.UpstreamFactory`

Factory class for Upstream encoder for easy instantiation of multipleUpstreams for multiple Minecraft PT clients.

`protocol`

alias of `MinecraftProxyEncoder.`**`UpstreamEncoder`**.

6.5 Minecruft_Main Module

Usage:

`Minecruft_Main.py` `client <aes_keyfile> <dest_ip> <dest_port> <fwd_ports> <encoder_actions_file> <decoder_actions_file> [options]`

`Minecruft_Main.py` `proxy <aes_keyfile> <dest_ip> <dest_port> <fwd_ports> <encoder_actions_file> <decoder_actions_file> [options]`

Options: -h –help Show help message -proxy_port ⟨proxy_port⟩ Proxy Listen Address [default: 25566] -t –test Send test message through Minecruft instead of TCP packets

`Minecruft_Main.`**`client_forward_packet`**`(`*`encrypt_tcp_q, decrypt_tcp_q, encoder_actions, decoder_actions, forward_addr, proxy_port`*`)`

Starts twisted's event listener for sending and recieving minecraft packets, and sets up the client bot to connect to the proper host machine using the runargs function. Also passes the *`encrypt_tcp_q`* (for forwarding encrypted packets) and the *`decrypt_tcp_q`* (for receiving encrypted packets) to the minecraft client encoder object.

`Minecruft_Main.`**`decrypt_enc_data`**`(`*`decrypt_tcp_q, response_q, key`*`)`

This is a function for receiving packets on the host.

Removes packets from *decrypt_tcp_q*, decrypts them with AES_ECB, and then stores the encrypted packets into the *reponse_q*.

`Minecruft_Main.`**`decrypt_load`**`(`*`cipher_str`*`, `*`key`*`)`

Encrypts with AES_ECB using a default password. This is not for security it is for creating an even chance of sending binary ones and zeros.

`Minecruft_Main.`**`encrypt_load`**`(`*`message`*`, `*`key`*`)`

Encrypts with AES_ECB using a default password. This is not for security it is for creating an even chance of sending binary ones and zeros.

`Minecruft_Main.`**`encrypt_tcp_data`**`(`*`incoming_tcp_q`*`, `*`encrypt_tcp_q`*`, `*`direction`*`, `*`key`*`)`

This is a function for sending packets from the host.

Removes packets from *incoming_tcp_q*, encrypts them with AES_ECB, and then stores the encrypted packets into the *encrypt_tcp_q*.

`Minecruft_Main.`**`filter_packets`**`(`*`incoming_tcp_q`*`, `*`duplicate_packets`*`)`

Factory Function for generating filters

`Minecruft_Main.`**`inject_tcp_packets`**`(`*`response_q`*`)`

Takes the received unencrypted *response_q* TCP packets and injects them onto the localhost's machine.

`Minecruft_Main.`**`proxy_forward_packet`**`(`*`encrypt_tcp_q`*`, `*`decrypt_tcp_q`*`, `*`downstream_encoder_actions`*`, `*`upstream_decoder_actions`*`, `*`minecraft_server_add`* *`minecraft_server_port`*`, `*`listen_addr`*`, `*`listen_port`*`)`

Starts twisted's event listener for sending and recieving minecraft packets, and sets up the proxy server which first connects to a Minecraft server set in offline mode and then waits for incoming Minecraft Client connections. Also passes the *encrypt_tcp_q* (for forwarding encrypted packets) and the *decrypt_tcp_q* (for receiving encrypted packets) to the minecraftProxyEncoder object.

`Minecruft_Main.`**`receive_tcp_data`**`(`*`tcp_port`*`, `*`direction`*`, `*`incoming_tcp_q`*`)`

This is a function for sending packets from the host.

Sniffs all packets of localhost that have ports with the value *tcp_port* traveling either from src or dst using the direction variable.

`direction`: src or dst *tcp_port*: String resembling TCP ports such as 9000, 20, 80 *incoming_tcp_q*: Queue for saving incoming packets

`Minecruft_Main.`**`runargs`**`(`*`encoder_actions`*`, `*`decoder_actions`*`, `*`args`*`, `*`encrypt_tcp_q`*`, `*`decrypt_tcp_q`*`)`

Creates a Minecraft Client Bot, and connects to the Minecraft proxy. This assumes that the Minecraft Proxy is already running on another machine.

Appendices

Chapter A

Appendix A: Figures

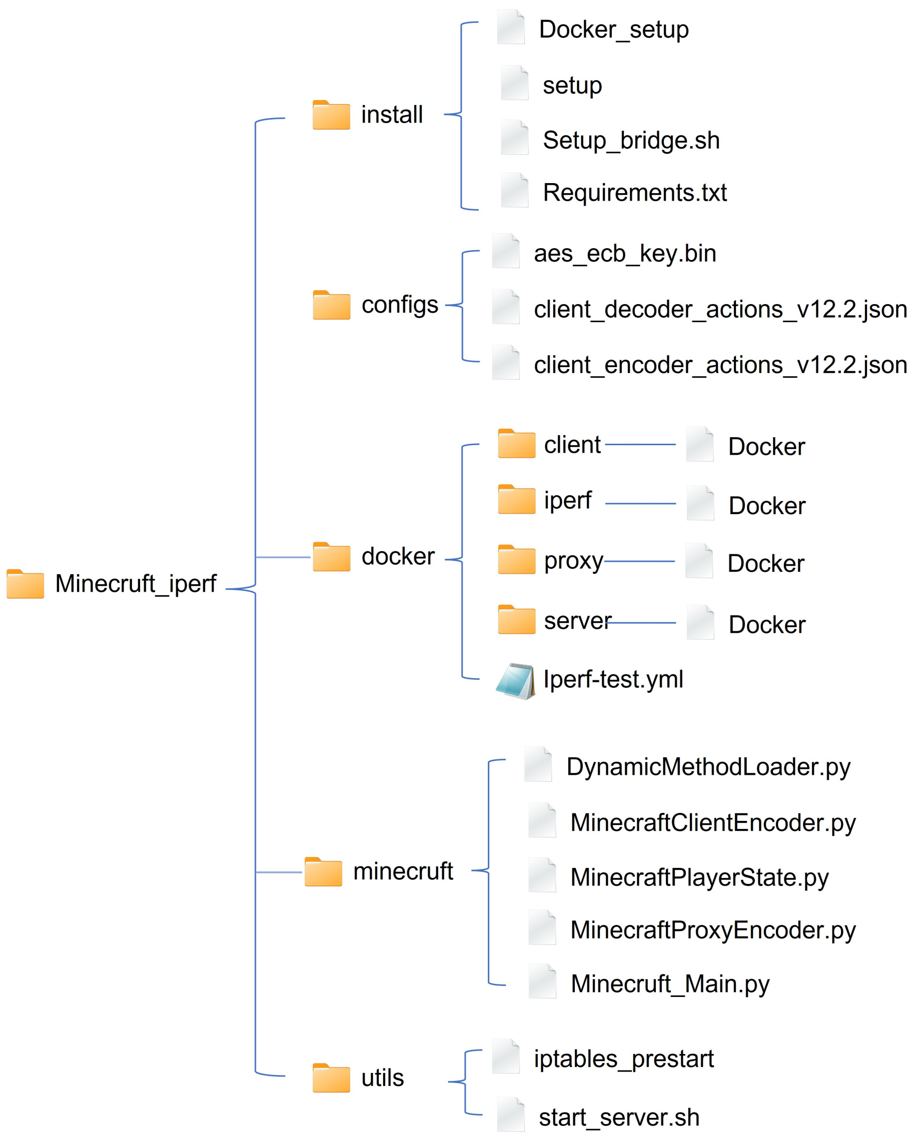


Figure A.1: Directory tree of “iPerf+Minecruft” package.

Chapter

Bibliography

[1] Install Docker Compose. https://docs.docker.com/compose/install/, 2021. [Online; accessed 22-December-2021].

[2] Install Docker Engine on Ubuntu. https://docs.docker.com/engine/install/ubuntu/, 2021. [Online; accessed 22-December-2021].

[3] iperf3 - Man Page. https://www.mankier.com/1/iperf3, 2021. [Online; accessed 22-December-2021].

[4] struct — Interpret bytes as packed binary data: Format Characters. https://docs.python.org/3/library/struct.html#format-characters, 2021. [Online; accessed 22-December-2021].

[5] Y. Angel. obfs4. https://github.com/Yawning/obfs4, 2020. [Online; accessed 22-December-2021].

[6] anilabhadatta. How to Install Minecraft on Linux? https://www.geeksforgeeks.org/how-to-install-minecraft-on-linux/, 2021. [Online; accessed 22-December-2021].

[7] K. S. Awaisi. How to Install Python 3.9 on Linux Mint 20? https://linuxhint.com/install-python-3-9-linux-mint/, 2021. [Online; accessed 22-December-2021].

[8] R. Beaumont. PrismarineJS/minecraft-data. https://github.com/PrismarineJS/minecraft-data, 2020. [Online; accessed 22-December-2021].

[9] L. Chepkoech. How To Install and Use Docker CE in Linux Mint 20. https://techviewleo.com/how-to-install-and-use-docker-in-linux-mint/, 2021. [Online; accessed 22-December-2021].

[10] Dimitrios. How to Install Linux Mint in VirtualBox [Screenshot Tutorial]. `https://itsfoss.com/install-linux-mint-in-virtualbox/`, 2020. [Online; accessed 22-December-2021].

[11] Dimitrios. Quarry: a Minecraft protocol librar. `https://quarry.readthedocs.io/en/latest/`, 2020. [Online; accessed 22-December-2021].

[12] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 61–72, 2013.

[13] Fayaru. Protocol. `https://wiki.vg/index.php?title=Protocol&oldid=14204`, 2018. [Online; accessed 22-December-2021].

[14] Fayaru. Protocol: Definitions: Data types. `https://wiki.vg/index.php?title=Protocol&oldid=14204#Data_types`, 2018. [Online; accessed 22-December-2021].

[15] Fayaru. Protocol: Handshaking: clientbound. `https://wiki.vg/index.php?title=Protocol&oldid=14204#Clientbound`, 2018. [Online; accessed 22-December-2021].

[16] Fayaru. Protocol: Handshaking: serverbound. `https://wiki.vg/index.php?title=Protocol&oldid=14204#Serverbound`, 2018. [Online; accessed 22-December-2021].

[17] Fayaru. Protocol:Play. `https://wiki.vg/index.php?title=Protocol&oldid=14204#Play`, 2018. [Online; accessed 22-December-2021].

[18] A. Garnaev, M. Baykal-Gursoy, and H. V. Poor. Security games with unknown adversarial strategies. *IEEE transactions on cybernetics*, 46(10):2291–2299, 2015.

[19] S. Muntaha. Play Minecraft with Linux Mint. `https://linuxhint.com/minecraft_linux_mint/`, 2019. [Online; accessed 22-December-2021].

[20] B. Wiley. Dust: A blocking-resistant internet transport protocol. *Technical rep ort. http://blanu. net/Dust. pdf*, 2011.

[21] G. Xiao. Install VirtualBox Guest Additions in Linux Mint Step by Step. `https://www.linuxbabe.com/linux-mint/install-virtualbox-guest-additions-in-linux-mint`, 2020. [Online; accessed 22-December-2021].

[22] X. Zhong, Y. Fu, L. Yu, R. Brooks, and G. K. Venayagamoorthy. Stealthy malware traffic-not as innocent as it looks. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 110–116. IEEE, 2015.