Pangya's Preservation

Preserving the dying videogame Pangya

Adán López Gutiérrez
Directed by Jordi Duch Gavaldà

A thesis presented for the bachelor of
Science in Computer Engineering

Spain, Tarragona
June 2024

Pangya's Preservation

Adán López Gutiérrez

June 2024

Abstract

Pangya, the beloved online multiplayer golf game developed by `Ntreev Soft`, captured the hearts of gamers worldwide with its unique blend of casual sports and anime aesthetics. Against its niche community, `Ntreev Soft`'s closure in 2024 threatens the game's future accessibility. This thesis proposes a multifaceted approach to preserving Pangya.

We explore the origins of Pangya, examining its evolution through the years. Technical challenges in preservation will be addressed, including emulation of the game server and tinkering of client software. Additionally, the thesis will discuss community-driven efforts such as digital archiving and reverse engineering to ensure Pangya's long-term survival.

By advocating for the preservation of Pangya, the aim is not to only safeguard a piece of gaming history but also celebrate the creativity and joy it brought to countless players.

Resumen

Pangya, el querido juego de golf multijugador en línea desarrollado por `Ntreev Soft`, capturó los corazones de los jugadores de todo el mundo con su combinación única de deportes casuales y estética de anime. Frente a su comunidad de nicho, el cierre de `Ntreev Soft` en 2024 amenaza la accesibilidad futura del juego. Esta tesis propone un enfoque multifacético para preservar Pangya.

Exploramos los orígenes de Pangya, examinando su evolución a lo largo de los años. Se abordarán los desafíos técnicos en la preservación, incluyendo la emulación del servidor del juego y la modificación del software del cliente. Además, la tesis discutirá los esfuerzos impulsados por la comunidad, tales como la archivación digital y la ingeniería inversa, para asegurar la supervivencia a largo plazo de Pangya.

Abogando por la preservación de Pangya, el objetivo no es solo proteger una pieza de la historia de los videojuegos, sino también celebrar la creatividad y la alegría que trajo a innumerables jugadores.

Resum

Pangya, el cèlebre joc de golf multijugador en línia desenvolupat per `Ntreev Soft`, va captivar els cors dels jugadors de tot el món amb la seva combinació única d'esports casuals i estètica d'anime. Davant de la seva comunitat nínxol, el tancament de `Ntreev Soft` el 2024 amenaça l'accessibilitat futura del joc. Aquesta tesi proposa un enfocament multifacètic per preservar Pangya.

Explorem els orígens de Pangya, examinant la seva evolució al llarg dels anys. Es tractaran els desafiaments tècnics en la preservació, incloent-hi l'emulació del servidor del joc i la manipulació del programari del client. A més, la tesi discutirà els esforços impulsats per la comunitat, com ara l'arxiu digital i l'enginyeria inversa, per assegurar la supervivència a llarg termini de Pangya.

En defensar la preservació de Pangya, l'objectiu no és només salvaguardar una part de la història dels videojocs, sinó també celebrar la creativitat i l'alegria que va portar a innumerables jugadors.

Dedication

This thesis has been in progress for almost as long as my career as a Computer Engineer started, three years.

There have been many times when I thought I would never achieve to get a functional emulator for the Game Server, multiple months stuck at problems that seemed unfathomable.

I thank wholeheartedly, and dedicate this thesis, to all those friends, family and teachers who supported me through the hard moments I had in this journey.

Specifically (but not exclusively):

- My father, mother and girlfriend, for helping me in those moments when I felt impotent to a problem I could not solve; and for celebrating each time I achieved to solve them.
- My friends, for encouraging me and being the first ones in line to try out each advance I made.
- The `retreev` community, formerly `Cadie's Cauldron`, for being such a great help and source of wisdom on all Pangya things related. Without them this would have been lots of times harder.
- To PandoraCloudGames, for teaching me how to code proper Game Server emulators.
- To the teachers to whom I showed the thesis and were amazed by it.

And also to the players of Pangya, lets continue what `Ntreev Soft` could not!

Contents

List of Figures

List of Listings

1 Introduction

Disclaimer: *most of the information about how everything came to be has been lost to time, and can only be recovered through The Wayback Machine, old fans' blogs or people that has the knowledge because they already compiled it to some extent.*

1.1 Birth of the game

1.1.1 The developer

`Ntreev Soft`'s roots trace back to the South Korean developer `Sonnori`, which underwent a split in late 2003, dividing the original company into two distinct entities.

`Ntreev Soft` emerged from this split with a focus on Massively Multiplayer Online (MMO) games, while `Sonnori` continued development on single-player titles. [1]

This separation marked the official beginning of `Ntreev Soft` as we know it.

Interestingly, the company's roots go even deeper. In the early 2000s, `Sonnori` employee Kwanhee Seo, developed the `WangReal Engine` used for the single-player horror game `White Day: A Labyrinth Named School`. [2]

This background sheds light on two key points:

1. The short timeframe between the split and the launch of Pangya's beta in `February 2004` becomes more understandable, suggesting that development likely began before the official separation. [3]

2. It explains why Pangya, developed by the newly formed `Ntreev Soft`, continued to utilize the `WangReal Engine`.

1.1.2 Game genesis

`Pangya` began its journey with a beta phase roll-out in 2004, assisted by the videogame publisher and developer `Hanbit Soft`. Beta testing launched first in South Korea, the developer's home country, on `February 25, 2004`, followed by Japan on `June 18, 2004`. [3, 4]

Shortly thereafter, both regions received the official game launch (`Season 1`) in `June 2004` and `November 11, 2004`, respectively. [3, 5]

Subsequently, it expanded into numerous other regions, spawning multiple spin-off games in the process. [6, 7]

Figure 1: An image showcasing Pangya

1.2 Evolution and Transformation

1.2.1 Development Over Time

Throughout its lifespan, `Pangya` has been managed and published by a variety of regional entities, each tasked with infrastructure management and payment integration. The degree of freedom and responsibilities each region had was different depending in their resources and agreements with `Ntreev Soft`. [8]

However, a pattern emerges in which many of these publishers only maintained their operations for a relatively short period, typically lasting between 2 to 3 years. [6]

The exact reasons for this frequent turnover remain unclear, but it may be attributed in part to the licensing arrangements done by `Ntreev Soft`.

The regions where `Pangya` found more stable footing were: its birth place Korea (initially overseen by `HanbitSoft` and later by `Ntreev Soft`), as well as Japan, Thailand, and the United States (albeit with numerous changes in hands over time). [6]

These regions experienced longer periods of consistent management and operation, suggesting a more sustainable approach to licensing and publishing within those markets, or perhaps yielding superior returns for the licensor.

Later on after multiple updates throughout the years, the stagnation of Pangya's development became evident on early 2016, when `Ntreev Soft` stopped the development of new features altogether.

This event marked a significant downturn in the trajectory of Pangya as the

cessation of updates typically signals a halt in the introduction of new content, features, and improvements crucial for player engagement and retention.

Soon after, the closure of the Korean server occurred on August 29, 2016, followed by the closure of the United States server on December 12, 2016. [6]

However, there were two notable exceptions. The Japanese version, reportedly retaining the ability to generate updates independently, purportedly due to access to the game's source code, persisted. Nevertheless, perhaps due to licensing issues with `Ntreev Soft`, they ultimately closed down the server on November 10, 2017. [9]

The other exception was the Thai version, which endured significantly longer than its counterparts without updates, differing by a span of seven years. Further discussion regarding its demise will be provided in the subsequent section, as its downfall occurred relatively recently.

1.2.2 Present-Day Landscape

Thailand's publisher `Ini3 Digital` was the one to maintain its operations for the longest time with the game, almost 20 years without any major hitches. But recently `Ntreev Soft` entered bankruptcy on February 15, 2024. [10]

This fact impacted directly `Ini3 Digital`, the long-lasting contract they had with `Ntreev Soft` could no longer be renewed, which is the reason why they announced on March 29, 2024 that they would cease operations on April 30, 2024. [11]

To show how many players were affected, I decided to craft a bot that automatically connected to Thailand's Login Server to gather the amount of players logged in on intervals of 10 minutes. A representation of the data gathered can be seen in Figure 2.

In scrutinizing the graphical representation, it becomes apparent that the player population tends to stabilize around the 300 mark during peak activity periods.

Additionally, discernible fluctuations coincide with the anticipated windows of leisure for Thai players, occurring between 7:00 PM and 1:00 AM GMT+7, presumably when they are less encumbered by obligations such as work or sleep.

Noteworthy is the observable surge in player engagement during the final two days leading up to the game's cessation. During this period, a pronounced escalation in concurrent players is evident, culminating in a peak of 700 individuals engaged in gameplay simultaneously.

Such an uptick likely signifies heightened interest or a sense of urgency prompted by the imminent closure of the game.

1.3 Preservation Efforts

Originally known as "Cadie's Cauldron" and later rebranded as `retreev` on November 24, 2021, this collective, founded by Andreas Nedbal (also known

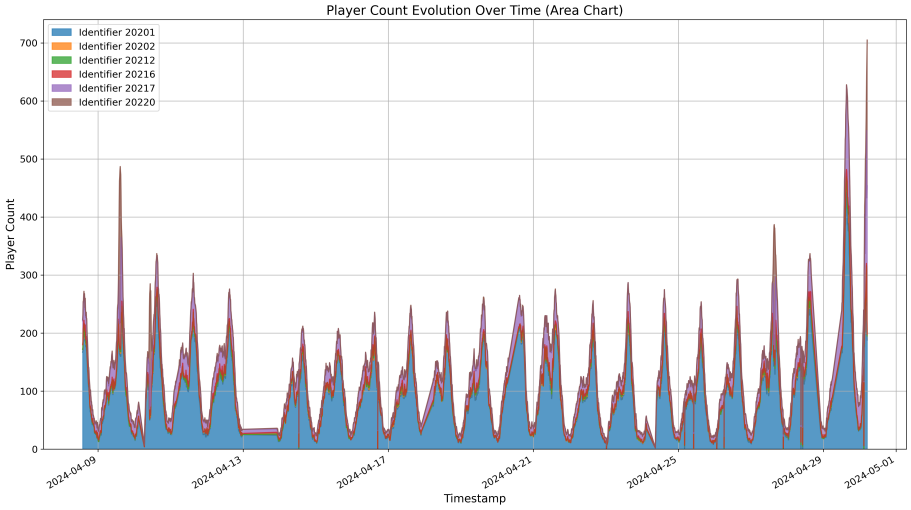

Figure 2: Player Count Evolution Over Time (Area Chart)

as "pixeldesu") on December 9, 2018, stands as a cornerstone in the realm of Pangya research, preservation, and tool development.

It serves as a vibrant community hub for developers keen on unraveling the intricacies of Pangya, crafting innovative tools, and documenting their discoveries.

Central to this collective is the `retreev` GitHub organization, a collaborative space where contributors can engage in joint projects that revolve around Pangya. [12]

Currently it has extensive documentation and tools pertaining to client files and their inherent structures. [13, 14, 15]

Additionally, it houses a comprehensive library that incorporates the obfuscation algorithm utilized by both the client and server for communication. [16]

Among the members, Andreas Nedbal and John Chadwick stand out as longstanding pillars and main contributors to the `retreev` organization.

Additionally, the contributions of individuals like Luiz Lopez, Hexagon (hexagon on Github) and Acrisio Filho have been instrumental in advancing the collective's mission. [17, 18]

Furthermore, lots of independent developers have played a pivotal role in laying the groundwork for various tools and server emulators. Here I leave some noteworthy contributors:

- HSReina: A pioneering figure from the early days, her adept reverse engineering skills have unraveled significant portions of the official server's

functionalities. She later open-sourced her Pangya Server Emulator and provided tools for accessing and modifying client files, including 3D models and animations.[19]

- KuramaxD: Known for open-sourcing his emulator, has contributed to the community's pool of resources. [20]
- Unogames: This group has generously shared multiple emulators tailored for different regions, fostering a collaborative spirit within the community. [21]
- Acrisio Filho: In recent years, he has been a beacon of knowledge, sharing remarkable tools and insights with the community. His contributions include what is arguably the most comprehensive Pangya Server Emulator to date. [22]

In summary, the collective's support and the independent contributions have significantly helped me in this research, providing detailed insights into the inner workings of Pangya whenever I lost track.

2 Objectives

The objectives of this thesis revolve around the preservation and continuation of Pangya, the online multiplayer golf game developed by Ntreev Soft. The primary goals are as follows:

1. Preserve the game as best as possible: Ensure that Pangya is maintained in its original form to the best extent possible.
2. Gather knowledge:
   (a) Collect comprehensive knowledge related to the internal workings of the game, particularly focusing on emulating the behavior of an official server.
   (b) Expand the existing knowledge base of the preservation community through original research.
   (c) Centralize the dispersed knowledge within the preservation community.
3. Design and develop an emulator:
   (a) Utilize the gathered knowledge to create a server that accurately replicates the original game behavior.
   (b) Develop a scalable server infrastructure capable of supporting an appropriate number of players.
4. Enhance security:
   (a) Conduct a thorough analysis of security vulnerabilities in both the official server and client.
   (b) Propose theoretical solutions and, if feasible, implement them to mitigate security risks in the emulator implementation.

By pursuing these objectives, this thesis aims to address the technical challenges in preserving Pangya, ensuring its accessibility for future generations.

Additionally, it seeks to foster community-driven efforts in digital archiving and reverse engineering to secure the long-term survival of Pangya.

Ultimately, the goal is not only to safeguard a piece of gaming history but also to honor the creativity and joy that Pangya has brought to its players worldwide.

3 Study and Analysis

3.1 Introduction

This section delves into the essential aspects that require study and analysis to ensure the successful operation of the client in the absence of the original infrastructure. This involves emulating or bypassing the necessary infrastructure behaviors.

Furthermore, it encompasses a fundamental analysis of the communication protocol that the client upholds with the various servers it interacts with to establish its identity and synchronize its state with other players.

All the analysis done is specific to the version `GB.R7.851.00`, from the US region.

3.2 Launcher Coordination

In order to ensure the successful operation of the client in the absence of the original infrastructure, it is crucial to address the coordination between the Launcher and the Client.

This coordination mechanism plays a vital role in ensuring that the client cannot operate independently from the launcher, likely implemented to prevent users from running outdated versions of the software.

As we delve deeper into the subsequent subsections, it becomes evident that this coordination is essential for maintaining the integrity of the system. With the original Content Delivery Network (CDN) no longer functional, the launcher is unable to perform its primary function of updating the game.

Additionally, it restricts the launch of the game if it cannot carry out its necessary routines against the CDN. This situation prompts us to search for ways to circumvent these limitations.

3.2.1 Environment Variable

The coordination between the Launcher and the Client is initiated by the client checking the environment variable `PANGYA_ARG`. If this variable is not set to the specific hard-coded GUID, the client will halt its execution and display a message before terminating (the message can be seen in Figure 3).

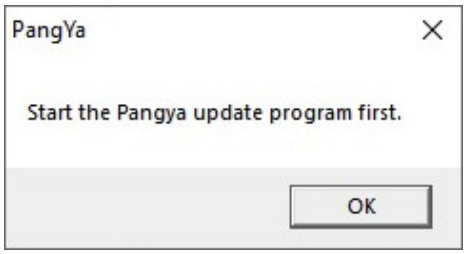

Figure 3: The client tells us to use the launcher first.

The GUID appears to be randomly generated, with each region apparently having its distinct GUID hard-coded within the code. This necessitates an external entity to configure this variable to the correct value prior to the execution of this check.

The one used by the US version is the following: {`2D0FA24B-336B-4e99-9D30-3116331EFDA0`}

3.2.2 Registry Keys

Registry values serve a dual purpose for the client, acting as repositories for user-configured client options and as a means for the launcher to communicate the base package to the client.

The significance of packages, which will be elaborated on in the subsequent File Integrity subsection, lies in their role as a virtual file system utilized by the client to access game data such as textures, models, sounds, and other essential components. It is worth noting that this virtual file system can be overlaid with incremental updates by adding new packages.

The base registry key used to save other registry keys and values varies depending on the region of the published client. In the US client we are analysing, it specifically uses `"HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Ntreev USA \Pangya"`.

The inclusion of `WOW6432Node` in the registry path is a result of the client being originally designed for Windows XP systems, which, at the time, predominantly operated on a 32-bit architecture rather than the 64-bit architecture commonly used in modern systems.

In this path we find the aforementioned value, which can be seen in Figure 4.

`IntegratedPak` defines the first package that should be loaded, making it the basis for the incremental updates. Manipulating its value, such as defining a higher package like `projectg800gb+.pak` that is not the "base package," can lead to the client closing before displaying the login screen.

The reason for this behavior is likely due to the client not loading essential files required for the login screen, resulting in an unexpected closure without

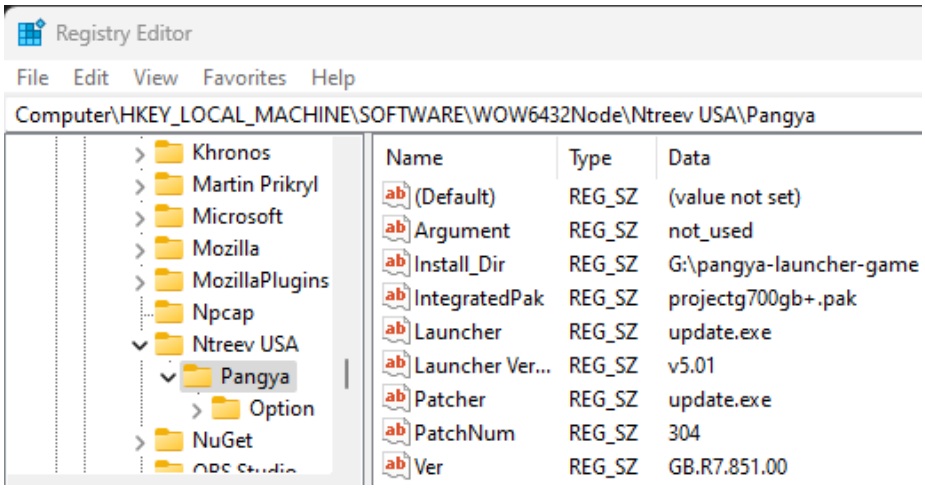

Figure 4: Windows 11's Registry Editor showing the previously mentioned path.

warning.

Interestingly, defining lower values than the existing packages, like `projectg600gb+.pak`, it indicates that `projectg632gb.pak` is corrupted; can be seen in Figure 5.

Although the exact cause remains unknown as further reverse engineering was not conducted, it is suspected that starting from package "600", the client attempts to load all other packages that are higher than "600" defined in the "updatelist", a topic that will be explored in detail in subsequent sections.

A preliminary examination of the decrypted version of the "updatelist" reveals a reference to `projectg632gb.pak`, even if the file is not present in the client's files (as it is from an older version of the game), which makes this hypothesis plausible. A fragment of the "updatelist" can be seen in Listing 1.

```
<?xml version="1.0" encoding="euc-kr" standalone="yes" ?>
<!-- ... -->
<updatefiles count="258">
    <!-- ... -->
    <fileinfo fname="projectg632gb.pak" fdir="" fsize="82327687"
    ↪  fcrc="-85837936" fdate="2012-06-07" ftime="07:21:51"
    ↪  pname="projectg632gb.pak.zip" psize="75535282" />
    <!-- ... -->
</updatefiles>
```

Listing 1: Extract from the original US' CDN decrypted "updatelist".

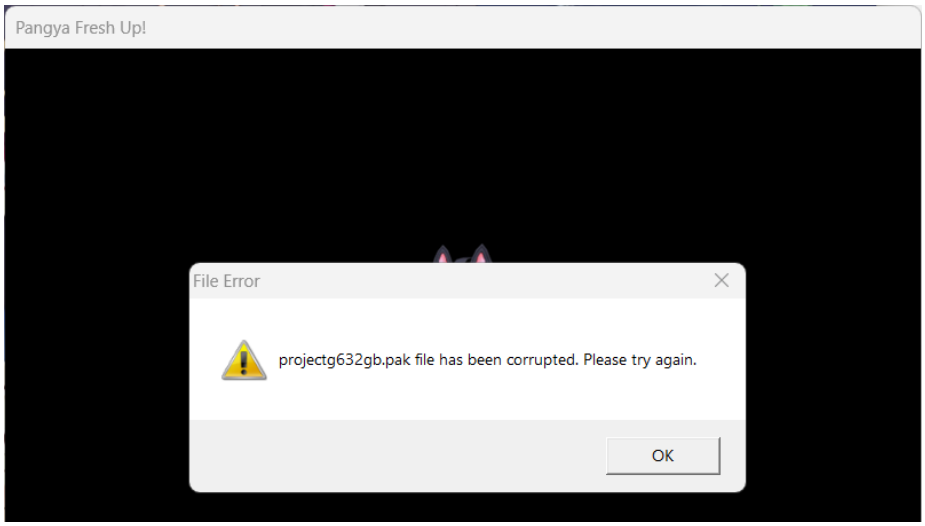

Figure 5: The client warning us about `projectg632gb.pak` being corrupted.

Removing `IntegratedPak` as a value makes the client warn about re-installing the game or using the launcher first (can be seen in Figure 6).

So, in summary, not having the `IntegratedPak` key value set in the correct place or with the appropriate value can render the client unable to launch successfully. This key serves as a critical reference point for the client to initiate the loading process of essential game data packages.

Failure to configure it correctly can disrupt the client's ability to access the necessary files, leading to errors during the initialization phase and ultimately preventing the client from launching as intended.

Apart from `IntegratedPak`, the other values that can be seen in Figure 4 are mainly for the launcher itself, a component of Pangya that was not extensively researched in this study to prioritize the investigation of other functionalities.

3.3 There Can Be Only One

The developers of Pangya implemented a restriction to allow players to run only one instance of the game simultaneously.

This limitation is enforced to prevent users from running multiple instances concurrently. Any attempt to launch a second instance of the game will result in the focus returning to the first instance, terminating the new instance immediately.

The rationale behind this restriction likely stems from the potential negative impact of players running multiple instances, a practice known as multi-boxing. Multi-boxing can be exploited by players for unfair advantages, such as gaining

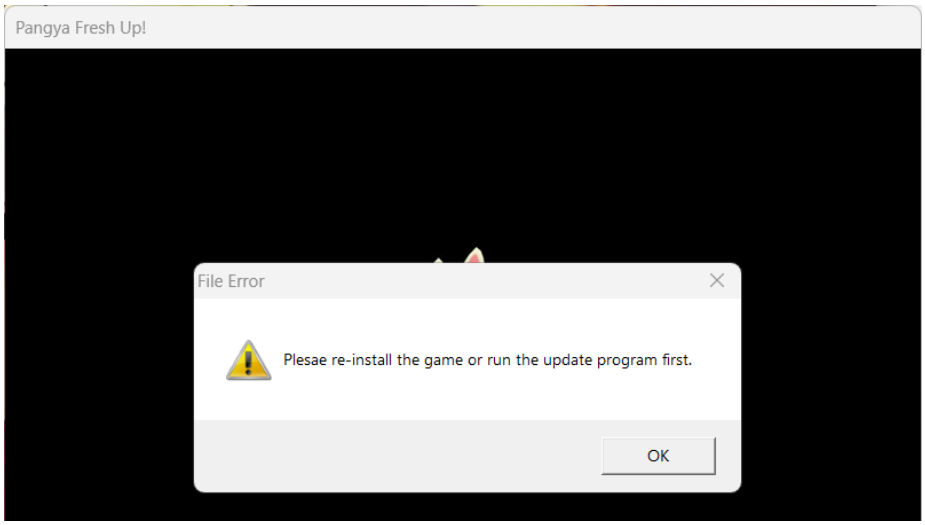

Figure 6: The client warning us that we need to re-install the game or use the launcher first.

more power, resources, or control over the game environment compared to other players.

Upon closer examination of the client's code using reverse engineering tools like Ghidra, it reveals a specific function designed to detect other instances of the game through a named mutex lock associated with a unique hard-coded GUID. Snippet of the reverse engineered code can be seen in Listing 2.

This approach of utilizing a hard-coded GUID to monitor the proper usage of the client mirrors the methodology observed with the `PANGYA_ARG` environment variable, which also relied on a GUID for validation.

As a person who has been developing a server emulator for the game, it is worth noting that the restriction on running multiple instances can pose challenges.

Testing synchronization in multiplayer golf games becomes cumbersome without the ability to run multiple instances simultaneously on a single machine. This limitation necessitates the use of multiple computers for comprehensive testing, adding complexity to the development process.

Sharing this analysis can be beneficial for others facing similar constraints in their development or testing environments, enabling them to leverage this knowledge to devise appropriate solutions.

```
HANDLE check_for_other_instances(void)
{
  HANDLE mutex_h;
  DWORD error_details;
  HWND hwnd_other_pangya;

  mutex_h =
  ↪  CreateMutexA(0x0,1,"{071784A2-EE35-4e6a-92D0-6E7A4B985171}");
  if (mutex_h != (HANDLE)0x0) {
    error_details = GetLastError();
    if ((error_details != ERROR_ALREADY_EXISTS) && (error_details
    ↪  != ERROR_ACCESS_DENIED)) {
      return mutex_h;
    }
    hwnd_other_pangya =
    ↪  FindWindowA((LPCSTR)&lpClassName_00ef9a88,(LPCSTR)0x0);
    if (hwnd_other_pangya != (HWND)0x0) {
      SetForegroundWindow(hwnd_other_pangya);
      SetActiveWindow(hwnd_other_pangya);
    }
  }
  FUN_00a623f0('\0');
  return (HANDLE)0x0;
}
```

Listing 2: Code from the TH client performing the Mutex exclusivity - Reverse engineered with Ghidra.

3.4 Required Data

Upon startup, the client initiates the download of essential data from the Content Delivery Network (CDN), which serves as a static file server. Within this section, we will delve into the specifics of the required data, outlining its significance and purpose.

This data is comprised of crucial information for the proper functioning of the client. Conversely, there are also non-essential data that, if not retrieved or found to be corrupted/unreadable, can be safely disregarded by the client.

In the event that any of the essential data cannot be retrieved and processed correctly, the client will promptly issue a relevant warning alerting the user of the issue.

Subsequently, the client will cease its execution to prevent any potential operational disruptions or errors.

3.4.1 Translation XML

Part of the user interface in the game relies on a file hosted in the CDN, which provides translations upon startup.

This file is a Base64 encoded and malformed XML that contains messages intended for display to the user. Its malformation stems from containing multiple roots, a scenario not typically allowed in XML.

The structure is straightforward, consisting of a key and its corresponding text. Notably, the file incorporates `HTML4` elements, particularly the `<font>` tag, which aligns with the game's development era predating `HTML5`. An extract of the aforementioned file can be seen in Listing 3.

```
<TEXT>
    <KEY>100502</KEY>
    <DEV></DEV>
    <SERVICE>Check
        <font color="#ff0000">My Room</font> for the prize.
    </SERVICE>
</TEXT>
<TEXT>
    <KEY>2014061901</KEY>
    <DEV></DEV>
    <SERVICE>Play to Win Event  Rewards items.</SERVICE>
</TEXT>
<!-- ... -->
```

Listing 3: Decoded "read.aspx" from the original US' CDN.

In the absence of this file, the client will cease execution and display the error message that can be seen in Figure 7.

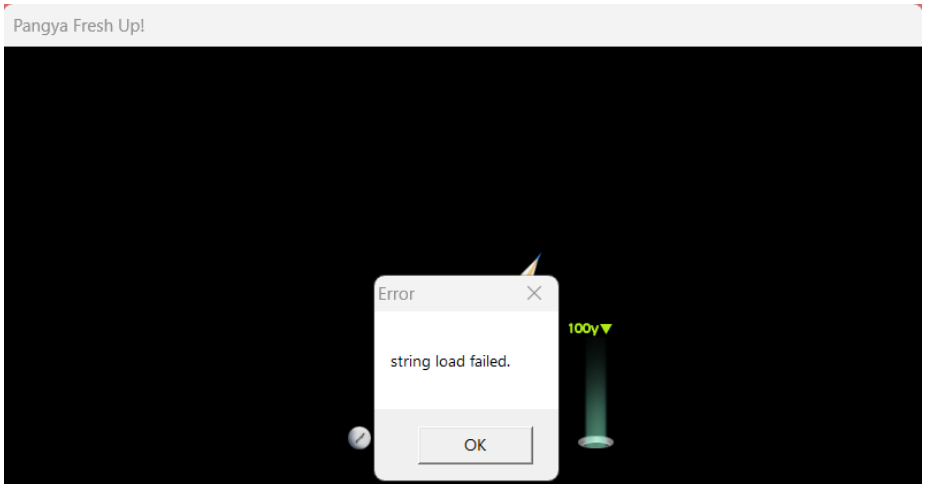

Figure 7: The client shows a cryptic message that doesn't give much explanation.

As can be seen, the message doesn't provide a user-friendly explanation of the issue to the user. It appears that this message may not have been intended for regular users, unlike other error messages that seem to be tailored for user comprehension.

3.4.2 File Integrity

To check the integrity of the game's virtual file system the client also depends on a file hosted in the CDN, which is downloaded on startup.

This file is called "updatelist", and you probably remember it from one of the previous sections. It contains a malformed XML file, and it is encrypted with a modified XTEA. The XTEA used can be considered insecure both because of the current attacks available and the number of rounds this modification uses. [23]

Each region has a key, in specific the US region we are working on uses the following one {`0x03F607A9, 0x036F5A3E, 0x011002B4, 0x04AB00EA`}.

Here we showcase the modifications that can be seen in its algorithm (in Listing 4) compared to the one released in the reference code for XTEA:

1. Instead of modifying the same buffer directly it relies on 2 buffers, one for the cryptogram and the other for the plain-text.
2. The number of rounds is hard-coded to 4, an insecure amount of rounds.
3. Instead of subtracting the delta from the sum, it is added.
4. The initial value of "sum" is a random hard-coded value instead of being the multiplication of the delta and the number of rounds.

```
void __fastcall xtea_decrypt(uint32_t *cryptogram, uint32_t
↪  *plaintext, int *key)
{
    const int rounds = 4;

    unsigned int delta = 0x61C88647;
    unsigned int sum = 0xE3779B90;
    unsigned int v0 = cryptogram[0];
    unsigned int v1 = cryptogram[1];

    unsigned int i;

    for (i = 0; i < rounds; i++) {
        v1 = v1 - ((v0 >> 5 ^ v0 << 4) + v0 ^ key[sum >> 0xb & 3]
        ↪  + sum);
        sum += delta;
        v0 = v0 - ((v1 >> 5 ^ v1 << 4) + v1 ^ key[sum & 3] +
        ↪  sum);
    }

    plaintext[0] = v0;
    plaintext[1] = v1
    return;
}
```

Listing 4: Code from TH client - Demystified Pangya's modified XTEA deciphering function - Reverse engineered with Ghidra.

After applying the algorithm to the original US' CDN "updatelist" we find the structure in Listing 5.

After taking a careful look, comparing it to what we find on the client's files and what the community currently knows, here are the meanings of each attribute: [24]

- `patchVer`: Patch's version
- `patchNum`: Number/nonce/index of the patch
- `updatelistVer`: Version of the updatelist's structure (basically the date of the last structure change)
- `updatefiles`: The amount of files to check
- `fileinfo`: Information to perform checks on each file:
  - `fname`: File name.

```
<?xml version="1.0" encoding="euc-kr" standalone="yes" ?>
<patchVer value="GB.R7.852.00" />
<patchNum value="306" />
<updatelistVer value="20090331" />
<updatefiles count="258">
    <!-- ... -->
    <fileinfo fname="projectg851gb.pak" fdir="" fsize="690312"
    ↪  fcrc="-90408369" fdate="2016-04-25" ftime="08:57:49"
    ↪  pname="projectg851gb.pak.zip" psize="682029" />
    <!-- ... -->
</updatefiles>
```

Listing 5: A more detailed extract from the original US' CDN decrypted "updatelist"

- `fdir`: Directory on which the file should be (an empty string means the game's root directory).
- `fsize`: Size of the file.
- `fcrc`: CRC32 checksum of the file represented as a signed integer (using the reversed IEEE polynomial `0xEDB88320`). [25]
- `fdate`: UTC date of the file's last modification.
- `ftime`: UTC time of the file's last modification.
- `pname`: The name the files have when packaged in the CDN for the updates the launcher performs.
- `psize`: The size it has when packaged.

I was unable to personally validate the following claims, but based on the trustworthy knowledge shared by John Chadwick and others in the `reetrev` group, this is the general process the client follows when handling the "updatelist":

1. The client downloads the latest "updatelist" from the content delivery network (CDN).
2. It scans through the list of packages specified in the "updatelist".
3. For each package, the client validates the CRC32 checksum using the previously mentioned reversed IEEE polynomial. [25]
4. The client then loads all the packages into the virtual file system in the order specified by the "updatelist", overlaying almost all the files with the same filename on top of the older ones; but some of them do seem to care about the directory structure.

Another claim I was unable to validate, documented by John Chadwick and Andreas Nedbal in the `reetrev` group, is that if you pass an "updatelist" with no `fileinfo` tags to the client, it will enter what seems to be a possible developer mode.

In this mode, instead of loading the packages, the client will attempt to load raw files from a `data/` folder, expected to be in the same location as the client's executable. However, it was noted that some files, such as sound effects, may not load correctly in this mode.

Also, there are various open-source libraries and tools available that provide implementations to work with "updatelists". [26, 27, 28]

Additionally, there is an official tool called `QuickPatch` that was allegedly leaked unintentionally in one of the updates the TH region did. This tool is capable of generating patches for the launcher and uploading them to an FTP server, which was most probably the CDN of the publisher. [29]

3.5 Anti-cheat Bypass - GameGuard

To preserve the functionality of the game we need to bypass the game's anti-cheat system, GameGuard. Here is why:

- The version of the anti-cheat used in the game does not support Windows 11 and can cause the entire system to crash when running the game.
- If we want to run the game on Linux (through Wine) we will also need to defeat it.
- GameGuard also requires access to a GameGuard update server to function properly. If it is unable to connect to the update server and perform the necessary verification routines upon the downloaded configuration file, the anti-cheat will prevent the game from continuing its execution. This poses a significant challenge in preserving the functionality of Pangya, as there are no more official servers serving this files.
- The anti-cheat blocks most tries to read and write to the memory of the client, which becomes a nuisance if we want to implement possible changes and enhancements.

To achieve these goals, we will need to find a way to bypass the GameGuard anti-cheat system. This may involve techniques such as reverse engineering the anti-cheat mechanisms, identifying and patching the vulnerable components, or potentially developing a custom solution that can emulate or replace the functionality of GameGuard while allowing the game to run without issues on the target platforms and enabling the necessary memory access for modding purposes.

But there is also another nuisance, almost all regions, including the US client, are packed using Themida, a known commercial packer that embeds several

features like anti-debugging, virtual machine emulation, and encryption. This difficulties any binary patching of the executable, requiring to do this patches on runtime once the real game gets unpacked to memory by Themida.

Fortunately, previous contributors have explored various approaches to bypass these obstacles:

1. Manually removing Themida and patching the client to disable GameGuard checks and start-up. This involves dumping the client after Themida's decryption and decompression of the real game's instructions. There are multiple files out there from different contributors, some with malware added.

2. For older versions of GameGuard, redirecting all communication to the GameGuard's update server to your own update server (a simple HTTP file server) by using the Windows' hosts file. This avoids modifying the client directly. This approach was done by Eric Antonio. [30]

3. Manually modifying both the client and anti-cheat binaries to keep Themida and GameGuard, but in a way that preserves memory integrity checks. This also involves customizing the RSA keys used for GameGuard's configuration file and, if you want to keep the GameGuard heartbeat function, running on the game server's side a GameGuard Authentication library. This work was done by Acrisio Filho, is partially private and was used for his "SuperSS" private server. [31]

4. Using a lightweight shim for `ijl15.dll` called `rugburn` that disables and nullifies the GameGuard checks, performs network traffic redirection, and sets the necessary game arguments. It uses a mix of function hooking and memory manipulation on runtime. This is the least invasive approach. This solution was developed by John Chadwick, and is the only considered to be well documented and open-source. [32]

From the various bypass approaches explored by previous contributors, the least invasive and most promising solution appears to be `rugburn` developed by John Chadwick. This approach has several advantages:

- It avoids the need to directly modify or patch the game client or anti-cheat binaries, which can be a complex and error-prone process, especially given the use of Themida packing.

- It operates by hooking and manipulating the necessary functions at runtime, rather than making permanent changes to the game files. This allows for a more flexible and maintainable solution that can potentially be updated or adapted as needed.

- It is well-documented and open-source, providing transparency and the opportunity for further development and improvement by the community.

In contrast, the other approaches, such as manually removing Themida and patching the client, or redirecting the GameGuard update server connection, require more invasive and potentially risky modifications to the game's internals. These solutions may also be more difficult to maintain and update over time, as they are often specific to particular game versions or rely on leaked or private resources.

Therefore, `rugburn` appears to be the most suitable and practical solution for bypassing the GameGuard anti-cheat system in Pangya, as it offers a lightweight, non-invasive, and well-documented approach that can be leveraged to preserve the game's functionality on modern systems.

3.6 Redirecting Connections

To successfully revive Pangya, we will need to redirect both the HTTP connections to the game's CDN, as well as the TCP connections to the game servers, as these original infrastructure components are now defunct.

There are a few different approaches that can be considered for this connection redirection:

- Hosts File Redirection: One option is to use the Windows hosts file to redirect all the connections to the defunct servers to an alternative location. This is a relatively simple solution, but it requires modifying the hosts file on each target system, which may not be practical for large-scale deployment.
- Binary Patching: Another approach is to unpack the game client and manually patch the hard-coded IP addresses and domain names within the binary. This would involve carefully modifying the string values to point to the new, replacement infrastructure. However, this method can be error-prone and may require extensive testing to ensure compatibility with different game versions.
- Runtime Hooking: The most flexible and maintainable solution appears to be the approach taken by `rugburn`, which hooks the relevant Windows API functions (such as `InternetOpenUrlA` and `connect`) at runtime and modifies the target addresses on the fly. This allows for a centralized and dynamic redirection mechanism that can be easily updated and adapted as needed, without requiring direct modifications to the game client.

Given the advantages of the runtime hooking approach `rugburn` uses, such as its flexibility, maintainability, and non-invasive nature, this appears to be the most suitable option for redirecting the necessary connections in the effort to preserve Pangya's functionality.

3.7 Communication Protocol

To understand how the Pangya client communicates with the game's servers, it is important to examine the underlying protocol and mechanisms used for

authentication, state synchronization, and other essential functionalities.

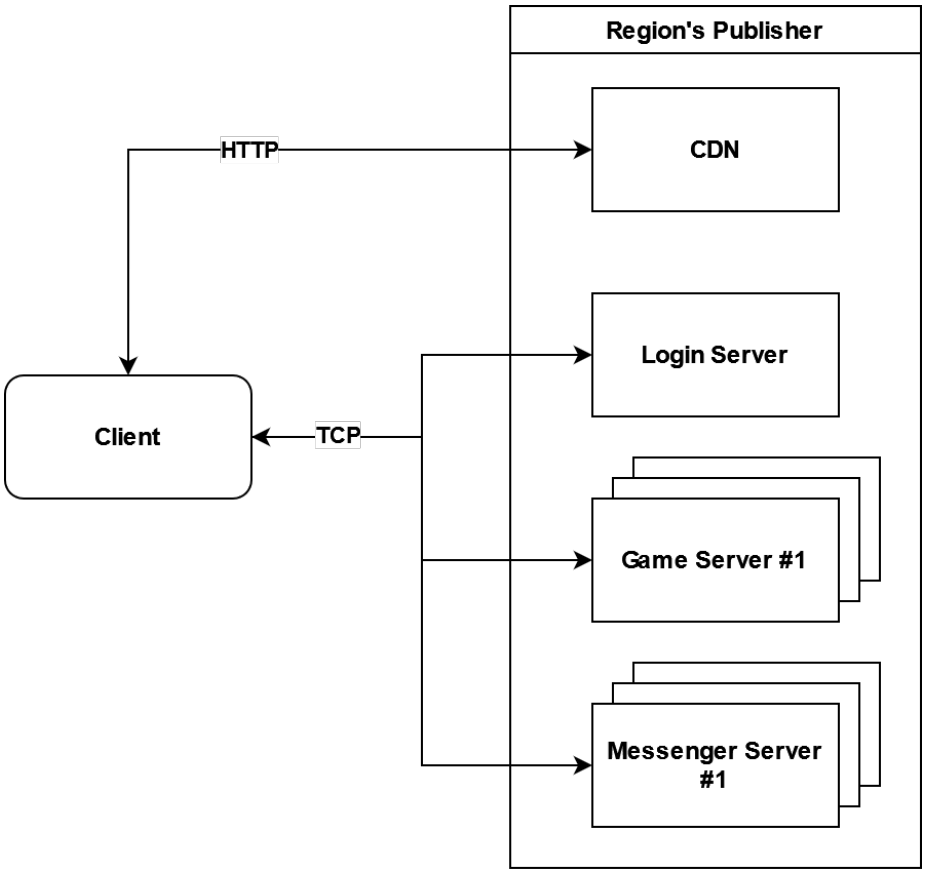

Figure 8: The basic architecture of the services that are exposed to the client.

As illustrated in Figure 8, the game client primarily utilizes two types of communication channels:

- HTTP-based CDN Connections: The client downloads various required and optional game files from a content delivery network (CDN) using standard HTTP requests. These downloads occur during the game's startup and throughout the gameplay session as needed.
- Custom TCP-based Protocol: In addition to the HTTP-based CDN connections, the client communicates with the game servers using a custom TCP-based protocol. This protocol is more complex and serves critical functions such as client authentication and game state synchronization with other players.

3.7.1 Obfuscation

The custom TCP-based protocol follows a specific handshaking process:

1. The client connects to a server's TCP socket and waits for an introductory packet.

2. The introductory packet's structure varies depending on the type of service, but it typically contains an obfuscation key or index. This obfuscation key is used to determine the appropriate part of an "oracle" that will be used to obfuscate future packets in the communication.

3. The obfuscation key or index value ranges from 0 to 15, allowing for 16 possible obfuscation configurations.

From the initial handshake and establishment of the obfuscation key, the client then proceeds to authenticate itself, utilizing the obfuscation scheme for the subsequent communication. Instead of sending data in a raw, unprotected format, the protocol employs a "frame" structure that precedes the obfuscated payload.

The frame format differs depending on the sender of the message:

- **Client Frame**: The client's frame is 5 bytes long.

    1. **Salt (1 byte)**: The first byte is a randomly generated salt.
    2. **Payload Length (2 bytes)**: The second and third bytes represent the length of the payload data that follows the frame. This length is calculated by subtracting the frame's fixed length of 5 bytes to the entire data's length and then adding 1 byte. The length is stored as an unsigned short (2 bytes).
    3. **Padding (1 byte)**: The fourth byte seems to be a reserved or padding byte.
    4. **Obfuscation Byte (1 byte)**: The fifth and final byte of the frame contains a value derived from the CryptTable2, which is accessed through an oracle index. This oracle index is calculated using the salt value (first byte) and the obfuscation key that was provided during the initial protocol establishment. It is used to start the XORing of the payload. [16]

- **Server Frame**: The server's frame, on the other hand, is a little bit more complex and builds upon the client frame, being 8 bytes long.

    1. **Client Frame (5 bytes)**: The first 5 bytes are the same except for the payload length, which is calculated by subtracting 3 bytes from the entire packet (frame + payload). In this case the Obfuscation Byte seems to lose meaning, as it doesn't affect the obfuscation of the payload.

2. **Packet Length Obfuscation (3 bytes)**: The last 3 bytes of the server's frame contain a peculiar obfuscation applied to the length of the entire packet. This additional obfuscation step is likely intended to make reverse-engineering the protocol more difficult. [16]

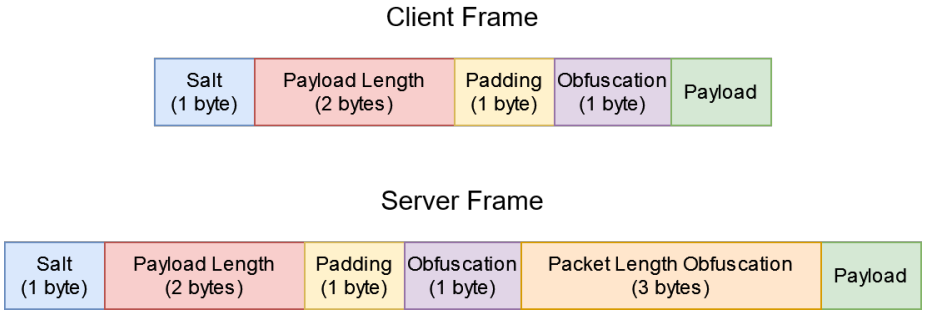

Figure 9: A representation of the frames used in the protocol.

The payload, depending on the frame type, starts the obfuscation at different points. This allows the receiving end to identify the packet type before fully deobfuscating the payload.

For the Server Frame, the obfuscation of the payload starts at the 3rd byte. This means the first 2 bytes of the payload are not obfuscated, and the client can use these bytes to identify the packet type. This is typically an unsigned short value that distinguishes each packet from the others.

The Client Frame starts the obfuscation of the payload at the 4th byte. The first 3 bytes of the payload are left deobfuscated, again allowing the server to identify the packet type before fully deobfuscating the rest of the payload.

This design choice could serve two purposes:

- It enables the receiving end to quickly identify the packet type and handle it appropriately, without the need to fully deobfuscate the entire payload first.
- The difference between the 2 frames' obfuscation adds an additional layer of confusion to the protocol, making it more difficult to reverse-engineer and understand.

3.7.2 Security Concerns

But after analyzing its protocol, it is evident that there are significant security vulnerabilities present. Conducting a MITM (man-in-the-middle) attack on the communication protocol is relatively straightforward, given that the only critical point of security lies in the obfuscation key/index transmitted during

the introductory packet. With that and the "crypto orcale", which can be extracted from the client, you are free to do whatever you want.

Furthermore, the independence of each packet within the communication stream makes it susceptible to manipulation, addition, or omission. This flaw facilitated the interception of communications between original clients and official servers, as well as the development of bots capable of monitoring player activity on a server (as demonstrated in the time-series tracking the player count on the final official Pangya server before its closure). [33, 34, 35]

To safeguard the integrity of the game, it is imperative to enhance the security of its communications. Implementing a more robust transport protocol, such as TLS or QUIC, could provide the necessary encryption and authentication mechanisms to fortify the communication channels effectively. These protocols have undergone extensive testing and can reliably verify the server's identity, thereby mitigating the risks associated with unauthorized access and manipulation of data.

Additionally, it is worth noting that discussions are underway with John Chadwick to define an abstraction layer over the original protocol. This abstraction aims to facilitate the integration of more secure transport mechanisms, ensuring that the game's communications are adequately protected against potential security threats, and will be potentially added at some point to `rugburn`.

3.7.3 Payload Structure

Now I will talk about the structure of payloads in their deobfuscated state.

From what we know and have observed, all payloads start with an unsigned short value (2 bytes) that serves as an identifier or packet type discriminator. This identifier varies depending on the type of service the client is communicating with, and each transceiver (client and server) has its own set of identifiers, which can overlap but have different meanings.

Both this identifier and the rest of the payload are in little-endian format. We have observed the use of the following primitive data types in the payloads: UInt64, Int64, UInt32, Int32, UInt16, Int16, SByte, Byte, Boolean, and Float (aka. Single).

Apart from these primitive types, there are also more complex data structures, such as strings. Two types of strings are used: pstrings and fixed-size strings.

Pstrings are Pascal-style length-prefixed strings, where an unsigned short value (2 bytes) precedes the string data, indicating the length of the string. This allows for variable-length string representations.

Fixed-size strings, on the other hand, have a predetermined length which seems to be implicit, already known by both sides, and the string data is stored directly without any length prefix.

While the overall payload structure is pretty straightforward, a single payload type can have multiple forms, what I call "polymorphic payloads", depending on the values it contains.

For example, an authentication payload may be able to represent both success and failure scenarios. In the case of a failure, the payload may only need to include an error code, but in the case of success, the payload may need to include additional information that identifies the player.

This polymorphic nature of the payloads adds a layer of complexity to the protocol, as the receiver needs to be able to interpret the payload correctly based on the context and the specific values it contains.

Understanding and properly handling these polymorphic payloads is crucial for effectively parsing and processing the communicated data on both the client and server sides.

3.7.4 Service Types

The PangYa server stack, as illustrated in Figure 8, is divided into several distinct network services, each with its own specific responsibilities:

- **Login Server**: This is the first service the client must interact with to authenticate itself. Once the authentication is successful, the Login Server provides the client with detailed information about all the available Game Servers. If the client selects a specific Game Server, the Login Server will issue a "Game Session Key" that the client can use to join the selected Game Server while maintaining its identity.
- **Game Server**: This is the most complex service in the system, as it is responsible for synchronizing the game state across multiple clients. The Game Server also provides clients with a list of available Message Servers upon request.
- **Message Server**: This service handles the social capabilities of the game, such as allowing users to send messages to other players who are not present in the same Game Server, as well as managing the players' friend lists.

In addition to these core services, there is also a **Ranking Server** that provides players with a leaderboard functionality. This Ranking Server was not present in the unmodified US version of the game, but it can be re-enabled through certain memory modifications. [36]

The division of responsibilities across these different network services allows for a more modular and scalable architecture, where each component can be optimized and scaled independently to meet the demands of the game's player base. This separation of concerns also simplifies the development and maintenance of the overall system.

With current technologies, different approaches could be taken to leverage the system's topology better. For instance, the Message Server could be removed, and the Game Server could instead use a pub/sub system to communicate with other Game Servers and provide the same functionality.

This would allow the client to use a single socket connection to the Game Server, which would then handle the communication with other servers as needed. This approach could simplify the client-side implementation and reduce the overall complexity of the system. However, such changes would require a more significant rework of the existing architecture and would be better suited for future developments on the abstraction layer mentioned in the previous sections.

3.8 IFF Files

IFF Files are likely named after the Interchange File Format (IFF), even though they have meaningful differences from the standard and are more akin to the Resource Interchange File Format (RIFF). They can be found inside a zip file called `pangya_gb.iff` and are one of the most important types of files contained in the game packages.

The main reason for this is that they are a mix of data and metadata that forms all the basic "representations" in this game. This means that the maps where players golf, the characters they can play with, the stats their golf clubs have, and many other game elements need to be defined in these files. If they are not present, these elements will technically not exist, even if the associated textures and models are available in the packages.

These files provide a foundation from which the client can load all the necessary resources, such as their unique identifier, in-game name, the models and textures they are related to, and other specific properties for the type of object they represent. For the emulation effort, the most interesting aspects are the specific properties that delve into the game's mechanics surrounding these objects.

It also needs to be noted that some information that should be for the servers only is also present in the IFF files, like the probabilities with which enhancements on clubsets should succeed or not. This makes it plausible that the original servers also used these IFF files as a source of truth for certain game mechanics and parameters.

All IFF files write their primitives using little-endian and share a common header structure:

- **Number of records** (2 bytes): indicates the number of entries/records in the file.
- **"Binding Id"** (2 bytes): seems deprecated, as files from EU301 revealed it wasn't written to when IFF files were generated. Possibly a revision specific identifier.

- **Version** (4 bytes): represent the version of the IFF file structure, although some versions may overlap with other regions' modifications, making it less useful.

Once this header is written, the next data that is written are all the records the IFF file should have (based on the Number of records defined earlier).

Some types of records have a common structure before their custom one, which provides a baseline set of properties that are shared across many types of game objects. Here we will showcase some of the fields it has: [37, 38]

- Active Item (4 bytes): Indicates if the item is up for sale or not.
- Id (4 bytes): The unique identifier for the item.
- Name (40 bytes): The name of the item.
- Level (1 byte): The minimum level required for the user to use the item.
- Icon (40 bytes): The filename of the icon associated with the item.
- Price (4 bytes): The price of the item.
- *for more, please take a look at the previous references*

After this common structure, the record will have its custom structure that defines the specific properties of the game object being represented. The format and contents of this custom structure will vary depending on the type of object (e.g. golf club, character, map).

About the earlier commented files from the version EU301, John Chadwick realised that the IFF files contained memory sanitization, so unwritten bytes were set to 0xCC, which allowed us to see which fields were used or unused. This gives a lot of insight on how they were really structured, and what things were padding or not. Which made us see that "Binding Id" was deprecated at some point, or just padding.

4 Design and Implementation

4.1 Content Delivery Network

To start up the client, I initially used `rugburn` to redirect all HTTP requests to a local file server. This allowed me to quickly setup a test environment that could provide the required "updatelist" and "read.aspx" files, which are essential for the proper functioning of the client. Both are the original ones from the US CDN, as I do not need to perform any modifications to the game for the time being.

```
{
  "UrlRewrites": {
    "http://[a-zA-Z0-9:.-]+/new/Service/(.*)":
    ↪  "https://pangya-static.xyz.dev/$0",
    "http://[a-zA-Z0-9:.-]+/(.*)":
    ↪  "https://pangya-static.xyz.dev/$0"
  }
}
```

Listing 6: Example JSON config file from `rugburn` with the Regex applied to the HTTP requests.

For the public version of the project, the one used for tests with friends of mine, the content is hosted using a combination of hosted virtual machines (VMs) and Cloudflare's R2 storage service (an AWS S3 equivalent). The lighter assets are hosted on the VMs through a CaddyServer, while the heavier game packages that need to be downloaded by the launcher are stored on Cloudflare R2.

This hybrid approach allows for efficient content delivery and cost management, with the VMs handling the more lightweight assets and Cloudflare R2 providing scalable storage and distribution for the larger game packages. By leveraging both hosting solutions, I can ensure a smooth and reliable user experience for the game's players.

4.1.1 Launcher

For the launcher, I've decided to use a cross-platform desktop application framework called Tauri. Tauri is similar to Electron in that it allows you to build desktop applications using web technologies like HTML, CSS, and JavaScript. However, instead of using Node.js as the backend, Tauri leverages the Rust programming language.

By using Tauri, I can separate the user interface, which is built using Vue 3 and TypeScript, from the backend logic that handles the installation and update processes. This separation of concerns allows me to take advantage of Rust's

performance and security benefits while still providing a modern, web-based user experience.

The current version of the launcher is capable of performing the initial installation of the game by downloading from the CDN, but it does not yet include features for incremental updates or comprehensive integrity checks with repair functionality. These are important capabilities that I plan to implement in the future, once the server-side infrastructure is more feature-rich and publicly accessible.

The Tauri framework also includes a built-in "updater" subsystem that allows the launcher to update itself autonomously. By configuring this feature, the launcher can check for updates on startup by downloading a JSON file that contains the necessary information to determine if the current version is up-to-date. If an update is available, the launcher can then download and install the latest version seamlessly.

In addition to the self-updating capabilities, I have made the launcher also update a specific file on startup: the `rugburn` configuration file. This file provides the necessary redirections for both HTTP requests and the initial TCP connection to the Login Server. By keeping this configuration file up-to-date, the launcher ensures that `rugburn` properly routes all the necessary network traffic, even if the underlying server infrastructure or domain names change over time.

A screenshot of the launcher can be found in Figure 10.

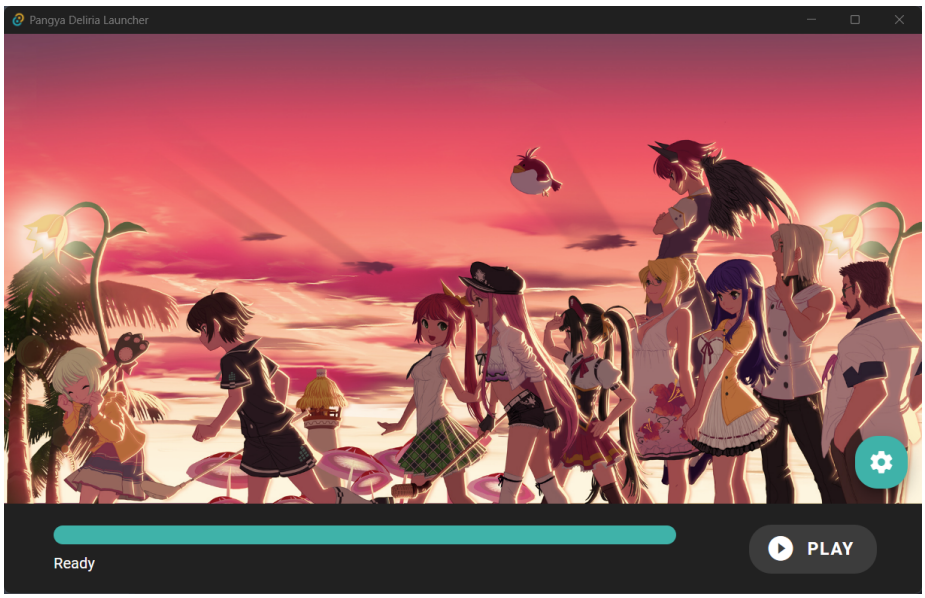

Figure 10: A screenshot from the main menu of the launcher.

In the future, once I make the server more featureful and public, I will enhance the launcher to properly perform the missing and necessary tasks, such as:

- Incremental updates: The launcher will be able to detect and download only the necessary game files that have been updated, rather than requiring a full reinstallation.
- Integrity checks and repair: The launcher will validate the integrity of the installed game files using the information provided in the "updatelist" file. If any files are found to be corrupted or missing, the launcher will automatically download and replace them.
- Seamless user experience: The launcher will provide a user-friendly interface that guides players through the installation and update process, minimizing any disruptions or confusion.

By leveraging the Tauri framework to make a robust and reliable launcher, its built-in updater, and dynamically updating the "rugburn" configuration, the launcher can maintain its functionality and provide a reliable and seamless experience for players, even as the game's backend infrastructure evolves.

4.2 System Architecture Design

For the implementation, I decided to use `C#` instead of the original `C++` language used in Pangya. This choice was primarily driven by my previous experience with `C#` and its higher-level features, which can potentially improve development productivity.

Despite the language change, I aim to maintain the basic exterior topology of the original server architecture to avoid the need for significant changes to the existing protocol. This decision is based on the primary focus of my preservation efforts, which is to achieve the game's basic functionality rather than to optimize its efficiency and performance.

The system architecture will utilize a combination of different database technologies to store and manage the necessary data:

- **Relational Database**: A PostgreSQL database will be used to store player accounts and their associated information.
- **Key-Value Database**: A Redis database will be employed to store session keys, which are crucial for player authentication between the Login Servers and Game Servers. The use of a key-value database is advantageous due to its Time To Live (TTL) feature, which allows for the easy removal of temporary data, similar to how cookies work in web browsers.

To facilitate the interaction between the various components, an intermediary service called `DbService` will be implemented. This service will act as a connection pool, indirectly managing the interactions with the SQL database.

By using this intermediary service, the Login Servers and Game Servers can efficiently communicate with the database without directly handling the expensive SQL connections.

The communication between the different server components (Login Servers and Game Servers) will be facilitated through Message Queues, using RabbitMQ. This approach allows for the implementation of a scalable and flexible architecture, where multiple `DbService` instances can handle requests in parallel or even be specialized into microservices in the future.

The Message Queues will also be used to enable a "heartbeat" communication, where the Game Servers can periodically notify the Login Server about their availability. This allows the Login Server to maintain a list of healthy Game Servers on the fly from which the players will be able to choose.

A diagram of the architecture we just described can be found in Figure 11.

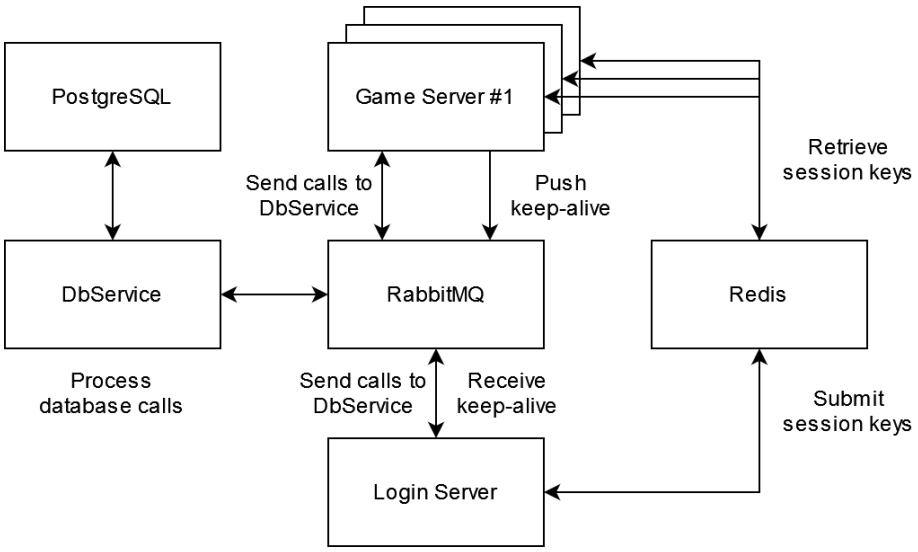

Figure 11: A diagram representing the architecture of the entire system.

Before proceeding with the implementation, I will need to focus on implementing the communication protocol. This is a crucial step, as it will enable the effective deserialization and serialization of the payloads received and sent between the client and the server components. Without a properly implemented communication protocol, it would be impossible to establish a functional communication channel.

4.3 Shared Libraries

To enable effective communication and reuse of common functionalities across the different programs (client, login server, and game server), a set of shared

libraries have been developed. These libraries encapsulate various utilities, communication protocols, and data abstractions that are essential for the Pangya game server emulation.

By providing these shared libraries, the development of the client, login server, and game server components can be streamlined, as they can leverage the common utilities and communication protocols without the need to reimplement them from scratch. This approach promotes code reuse, maintainability, and consistency across the different parts of the Pangya game server emulation project.

4.3.1 Common Utilities

The common utilities library provides a collection of general-purpose functions and classes that can be used across the different components of the system. This includes:

- **IRandomGenerator**: An interface that defines a contract for generating random numbers, allowing for easy substitution of different random number generation implementations.
- **LogExtensions**: A set of extension methods that enhance the logging capabilities, making it easier to log messages with additional context information using `Serilog`.
- **RandomGenerator**: A concrete implementation of the `IRandomGenerator` interface, providing a default random number generation mechanism.
- **ServiceCollectionExtensions**: Extension methods for the .NET `IServiceCollection` interface, simplifying the registration of services in the dependency injection container.
- **ServiceProviderExtensions**: Extension methods for the .NET `IServiceProvider` interface, providing utility functions for resolving services.

4.3.2 Internal Communication

The internal communication library focuses on facilitating the exchange of messages between the different server components, primarily using the MQTT protocol (and through the library MassTransit). This library includes the following classes:

- **AccountGet***: A set of classes representing requests and responses for retrieving account information, such as by ID or username.
- **InventoryAdd*, InventoryDelete*, InventoryGet*, and InventoryUpdate***: Classes that handle various inventory-related operations, such as adding, deleting, retrieving, and updating items and caddies.

- **PlayerCreate, PlayerGet*, and PlayerUpdate**: Classes that encapsulate player-related operations, including creating, retrieving, and updating player data.
- **CommunicationServiceExtensions**: Extension methods that simplify the registration and configuration of the communication-related services.
- **IServiceStatusHolder and ServiceStatusHolder**: Interfaces and implementations for managing the status of the various game services, enabling the tracking of their availability.
- **Account/Caddie/Character/Item/Mascot/Player/PreferencesDto**: Data transfer object (DTO) classes that represent the different entities within the Pangya game, facilitating the serialization and deserialization of data.
- **GameServiceStatus and ServiceHeartbeatGameServiceResult**: Classes that represent the status of the game services and the results of the heartbeat communication, respectively.

4.3.3 SQL Database

This library provides the basic models, repositories, and database migrations for the `DbService`. It includes the following components:

- **PlayerEntity** and **PlayerEntityConfiguration**: The entity class and its configuration for the player data, including properties such as player ID, nickname, and account ID.
- **AccountEntity** and **AccountEntityConfiguration**: The entity class and its configuration for the account data, including properties such as the account ID, username, and password.
- **20220101142633_FirstStep.cs**: A database migration file that sets up the initial schema for the player and account entities.
- **PangyaContextModelSnapshot.cs**: A snapshot of the current database model, used for database migrations.
- **AccountRepository** and **PlayerRepository**: Repository classes that provide data access and manipulation methods for the account and player entities, respectively.
- **IRepository**: An interface that defines the contract for the repository classes, allowing for easy substitution of different repository implementations.
- **DatabaseServiceExtensions**: Extension methods for registering the database-related services in the dependency injection container.

- **PangyaContext**: The main database context class that manages the lifecycle of the database connections and entities.
- **SchemaNames**: A class that holds the names of the database schemas used in the Pangya game server emulation.

4.3.4 Event Handler

The event handler library provides a way for the game servers to decouple the handling of payloads or packets from the events they can cause. It includes the following components:

- **EventPipeline**: The main class that manages the registration and execution of event handlers.
- **EventServiceExtensions**: Extension methods for registering the event-related services in the dependency injection container.
- **IEventAsyncHandler** and **IEventSyncHandler**: Interfaces that define the contracts for asynchronous and synchronous event handlers, respectively.
- **IEventPipeline**: An interface that defines the contract for the event pipeline, allowing for easy substitution of different pipeline implementations.

4.3.5 IFF Files

The IFF files library is one of the first libraries to be released to the public, indirectly through contributions to the `PangLib.IFF` library from the `retreev` group. It facilitates data extraction from IFF Files, specifically the ones for the US version. It includes the following components:

- **VersionType** and **RegionType**: Enumerations that represent the version and region of the IFF files, respectively.
- **IIffModel**: An interface that defines the contract for IFF file models, ensuring consistency across different types of IFF file representations.
- **IffFile** and **IffInfo**: Classes that represent the IFF file structure and metadata, respectively.
- **StatsStruct**, **SystemDateTime**, **PartType**, **CardEffectType**, **ShopFlagType**, and other similar classes: Structures and enumerations that represent the various data types and properties found in the IFF file records.
- **IffPart**, **IffItem**, **IffCourse**, **IffCommon**, **IffCommonItem**, **IffClubSet**, **IffCharacter**, **IffCaddie**, and **IffBall**: Classes that represent the specific IFF file record types, encapsulating their custom data structures.
- **IffCourseExtensions**: A set of extension methods that provide utility functions for working with course-related IFF file records.

4.3.6 Key-value Database

The key-value database library provides common utilities and functionality related to the use of Redis, a popular in-memory data structure store. It leverages the StackExchange.Redis library as its underlying implementation.

This library includes the following components:

- **GameSessionKeyManager** and **LoginSessionKeyManager**: Classes that handle the storage and retrieval of session keys for the game and login servers, respectively. These session keys are crucial for player authentication between the different server components.
- **StatusManager**: A class that manages the status information for the various game services, such as the availability of the game servers.
- **RedisConfiguration**: A class that encapsulates the configuration settings for the Redis database, making it easier to manage and inject these settings into the application.
- **RedisManager**: The main class that provides a centralized interface for interacting with the Redis database, abstracting away the low-level details of the `StackExchange.Redis` library.
- **RedisServiceExtensions**: Extension methods that simplify the registration and configuration of the Redis-related services in the dependency injection container.
- **IRedisScopedLockService** and **RedisScopedLockService**: An interface and its implementation that provide a distributed locking mechanism using Redis. This is useful for ensuring atomicity in complex operations that require coordination across multiple server instances, like session keys.

4.3.7 Network Manager

The development of the Network Manager library posed significant challenges due to the polymorphic nature of the payloads/packets in the Pangya Protocol. Achieving proper and user-friendly serialization and deserialization utilities in the presence of polymorphism was particularly demanding.

Despite these challenges, I was able to implement a robust serialization mechanism that effectively handles polymorphism. However, deserialization, especially when dealing with polymorphism, proved to be more complex and less streamlined. Nevertheless, the library has proven to be valuable during the development of both the Login and Game Server components.

In hindsight, I acknowledge that utilizing Source Generators instead of Reflection for the library could have been a more efficient approach. Source Generators offer improved performance by generating code at compile time, eliminating the need for runtime analysis of metadata and dynamic instantiation of classes.

While I have not yet had the opportunity to transition to Source Generators, it remains a potential enhancement for future iterations of the library.

Beyond Serialization and Deserialization functionalities, the Network Manager library provides essential tools for packet processing through pipelines, packet rate limiting over specified time spans, and a mechanism for defining new payload structures using classes and attributes.

These structure definitions, housed in a separate library, seamlessly translate to the various primitive data types utilized in the Pangya Protocol. For instance, consider the following example shown in Listing 7.

```
[PacketIdentity(PacketType.Game,
↪  (ushort)ServerGamePacketType.GamePrizeList)]
public class PacketGsGamePrizeList : IServerPacket
{
    [PacketIndex(0)]
    public ushort PlayersCount => (ushort)PlayerDrops.Count;

    [PacketIndex(1)]
    public List<SubPacketPlayerDrops> PlayerDrops { get; init; }
}

public class SubPacketPlayerDrops : ISubPacket
{
    [PacketIndex(0)]
    public int ConnectionId { get; init; }
    [PacketIndex(1)]
    public byte Option { get; init; }
    [PacketIndex(2)]
    public ushort DropCount => (ushort)DropItemIds.Count;
    [PacketIndex(3)]
    public List<int> DropItemIds { get; init; }
}
```

Listing 7: Example of a real packet structure, in this case this packet is the one that is sent after the end of the plays, showing the player the prizes he/she has obtained.

In addition to serialization and deserialization capabilities, the library includes optimized versions of PangCrypt's obfuscation algorithm. These modifications focus on enhancing algorithm performance by reducing memory allocation and simulating the costly compression algorithm, albeit at the expense of increased payload size. The library also offers utilities to simplify the management of TCP sessions for player connections.

Despite the library's extensive functionality, there are crucial components that are currently absent, leading to duplication in both the Login and Game

Servers. Specifically, the direct handling of raw TCP payload reception and processing, which involves tasks such as deobfuscation using one of the Ciphers and subsequent execution through the Packet Pipeline, should be consolidated within this library to avoid redundancy and ensure a more efficient network management approach.

4.4 Login Server

The Login Server is responsible for player/user authentication and the initial setup of their account. It handles tasks such as assigning a player's nickname and selecting their playable character (which can be expanded later in the game as the player unlocks more characters).

Additionally, the Login Server is responsible for issuing session keys, which serve two purposes:

1. To allow the player to join a Game Server without having to provide credentials again
2. To enable the client to retrieve the list of available Game Servers in case of a connection issue.

In Figure 12 and 13 the authentication procedure can be seen.

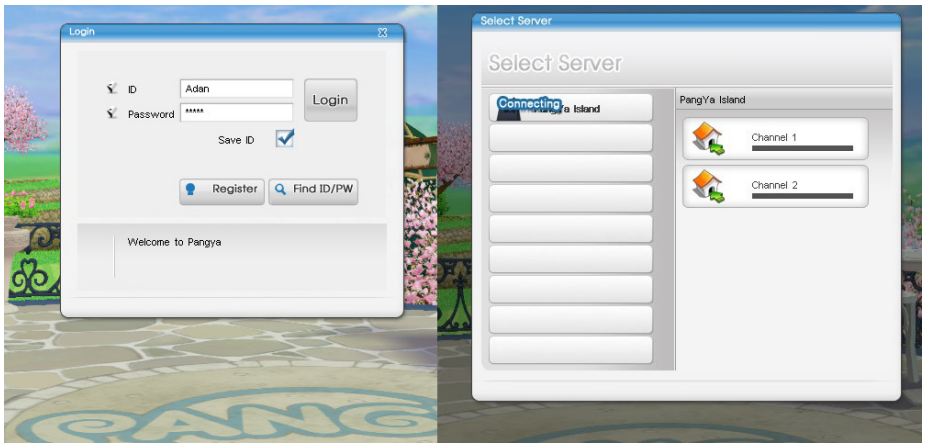

Figure 12: A screenshot showcasing the authentication procedure being performed in the Login Server emulator.

The Login Server was the first component to be implemented, as it is the first service the client interacts with and it is also the simplest one. This initial implementation allowed for the thorough testing of the Network Manager library's handling of polymorphic authentication result packets, which was a crucial step in the development process.

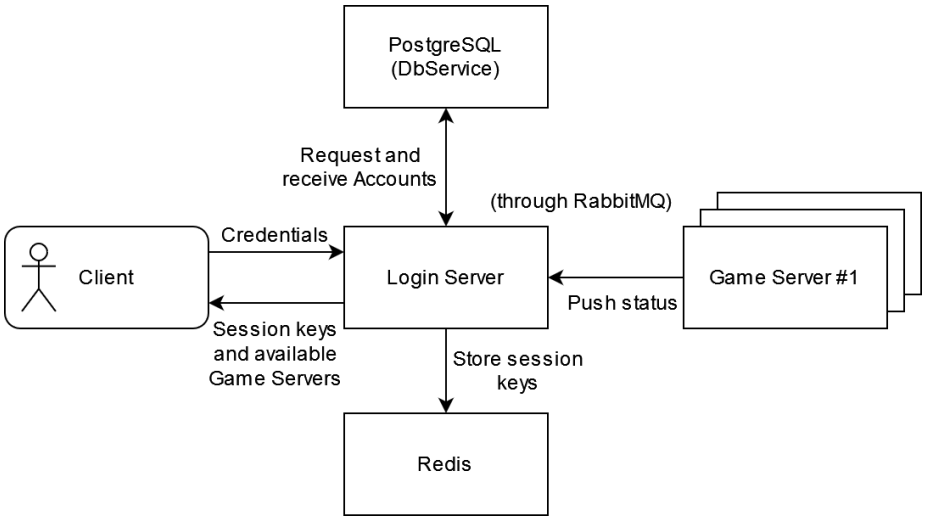

Figure 13: A diagram showing the authentication procedure being performed internally in the Login Server emulator.

4.4.1 Password Management

The separation of authentication functionality in the Login Server is a beneficial design choice, as it allows the Login Server to handle the computationally expensive password key derivation process without impacting the performance of the Game Servers and the player's overall gaming experience.

The password key derivation algorithm used in the storage of passwords is Argon2id, which is currently considered a state-of-the-art Key Derivation Function (KDF). However, the Pangya Protocol requires the client to send an MD5 hash of the original password, rather than the plaintext password. While this approach provides some protection against password sniffing (since the Pangya Protocol only obfuscates, not encrypts, the communication), it has drawbacks: [39, 40, 41]

1. MD5 is now considered a weak hashing algorithm, as it is relatively easy to compute and many password dictionaries are available for attacks.
2. The use of an MD5 hash reduces the possible entropy that the Argon2 KDF can obtain from the user's password. If the plaintext password was used and transferred through a secure channel (e.g., TLS), it would provide better security.

4.4.2 Session Keys

After sending the list of available Game Servers to the client, the Login Server waits for the client to select the Game Server they wish to connect to. Once

the client sends the appropriate obfuscated payload indicating their choice, the Login Server proceeds to store a Game Session Key in the distributed key-value store (in this case, Redis) with a short Time-To-Live (TTL).

The use of a short TTL for the stored Session Key serves several purposes:

1. **Client Connection Urgency**: The client is expected to connect to the selected Game Server as quickly as possible, typically within a second under normal circumstances. The short TTL ensures that the Session Key is only valid for a limited time, encouraging the client to connect without delay.
2. **Efficient Key-Value Store Management**: The short TTL prevents the Key-Value Store from accumulating unnecessary information. Any connection attempt that fails to occur within the specified time frame (e.g., one minute) can be safely assumed to have been abandoned, and the corresponding Session Key can be discarded.
3. **Security Enhancement**: The short TTL reduces the window of opportunity for potential attackers to guess or brute-force the Session Key. Even if the chances of such an attack are minimal, the reduced validity period further mitigates the risk.

To enhance the security of the Session Keys, the implementation uses GUIDs (e.g., `07f017d4-4db0-4f81-889c-5c5eb1785b8e`) instead of the original short and sequential hexadecimal numbers (e.g., `a12579e`). The use of GUIDs provides a much larger range of variability, making it significantly more difficult for an attacker to guess or brute-force a valid Session Key. [42]

This approach to Session Key management, with the combination of a short TTL and the use of GUIDs, helps to ensure the security and efficiency of the authentication and connection process between the client and the Game Server.

4.4.3 Heartbeat System

The Login Server implements a Heartbeat system to maintain a list of available Game Servers. It subscribes to an MQTT topic and waits for "GameServiceStatus" messages published by the Game Servers. These messages contain information about the Game Servers, such as their name, ID, IP address, port, player count, player capacity, icon, boosts, and server type.

The Login Server builds a list of available Game Servers based on the received messages, but each entry in the list has a time-to-live. This ensures that the list only contains information about healthy, available Game Servers, as entries will be removed if no new "GameServiceStatus" message is received within the specified time frame.

4.5 Game Server

Disclaimer: *Only the most basic functionalities were implemented in this Game Server due to time constrain.*

4.5.1 Packet Structure Difficulties

Reverse-engineering the packet structures for the Game Server was a significant challenge, as I did not have access to the original packet dumps from the US region or complete open-source US emulators. This meant that determining the correct packet structures became a trial-and-error process, relying heavily on the efforts of the community and the structures provided by the JP emulator from Acrisio.

One of the most complex and difficult-to-implement packets was the one that confirms the successful authentication of a client. The complexity arose from the size of the packet, which depended on the various types of state information the client needed to build the proper general state for their player. Some of these state elements did not appear to be used, but I included them just in case. Additionally, the lack of original packet dumps made it challenging to ensure complete accuracy, as the US packet structure I was trying to replicate was not compatible with other available dumps (such as the TH region).

Another structure that posed significant challenges during the development of the Game Server emulator was related to the implementation of the practice mode plays. I made a mistake in the structure, swapping a byte for an integer, which caused the packet to be 3 bytes larger. This issue went undetected for 6 months, as the crash only occurred during the play's entry animation, with a relatively high chance of 49 in 50. Thinking the issue was caused by "corruption" from a malformed packet, I embarked on an extensive search, manually re-checking and comparing my packet structures to multiple open-source emulators and dumps.

The root cause of the issue was eventually traced back to the packet that sends the detailed state information of all the players in the play before the start. I had confused the specific sizes related to the practice mode with those of other types of plays, leading to the incorrect structure. Once this issue was resolved, the rest of the development process became more straightforward.

These experiences highlight the challenges of reverse-engineering network protocols, especially when dealing with limited resources. The meticulous attention to detail and the ability to systematically investigate and isolate issues were crucial in overcoming these obstacles and successfully implementing the Game Server emulator.

4.5.2 IFF Explotation

The IFF files are crucial for the Game Server to provide appropriate state responses to the client. Without access to the information contained in these files, the server would be unable to properly identify and describe the various game elements, such as items, characters, and maps, that the client expects to interact with.

To leverage the data in the IFF files, we utilize a Shared Library that allows us to extract only the relevant information, leaving out unnecessary data like

models, textures, and effects. This extracted data is then stored in a Singleton, providing centralized and efficient access to all the information the server components may need.

By maintaining this in-memory representation of the IFF data, we avoid the performance overhead of repeatedly reading from the files. Additionally, the Singleton approach ensures that the data is not duplicated, as it is passed by reference to the various server components that require it.

This centralized IFF data is crucial for implementing the correct behaviors and attributes of the game's items, characters, and other elements. For example, a special Comet item may have the effect of granting an additional 15 Pangs (the game's currency) per shot. To accurately emulate this behavior, the server must be able to look up the item's ID in the IFF data and determine the specific effects it should apply.

Similarly, when the client requests information about an item, the server must consult the IFF data to provide the correct attributes, such as the stats (accuracy, power, etc.) items provide. This deep integration between the server's business logic and the data stored in the IFF files is essential for achieving a faithful emulation of the original behaviour.

By leveraging the Shared Library and the Singleton pattern to manage the IFF data, the Game Server can efficiently access and utilize the necessary information to properly emulate the game's mechanics and respond to the client's requests.

4.5.3 Inventory Management

The management of player inventories is closely tied to the exploitation of IFF files, as these files provide the necessary representations of the various items that can be acquired and used by players.

In the early stages of development, we have opted for a more flexible approach to player inventory management, as it allows us to rapidly iterate and adapt to new issues that arise during the emulation process. This approach is a hybrid between a traditional SQL-based storage and a more flexible, NoSQL-like solution.

Specifically, we are storing a portion of the player's inventory data as rows in a SQL table, while the remaining data is saved as a JSON document in a specialized PostgreSQL column type. This hybrid approach provides the following benefits:

- **Rapid Iteration**: By using a DTO (Data Transfer Object) to represent the player's inventory, we can quickly update and modify the inventory management logic without being constrained by the rigidity of a pure SQL-based solution. This allows us to adapt to new requirements and issues more efficiently during the early stages of development.

- **Flexibility for future expansion**: As the emulator matures and we introduce more complex functionalities, such as player-to-player transactions (e.g., trades, shops), the hybrid approach provides a foundation that can be more easily transitioned to a fully SQL-based solution. This will allow us to enforce data integrity and perform efficient queries on the player inventory data.

Our packet processing doesn't directly work with the `PlayerDTO`, instead we provide a decoupled interface from which the packet processing interacts. The reason behind this is to have an intermediate state that is more 'packet processing friendly' by already including part of the business logic that is shared by most parts. It also opens up an avenue for easily changing the persistence without interfering with the processing part.

The rationale behind this hybrid approach is to balance the need for rapid development and adaptation in the early stages with the long-term goal of maintaining data integrity and supporting more advanced features. In Figure 14 a representation of the usage can be seen.

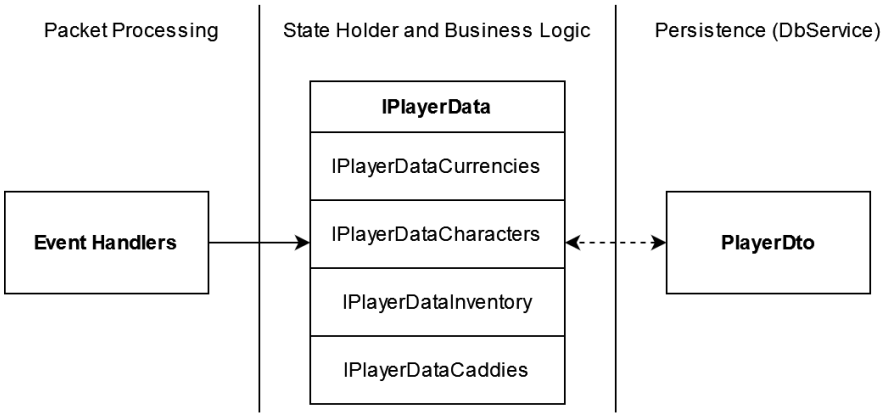

Figure 14: A diagram showing the flow of our inventory management.

Once the core functionality of the emulator is stable and the majority of the required features are implemented, we plan to transition to a fully SQL-based inventory management system.

This will provide better data integrity for corruption (which the current hybrid system can cause), a better support for complex queries, and a more robust foundation for future expansions and integrations with a backend system.

By leveraging the flexibility of the hybrid approach and the power of SQL-based storage, we can ensure that the inventory management system evolves alongside the emulator's development, ultimately providing a reliable and scalable solution for managing player inventories.

4.5.4 Game System

The Game Server's management of game state is heavily influenced by the client's expectations. The official server was organized into Game Servers, each containing multiple channels with varying player capacities and level-based restrictions (e.g., newbie channels and more experienced channels).

Channels isolate players, even if they are on the same Game Server. Attempting to chat or play with someone in another channel would not be possible.

Within a channel, players can access the "Multiplay" list, which contains rooms for multiplayer gameplay or social interaction (e.g., chatting, item trading). In Figure 15 a representation can be seen.

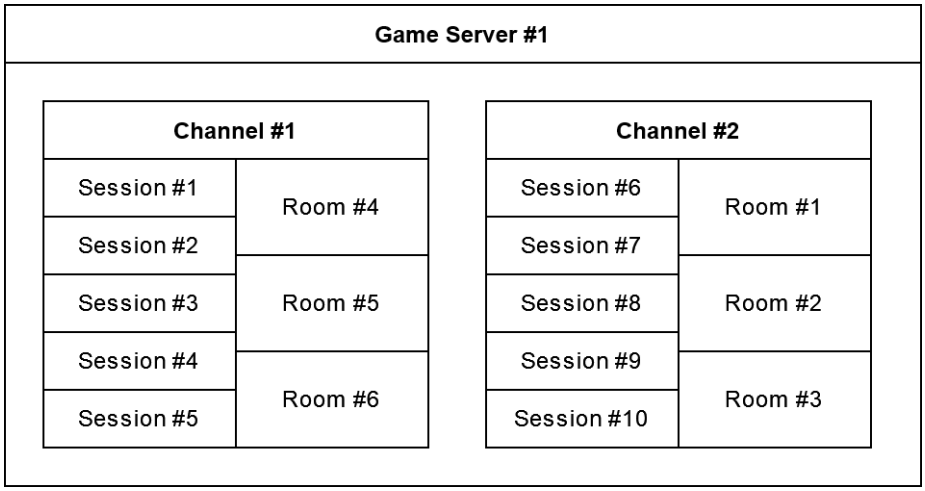

Figure 15: A diagram showing the isolation the different parts provide to a Game Server.

To manage this complex system, I implemented a set of "Managers" - Singletons that serve as the source of truth for different entities. The Managers I defined include Player Sessions, Channels, and Rooms.

This approach allows for centralized access to player information and state, without the need for individual player sessions to track their own location. Additionally, the Singleton pattern facilitates concurrency management, as the asynchronous packet processing from multiple clients can be handled safely without risking data corruption.

The distinction between gameplay rooms and social "lounges" led to the need for a separate Game System component. Rooms serve as a container for these Game Systems, which can be added to provide specific gameplay functionality. In Figure 16 a representation can be seen.

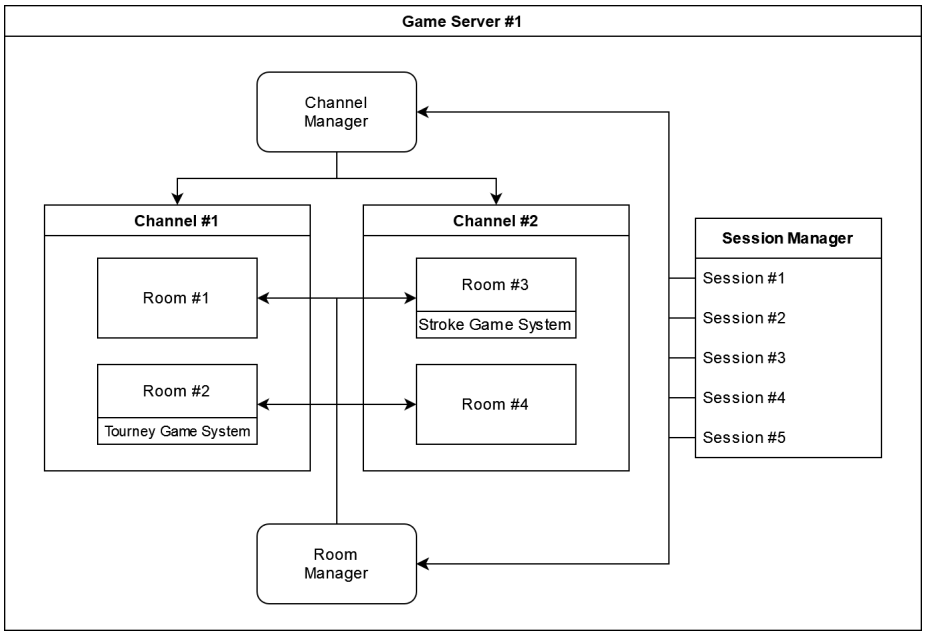

Figure 16: A diagram showing the different components that form the implemented system.

I have implemented two Game System types so far: Stroke and Practice Mode. Stroke play tracks the total score for an entire game, with the player achieving the lowest score declared the winner. Practice Mode, on the other hand, is a single-player test environment.

In the future, I plan to implement additional Game System types, such as:

- Match: Teams of two players compete for a win on each hole, with the team winning the most holes declared the victor.
- Tourney: Up to 30 players compete in a short 9- or 18-hole game, with the player achieving the lowest score winning.
- Guild Battle: Members of two guilds compete, with the guild's total points determining the winner.
- Approach Battle: Players aim to get their Comet closest to the hole without going in, with the closest player winning the entrance fee.
- Pang Battle: Players compete on a hole-by-hole basis, with the best player on each hole winning the money at stake (a draw results in a push to the next hole).

This modular Game System design allows for the easy addition of new gameplay modes in the future, while maintaining a clear separation of concerns and centralized state management.

4.6 Overview

We have successfully established a functional setup to emulate the essential parts of the original Pangya infrastructure, including the CDN and the services needed for client synchronization with other players (Login Server and Game Server, while excluding the Message Server due to time constraints).

The project prioritizes maintainability, achieved by modularizing the code into shared libraries that are utilized by various components. The following dependency graph (Figure 17) illustrates this structure. As depicted, most libraries are employed by multiple components, highlighting their reusability and the cohesive architecture of our project.

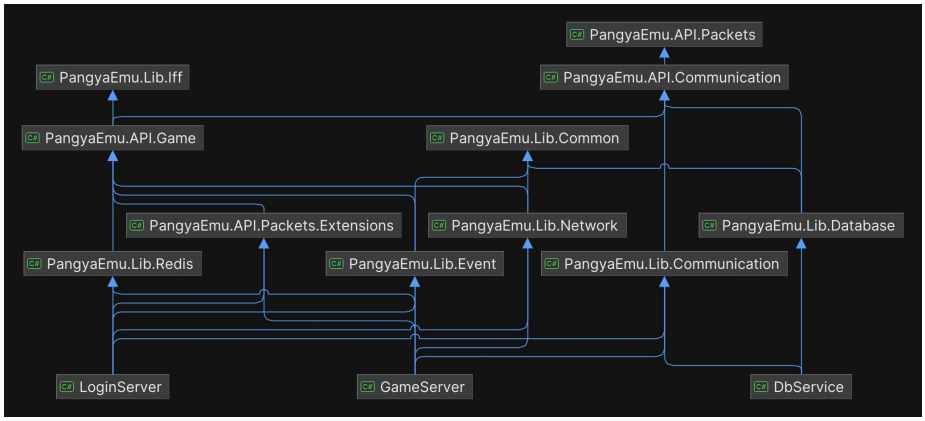

Figure 17: Dependency graph of the different components and libraries in the project's source code.

In addition to the core infrastructure, we developed a launcher that, although it provides only minimal functionalities, represents a significant step towards a public release playable by the Pangya community.

Overall, I am pleased with our accomplishments. Despite the numerous challenges encountered, we have managed to bring the project to a satisfactory state, ensuring both functionality and maintainability.

5 Evaluation

In this section, we will evaluate the extent to which our project provides game functionalities and how well it performs in terms of resource consumption.

5.1 Functionalities

Given the scope of this thesis, we have implemented a set of core functionalities prioritized for their importance to players:

- **Basic Inventory Management**: Players can equip most types of items to their characters. Our current implementation excludes niche features such as clubset upgrades, mascots, passive buffs, custom-designed clothes, and decorations.
- **Basic Gameplay**: Players can engage in two game modes: Stroke and Practice. This basic gameplay implementation includes essential features such as wind, weather, course track generation, and different shot types. Additionally, players receive experience points and in-game currency rewards upon completion of each game. Consumables, which provide temporary buffs, are not yet implemented.

While these features form a substantial part of the game's functionality, several critical elements are still missing:

- Tutorial Courses
- In-game Shop (for spending in-game currency)
- Functional Lounges (special rooms for social interactions, trading, and setting up shops)
- Achievements and Trophies
- Mail System
- Friends System and Inter-Game Server Messaging
- And more

To ensure the implemented functionalities work correctly, we have tested the project with twelve players in total by giving them access to all the available objects and allowing them to play Stroke games together.

This testing process, which has been repeated multiple times throughout the development, has helped us identify and correct multiple issues, improving the stability and reliability of the core game systems.

In Figure 18 and Figure 19 I leave some images from this tests.

Figure 18: An image from one of the test sessions. She has just holed in!

Figure 19: Another image from one of the test sessions. Will he win?

5.2 Performance Profiling

Using the latest functional version of the project, we conducted a test with three players, focusing on the most demanding part of the game: playing in rooms.

The test session lasted for 82 minutes and 44 seconds, during which we profiled the Game Server's performance.

The profiler results revealed no memory leaks or significant heap allocation issues, which is a relief. However, it identified a major inefficiency: we log all packets that get serialized and deserialized when running a Debug version of the server. It seems that logging is one of the most CPU-intensive task in our build (a thing we never thought about).

Logging accounted for approximately 80% (5.5s/7s) of the CPU time during packet processing, while the remaining 20% (1.5s/7s) was due to reflection and heap allocation for intermediary representations. In Figure 20 we can see the weight logging has in serialization.

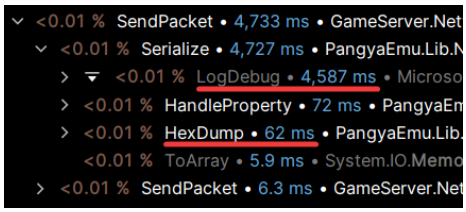

Figure 20: A graph illustrating the CPU usage distribution in the serialization of packets during the test session.

Despite the logging overhead, the total execution time impact was minimal, not exceeding 0.01% on average, with a maximum observed burst of 0.6%. We can safely assume that without the debug logging, performance would improve considerably; even if it is already good enough.

Taking into account Reflection, the impact is not as severe as with logging, but it still leaves room for improvement. And as discussed in previous sections, switching to Source Generators could reduce the overhead caused by it significantly.

Other tasks such as obfuscation, deobfuscation, and business logic execution had negligible impact on performance.

The tests were conducted on a Debug build with an AMD Ryzen 5 3600 processor. Running a Release build (which also excludes debug logging) would yield even better performance results.

In conclusion, our implementation is highly performant with no significant stability issues. The identified inefficiencies are solvable and do not impact overall system stability.

6 Conclusions

This thesis has been a significant learning experience for me. While I did not anticipate the depth of knowledge and skills I would gain, the process of reverse-engineering and rebuilding the Pangya game infrastructure has been both challenging and rewarding.

One of the key lessons I've taken away is the importance of persistence and a willingness to explore the unknown. When faced with the complexities of the client's behavior, I had to venture into the realm of reverse engineering, even when my understanding of the underlying concepts was limited. This trial-and-error approach not only expanded my knowledge of binary data structures but also instilled in me a greater appreciation for the art of deciphering black-box systems.

Furthermore, this project has pushed me to expand my technological skillset. From learning new programming languages like Rust and TypeScript to mastering the intricacies of building proper software architecture. I have broadened my capabilities in ways I had not previously considered. This diverse exposure has highlighted the value of adaptability in the ever-evolving landscape of software development.

Perhaps most importantly, this experience has reinforced the notion that even the most daunting challenges can be overcome through determination and a commitment to continuous learning. When faced with seemingly insurmountable obstacles, I learned to persevere, to dig deeper, and to trust in my own problem-solving abilities. This has instilled in me a greater confidence in my capacity to tackle complex problems.

In conclusion, this thesis project has been a valuable learning experience, one that has not only expanded my technical expertise but also shaped my personal growth. The lessons I've learned – from the importance of reverse engineering to the value of adaptability and perseverance – will undoubtedly continue to guide me in my future endeavors, both academic and professional.

I hope that this document can help to centralize the knowledge that has been scattered all around the internet into one single resource, assisting the community of reverse-engineers in their efforts to preserve the beloved Pangya game.

References

[1] *English Wikipedia — Ntreev Soft.* URL: https://en.wikipedia.org/wiki/Ntreev_Soft. Accessed on March 29th, 2024.

[2] *English Wikipedia — White Day: A Labyrinth Named School.* URL: https://en.wikipedia.org/wiki/White_Day%3A_A_Labyrinth_Named_School. Accessed on March 29th, 2024.

[3] Korean Wikipedia — Pangya. URL: https://ko.wikipedia.org/wiki/%ED%8C%A1%EC%95%BC. Accessed on March 23th, 2024.

[4] Gamepot Inc. — Ntreev Soft Co. Ltd — HanbitSoft Inc. *The Korean popular online golf game, "Scat Golf Pangya", has started its advanced beta test service today!* June 2004. URL: https://web.archive.org/web/20221110093329/http://www.gamepot.co.jp/pdf/press_release/040618_pangya.pdf.

[5] Gamepot Inc. — Ntreev Soft Co. Ltd — HanbitSoft Inc. *"Skat Golf Pangya" — Official Service Starts Today (Free for Basic Play)!* Nov. 2004. URL: https://web.archive.org/web/20221110092021/http://www.gamepot.co.jp/pdf/press_release/041111_pangya.pdf.

[6] *History of Pangya Servers.* URL: https://pangya.community/t/history-of-pangya-servers/25. Accessed on May 19th, 2024.

[7] *English Wikipedia — Pangya - Other versions.* URL: https://en.wikipedia.org/wiki/PangYa#Other_versions. Accessed on May 19th, 2024.

[8] *Service Closure Announcement - PANGYA FORUM - atomicpang12's comment.* URL: https://web.archive.org/web/20161108001614/http://forum.gamerage.com/pangya/forums/t/42958.aspx?PageIndex=3.

[9] *"Skat Golf Pangya" — Notice of Service Termination.* URL: https://web.archive.org/web/20180318014748/http://www.pangya.jp/board_notice_view.aspx?seq=15093&cat=1.

[10] Lee Ju-hwan. *NCC Cleans Up Subsidiary Entries... Choosing and Focusing.* Jan. 2024. URL: https://web.archive.org/web/20240329180450/https://www.gamejob.co.kr/community/news/detail?mode=V&comm_stat=11&Sub_Comm_Stat=0&idx=101557.

[11] Ini3 Digital. *Announcement! Termination of Pangya game service.* Mar. 2024. URL: https://web.archive.org/web/20240329174525/https://pangya.mygame.in.th/movement/notice_view?id=MjAyNDAzMjcxNzU3NTk.

[12] *retreev — Github Organization.* URL: https://github.com/retreev. Accessed on May 20th, 2024.

[13] *retreev/io_scene_mpet — Blender import script for PangYa models (.mpet).* URL: https://github.com/retreev/io_scene_mpet. Accessed on May 20th, 2024.

[14] *retreev/PangLib — Series of tools to interact with Pangya PC MMO game files.* URL: https://github.com/retreev/PangLib. Accessed on May 20th, 2024.

[15] *retreev/documentation — Pangya PC MMO file formats and general structure.* URL: https://github.com/retreev/Documentation. Accessed on May 20th, 2024.

[16] *retreev/pangcrypt — Library implementing PangYa's transport encryption.* URL: https://github.com/retreev/PangCrypt. Accessed on May 20th, 2024.

[17] *GameRaze Brazil — Servidor-GB-by-LuisMK.* URL: https://github.com/eantoniobr/Servidor-GB-by-LuisMK. Accessed on May 20th, 2024.

[18] *alter-pangya — A PangYa GB.852 server emulator.* URL: https://github.com/hex-agon/alter-pangya. Accessed on May 20th, 2024.

[19] *Shadosoft TM — Pangya search.* URL: https://hsreina.shadosoft-tm.com/#:~:text=pangya. Accessed on May 20th, 2024.

[20] *KuramaxD — HIO-Emulator.* URL: https://github.com/KuramaxD/HIO-Emulator. Accessed on May 20th, 2024.

[21] *UGPangya — Unogames Pangya Server Development.* URL: https://github.com/luismk/oijdasoidjaoisjda. Accessed on May 20th, 2024.

[22] *Acrisio-Filho — Superss-Dev.* URL: https://github.com/Acrisio-Filho/SuperSS-Dev. Accessed on May 20th, 2024.

[23] *XTEA - Wikipedia.* URL: https://en.wikipedia.org/wiki/XTEA. Accessed on May 30th, 2024.

[24] *Pangya! Documentation - Updatelist.* URL: https://docs.pangya.golf/pc/file-formats/updatelist. Accessed on May 30th, 2024.

[25] *pycrc32/crc.go — pangbox/pangfiles - John Chadwick.* URL: https://github.com/pangbox/pangfiles/blob/master/hash/pycrc32/crc.go. Accessed on May 30th, 2024.

[26] *pangbox/pangfiles - John Chadwick.* URL: https://github.com/pangbox/pangfiles. Accessed on May 30th, 2024.

[27] *SuperSS-Dev/Tools — Acrisio Filho.* URL: https://github.com/Acrisio-Filho/SuperSS-Dev/tree/master/Tools. Accessed on May 30th, 2024.

[28] *PangLib.Pak — pixeldesu - Andreas Nedbal.* URL: https://github.com/retreev/PangLib/tree/master/PangLib.PAK. Accessed on May 30th, 2024.

[29] *QuickPatch - Pangya Wiki.* URL: https://pangya.wiki/wiki/QuickPatch. Accessed on May 31th, 2024.

[30] *Unogames BR - GameGuard.* URL: https://web.archive.org/web/20190908144159/http://unogames.com.br/GameGuard.

[31] *GGSrvLib26-1 — Acrisio Filho*. URL: https://github.com/Acrisio-Filho/SuperSS-Dev/tree/master/Server%20Lib/GGSrvLib26-1/GGSrvLib26-1. Accessed on June 1st, 2024.

[32] *pangbox/rugburn: A small library that allows you to run PangYa without GameGuard*. URL: https://github.com/pangbox/rugburn. Accessed on May 31th, 2024.

[33] *Wireshark dissector for PangYa — John Chadwick*. URL: https://github.com/pangbox/wireshark-dissector. Accessed on June 2nd, 2024.

[34] *PangPacketSniffer — fumitti*. URL: https://github.com/fumitti/PangPacketSniffer. Accessed on June 2nd, 2024.

[35] *pangya-sniffer — Adanlink*. URL: https://github.com/Adanlink/pangya-sniffer. Accessed on June 2nd, 2024.

[36] *Enabling Ranking — Acrisio Filho's contribution*. URL: https://github.com/pangbox/rugburn/blob/master/src/hooks/projectg/us852/ranking.c. Accessed on June 2nd, 2024.

[37] *PangLib.IFF, library to handle and parse (.iff) files — pixeldesu*. URL: https://github.com/retreev/PangLib/tree/master/PangLib.IFF. Accessed on June 3rd, 2024.

[38] *IFF - Pangya! Documentation — pixeldesu*. URL: https://docs.pangya.golf/pc/file-formats/iff. Accessed on June 3rd, 2024.

[39] *MD5 - Security — Wikipedia*. URL: https://en.wikipedia.org/wiki/MD5#Security. Accessed on June 5th, 2024.

[40] Ben Tasker. *Why You Shouldn't be using SHA1 or MD5 to Store Passwords*. URL: https://www.bentasker.co.uk/posts/blog/security/201-why-you-should-be-asking-how-your-passwords-are-stored.html. Accessed on June 5th, 2024.

[41] *Password Storage - OWASP Cheat Sheet Series*. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#password-hashing-algorithms. Accessed on June 5th, 2024.

[42] *LoginService Server Session Key*. URL: https://packets.pangdox.com/packets/loginservice/server/0003. Accessed on June 5th, 2024.