

HSC Major Design Project 2020

Design & Technology

“Project Goodwitch: Full Scale C# Anti-Cheat”

Student ID: [REDUCTED], 2020

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Abstract

The motivation for this project stemmed from the observation that majority of modern anti-cheats are developed under the native language(C++) and due to many issues surrounding portability of native environment to .NET/C#, Goodwitch was born. I also wanted to test the limitations of my knowledge and skills in C# and also was interested on how far I could achieve with this project

Contents

Chapter 1.	
Introduction	5
Chapter 2.	
Introduction To And Research On Cheats	6
2.1 Operations of A Cheat Software	7
2.2 Tools Used In A Cheat Software	8
2.3 Publicity of A Cheat Software	10
2.4 Techniques Used In A Cheat Software	11
Chapter 3.	
Introduction To Goodwitch	12
3.1 What Is Goodwitch	12
3.2 Why C# Though?	12
3.3 Overall Design of Goodwitch	13
3.4 Breakdown of Goodwitch Modules	15
Chapter 4.	
Development of Goodwitch	
4.0 Initial Coding	16
4.1 Implementing It To A Real World Example - Game	23
4.2 Feature Demonstrations	24
Chapter 5.	
Project Schema	
5.1 Criteria to evaluate success	24
5.2 Time & Action Plan	26
5.3 Area of Investigation	28
5.4 Financial Plan	29
Chapter 6.	
Evaluation	
6.1 Application of Evaluation & Analysis and Evaluation of Functional and Aesthetic Aspects of Design	29
Analysis of Preventing Third-Party Injection	29
Analysis of Preventing Tampering of The Game (Anti-Dumping, Anti-Debugging, Anti-Decompiling) & Preventing Execution of Third-Party Applications	30
Analysis of Detecting Known/Detected Cheats	31
References	32

Chapter 1.

Introduction

The gaming industry has grown significantly over the past decade. Games that are now out in the market provide an online game mode to players for playing against other players(also known as PVP) which wasn't the case 5 or 7 years ago where the single player games were the only games in existence. This provides a great opportunity for cheat developers since the competitive gaming scene is the hot place for gamers and even professional players around the world. With an increased amount of players in the competitive scene where the "money is at", it's now an attractive target for the people who are trying to exploit those games with cheats and gain illegal profits. Cheat developers gain estimatly \$10 million per year. For instance, CS:GO(Counter Strike: Global Offensive) consists of a large volume of cheaters in the game. Cheat is a software which modifies the game environment in a way not intended by the game developers to give players an unfair access and advantage against other players.

This unfair advantage can be provided by such as:

- Displaying information which normally is hidden from players.
- Modifying the game's memory to allow players to perform actions that are over the level of human abilities.
- Automating and simulating game client behaviours, mouse movements and key presses.

To target and fight against the rising of the cheats, anti-cheat software is crucially needed. There are many anti-cheat software out in the market, targeting small to large scaled games and even the development of an in-house anti-cheat software are in action to mitigate against the cheats in their own game. However, EAC(EasyAntiCheat) and BE(BattlEye) the well-known anti-cheat software in the industry does not support C# based games such as Unity developed games and so forth, this is where Goodwitch comes in to provide full scale C# anti-cheat solution for C# based games and even for native developed games.

Chapter 2.

Introduction To And Research On Cheats

Cheat software uses common methods that a typical malware uses to perform malicious actions. However, the main difference between a cheat and a malware is that the cheat executes with the intention of a user. Below are the common methods that a cheat software majorly shares with a malware:

- DLL Injection(also known as Byte Injection)
- Modification of the process's code, memory and network protocols
- Modification of operating system's internal calls to hide the existence of the cheat

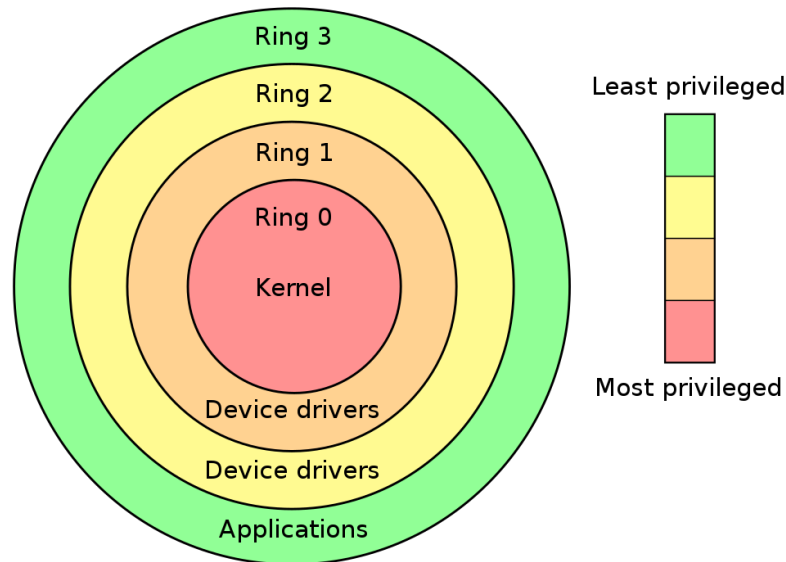
Just like every windows applications, cheat software also gets allocated its own memory space by the OS's kernel formerly named as the Virtual Address Space. Virtual Address Space is a memory space allocated by the OS's kernel in order to separate the working space of memory addresses from the physical storage, also known as RAM(Random Access Memory). When a virtual memory space is allocated to a process, it is then registered as a private memory section or a region to protect from other processes accessing the process's address space. This concept is usually brute-forced and exploited by cheat softwares in order to access a game's memory and information of the gameplay.

There are two features to identify the type of a cheat, **internal** and **external**.

Internal cheats are a cheat software that are majorly developed and used and by referencing the feature "internal", where internal cheats works with-in the game's memory space by injecting a DLL(Dynamic Link Library) into the game's memory space. It is majorly developed due to the factor that it directly accesses game's memory in a live manner without any performance redundancy. On the other hand, there are external cheats. External cheats are a lot less developed and used however its demands are high due to the fact that it runs "externally" from the game. The way how external cheat works is, it runs as a separate process from the game and calls Windows API function *ReadProcessMemory* to read game's process memory without any manipulations to the game's memory space. This way, it ensures the safety and the level of detection of the cheat as low as possible.

2.1 Operations of A Cheat Software

To circumvent anti cheat softwares, cheat software runs mainly on two virtual lands, user-land and kernel-land.



[Source 1.1]

Source 1.1 references different privilege ring levels often so called protection rings. Protection Rings are mechanisms to protect data and functionality from faults and malicious behaviors, and the above diagram indicates the different hierarchical privilege levels of a computer architecture. In the above diagram, there are varieties of protection ring levels. Not all levels are actually used or applications running in those levels and theoretically, there are a total of 4 ring levels. When viewed practically and technically, there are only 2 ring levels which are the user-land and the kernel-land. Device drivers such as audio driver, keyboard driver, mouse driver and USBs are all unitively in one kernel land. However, ring1 and ring2 are just an abstraction layer to distinguish and to remove some complications towards understanding protection levels. Although in a technical sense, they all reside in and share the same ring 0 nature. In order to clear some abstractness, the user-land and kernel-land will be referenced as ring3 and ring0 respectively from here and onwards.

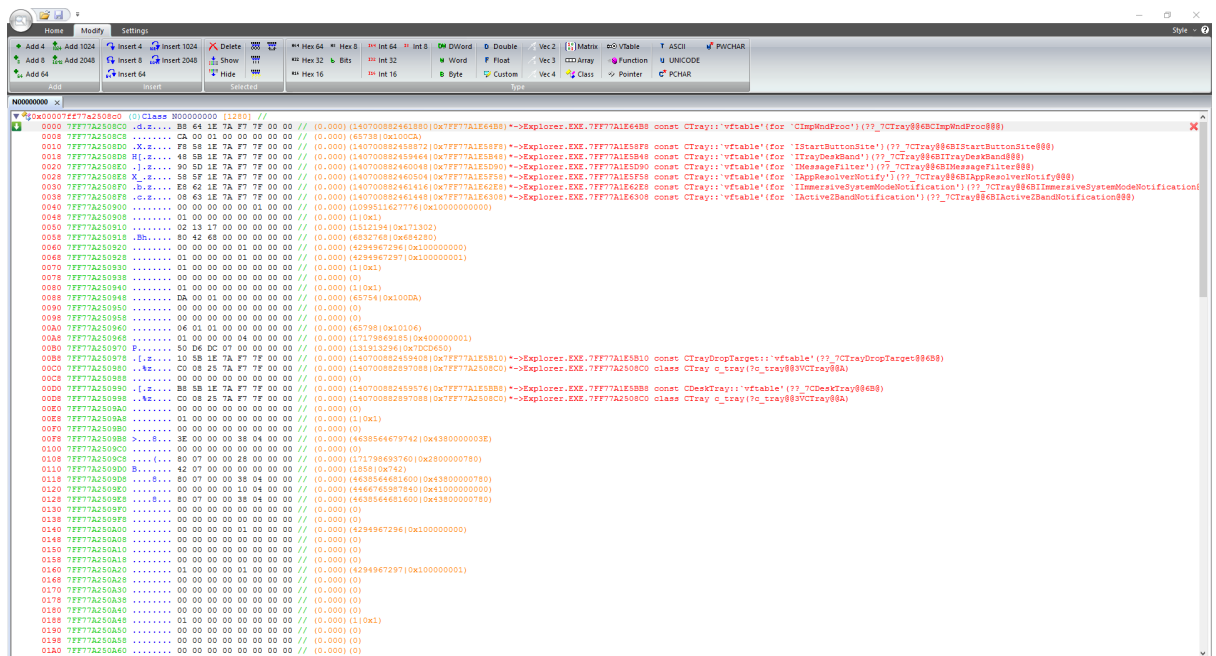
As stated earlier, cheat software runs mainly on two virtual lands, user-land, which is the applications layer formerly called ring3, and kernel-land, which is the kernel layer formerly called ring0. Ring3 cheats run on the ring3 layer, which has access to applications that are running on the same level of the ring protection limitedly. It is widely developed in and run on due to the ease of setup and application portability of operation. However, for ring3 cheats, due to the nature of its level, it is subjected to a more wider detection vector of the application by the anti-cheat softwares. On the other hand, ring0 cheats run on a ring0 layer where it is hosted in a ring0 driver, which has capabilities of accessing both applications that

are running on ring3 and on ring0. Typically, ring0 cheats set up a ring0 kernel driver in order to circumvent ring3 anti cheat softwares and to battle against ring0 anticheat drivers. While running a cheat on ring0 layer can circumvent ring3 anticheats, it is subjectively difficult to develop and set up due to the amount of skills and knowledge of hardware level is required in order to develop a ring0 driver. It is also again subjectively difficult to distribute the cheat and the driver since ring0 drivers are bound to the specific type of CPU instructions set and its architecture.

2.2 Tools Used In A Cheat Software

- **ReClass / ReClassEx**

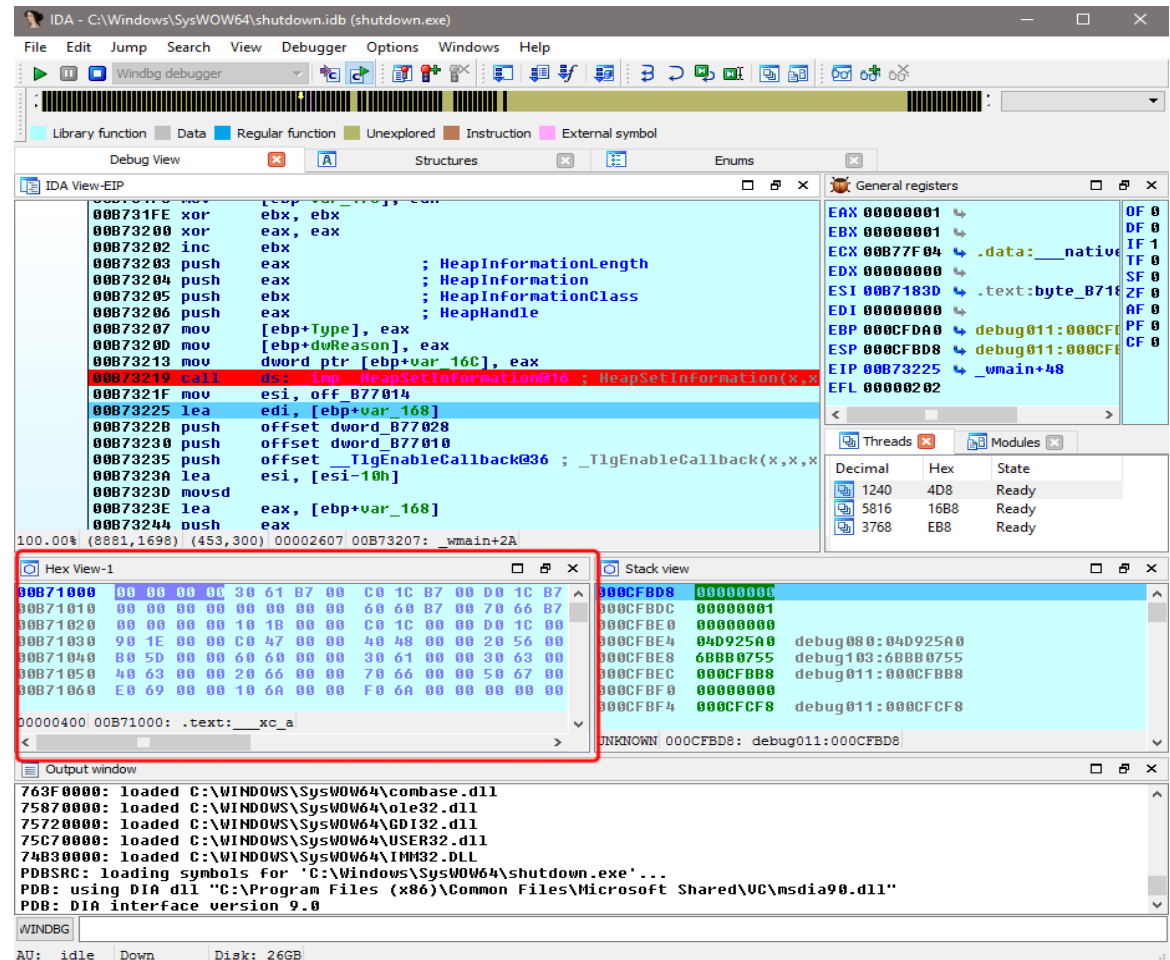
ReClass or hence ReClassEx the improvised version of ReClass is an open sourced live memory analysis tool to view data structures in a process's memory. In the case of a cheat, it is mainly used to view and analyse the data structures of a player and the game itself.



[Source 1.2]

- IDA

IDA or formerly known as Interactive Disassembler is a closed sourced free and paid commercial static and dynamic binary and memory analysis tool. In the case of a cheat, it is mainly used to view a graphical representation of the raw binary and disassemble lower level codes of and in the game.



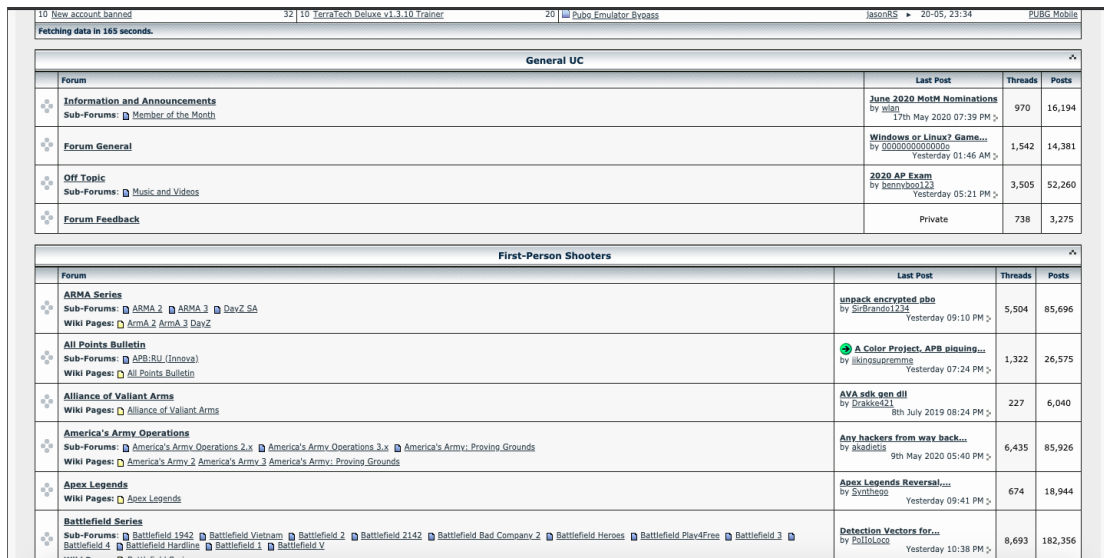
[Source 1.3]

2.3 Publicity of A Cheat Software

- **Public Cheats**

Public cheats are usually referred to open sourced cheat softwares and as a trade off, they are subjectively bound to being in a larger detection vector.

Example: www.unknowncheats.me



General UC			
Forum	Last Post	Threads	Posts
Information and Announcements Sub-Forums: Member of the Month	June 2020 MoM Nominations by m881 17th May 2020 07:39 PM >	970	16,194
Forum General	Windows or Linux? Game... by 0000000000000000 Yesterday 01:46 AM >	1,542	14,381
Off Topic Sub-Forums: Music and Videos	2020 AP Exam by beatmop143 Yesterday 05:21 PM >	3,505	52,260
Forum Feedback	Private	738	3,275

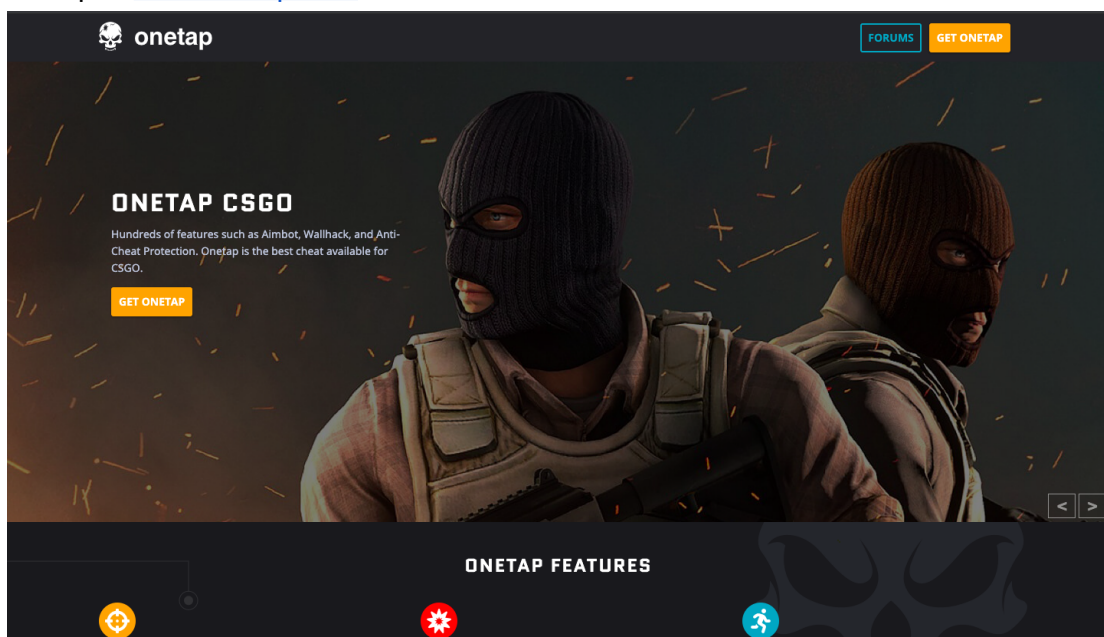
First-Person Shooters			
Forum	Last Post	Threads	Posts
ARMA Series Sub-Forums: ARMA 2 ARMA 3 DavZ SA Wiki Pages: Arma 2 Arma 3 DavZ	unpack encrypted sbo by SirBrando1234 Yesterday 09:10 PM >	5,504	85,696
All Points Bulletin Sub-Forums: APB:RL (Innova) Wiki Pages: All Points Bulletin	A Color Project, APB pluin... by @Innoscentia Yesterday 07:24 PM >	1,322	26,575
Alliance of Valiant Arms Wiki Pages: Alliance of Valiant Arms	AVA sdk gen dll by Drakke421 8th July 2019 08:24 PM >	227	6,040
America's Army Operations Sub-Forums: America's Army Operations 2.x America's Army Operations 3.x America's Army: Proving Grounds Wiki Pages: America's Army 2 America's Army 3 America's Army: Proving Grounds	Any hackers from waw back... by asadsgs 9th May 2020 05:40 PM >	6,435	85,926
Apex Legends Wiki Pages: Apex Legends	Apex Legends Reveal... by Skottmop Yesterday 09:41 PM >	674	18,944
Battlefield Series Sub-Forums: Battlefield 1942 Battlefield Vietnam Battlefield 2 Battlefield 2142 Battlefield Bad Company 2 Battlefield Heroes Battlefield Play4Free Battlefield 3 Battlefield 4 Battlefield Hardline Battlefield 1 Battlefield V Wiki Pages: Battlefield Series	Detection Vectors for... by @Silence Yesterday 10:38 PM >	8,693	182,356

[Source 1.4]

- **Private Cheats**

Private cheats are usually referred to as closed source and Pay-2-Cheat(P2C) softwares. It is bound to limited hands-on products by the people and is sold at a high price tag.

Example: www.onetap.com



[Source 1.5]

2.4 Techniques Used In A Cheat Software

- **LoadLibraryA - WINAPI(Kernel32.DLL)**
(<https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrarya>)
WINAPI's LoadLibraryA function is a user-mode WINAPI(Windows API) call to load a file/module or often so called DLL(Dynamic Link Library) into a windows process. It is widely and commonly used in internal cheats to load cheat DLL into the game's process.
- **WriteProcessMemory - WINAPI(Kernel32.DLL)**
(<https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-writeprocessmemory>)
WINAPI's WriteProcessMemory(WPM) function is a user-mode WINAPI call to write a byte or bytes of memory into a windows process. It is also widely and commonly used in external cheats to manipulate player's properties. In practice, it can be used to modify a player's view angle for aimbotting purposes.
- **VirtualProtectEx - WINAPI(Kernel32.DLL)**
(<https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualprotectex>)
WINAPI's VirtualProtectEx(VPEX) function is a user-mode WINAPI call to change protection of a memory region on a windows process. It is used with WriteProcessMemory as to write memory into a process and if the memory region is protected, it will change from PAGE_NOACCESS protection state to PAGE_EXECUTE_READWRITE state.
- **OpenProcess - WINAPI(Kernel32.DLL)**
(<https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-openprocess>)
WINAPI's OpenProcess function is a user-mode WINAPI call to open a new handle object to a windows process. It is also used with WriteProcessMemory as it needs to open a new handle object if the current handle rejects memory read/write permission.

Chapter 3.

Introduction To Goodwitch

3.1 What Is Goodwitch

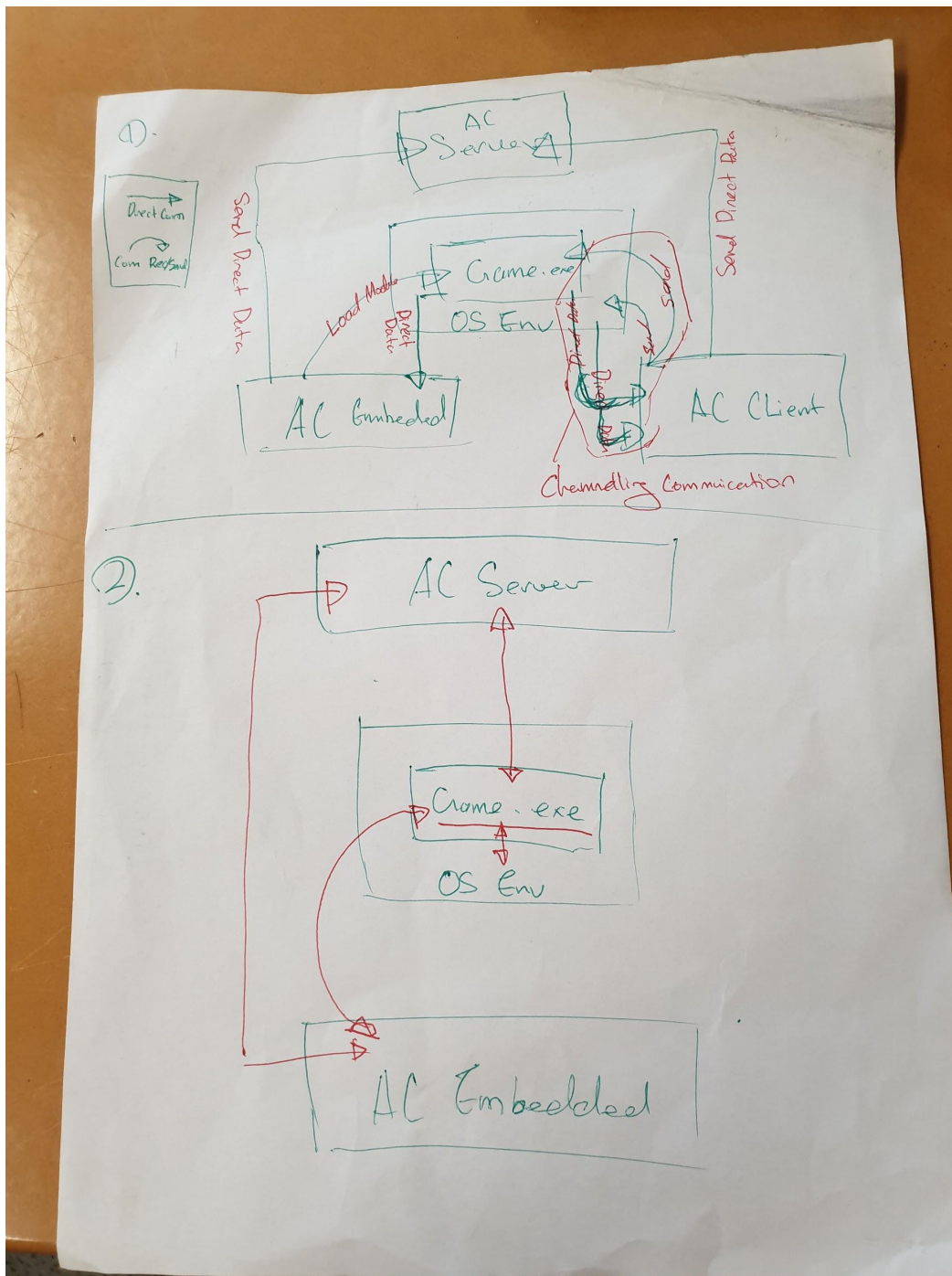
Goodwitch is an anti-cheat written in C# which supports traditional and modern user mode(ring3) anti-cheat features from process and debugger detection to detection of memory manipulations. It's name derived from the concept of the book 'Wizard of Oz', where the name Goodwitch is named after the witch who helped Dorothy in the storyline.

3.2 Why C# Though?

For Goodwitch, I have chosen to go with C# since C++ is traditionally and commonly used in anti-cheat development. Due to the above, I wanted to reverse the mindset that anti-cheats can only be developed in C++. I also wanted to show that for C# based games such as Unity Engine games, the Goodwitch library can be used to easily integrate and implement compatible C# anti-cheat solution. Java was also another option, however due to it's abstractness of implementing it to a native and managed processes as well as Java being compiled and run under the JVM which when compared to other languages it is slower in time complexity and resource heavy, it was disregarded.

3.3 Overall Design of Goodwitch

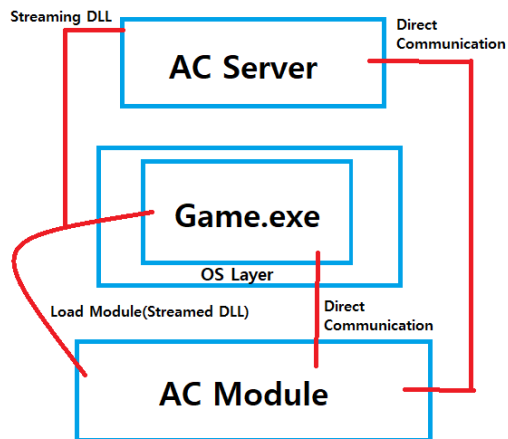
- Idea of Design 1



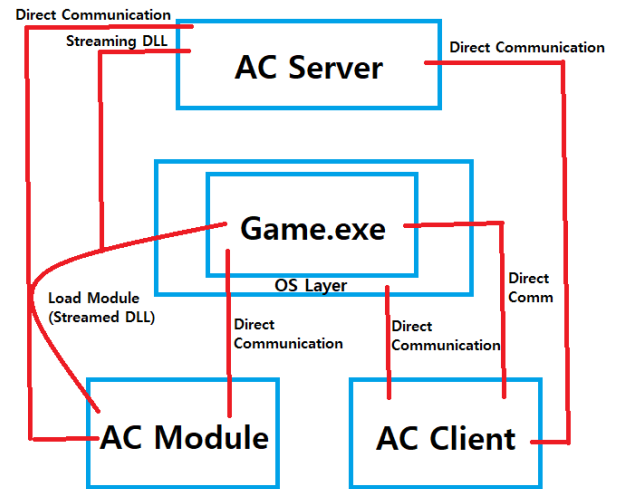
[Source 1.6]

- Idea of Design 2

1 Standalone AC Module



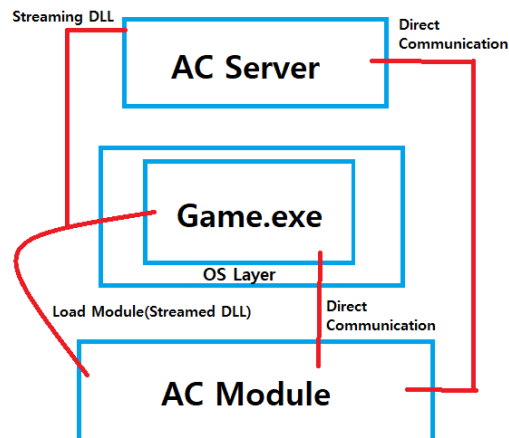
2 Multi-level AC with Module and Client



[Source 1.7]

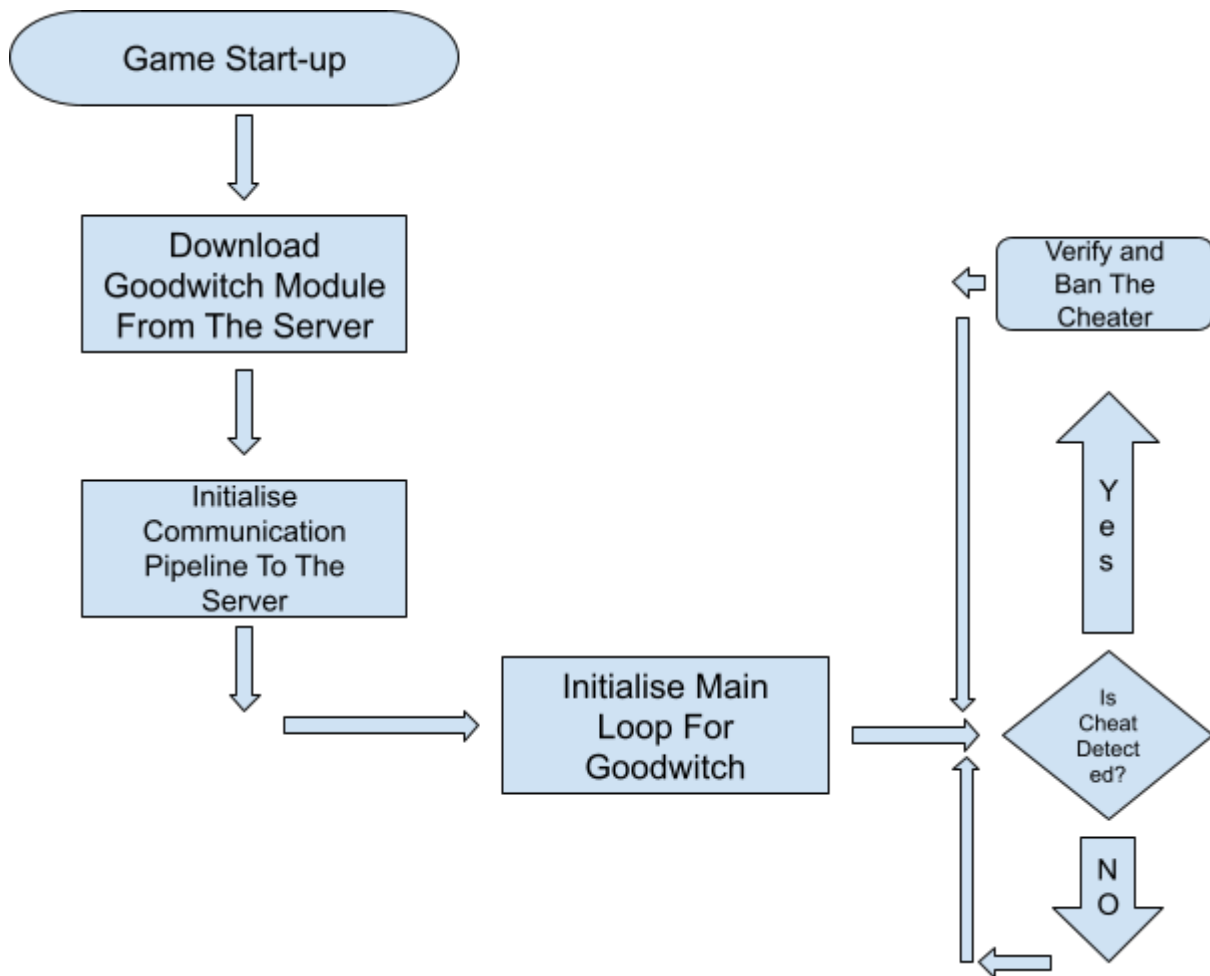
- Final Concluded Design To Proceed

1 Standalone AC Module



[Source 1.8]

- **Advanced Design Form of Final Concluded Design**



3.4 Breakdown of Goodwitch Modules

The module names derived it's concept from Wizard of Oz, where they are named after from strongest to weakest protagonists in hierarchical order in the book's worldview.

- **Oscar(Highest Hierarchical Module - 1) - Name of the Oz from Wizard of Oz**
Oscar module detects internal memory modification and memory related manipulations.
- **Kalidahs(Lowest Hierarchical Module - 2) - Name of the lion from Wizard of Oz**
Kalidahs module detects anti-debugging and process detection.

Chapter 4.

Development of Goodwitch

All the source code of this project is hosted on my personal Github:
<https://github.com/ZeroLP/Goodwitch>

4.0 Initial Coding

- **4.0A Setting Up Hooks - Oscar Module**

Hooking is used to alter the memory instructions, so that such function calls are replaced with modified function call instructions. Hooks here used are detour hooks, which will replace the first 5(6 instructions in total counting "ret") bytes of instructions of the original function, NOP(No Operation operand) the first 4 bytes of the instruction and JMP(JuMP operand) the last instruction to our modified function address.

- CreateThread Hook

Kernel32.CreateThread function is used to create a new additional operating thread in the calling process. When the CreateThread function is hooked, any assembly/module that calls CreateThread or System.Threading.Thread, it's stack and arguments will be monitored.

Code example of this hook is as below:

```
private unsafe static uint HookedCreateThread(UIntPtr lpThreadAttributes, uint dwStackSize, ThreadStart
    lpStartAddress, UIntPtr lpParameter, uint dwCreationFlags,
    ref uint lpThreadId)
{
    FlagRaiserService.RaiseFlag(DetectionFlags.THREAD_CREATED, $"Stacksize: {dwStackSize}
        | StartAddress:
        {lpStartAddress}");

    return pDetour.CallOriginal<uint>(OriginalInstance, new object[]
        { lpThreadAttributes, dwStackSize,
        lpStartAddress, lpParameter,
        dwCreationFlags, lpThreadId });
}
```

This hook monitors and sends a report of the called CreateThread function's arguments, where it can be assessed once received by the server.

The critical arguments that are closely monitored and assessed are the stacksize and start address of thread that is being created. Stacksize and start address relates to where the thread is started and where in the stack(page in the memory) is located, which can be assessed whether it was created inside of the actual game process or external third party assembly.

- VirtualProtect Hook

Kernel32.VirtualProtect function is used to change the protection of a section of a memory region/page in the calling process. When VirtualProtect function is hooked, any assembly/module that calls VirtualProtect, its stack and arguments will be monitored.

Code example of this hook is as below:

```
private unsafe static IntPtr HookedVirtualProtect(IntPtr lpAddress, IntPtr dwSize, uint flNewProtection,
                                                uint flProtect)
{
    FlagRaiserService.RaiseFlag(DetectionFlags.PROTECTION_CHANGED, $"Address: {lpAddress}
                                | New protection:
                                {flNewProtection}");

    return pDetour.CallOriginal<IntPtr>(OriginalInstance, new object[]
    { lpAddress, dwSize,
      flNewProtection, flProtect });
}
```

This hook monitors and sends a report of the called VirtualProtect function's arguments, where it can be assessed once received by the server.

The critical arguments that are closely monitored and assessed are the lpAddress and flNewProtection flag of the page section in memory. lpAddress and flNewProtection flag relates to where the protection of the memory is being changed and what type of protection is being changed, which can be assessed whether the page of the memory section is the critical part of the game's process and the protection flag being changed.

4.0A.1 - [Challenge Faced While Creating and Setting Up Hooks]

While creating and setting up the aforementioned hooks, a critical challenge was faced and essentially led to giving up on the hook implementations.

Due to how CLR(Common Language Runtime) manages and allocates the stacks and the limitation of detour hooking, hookings were not implemented in the final production. Initially for detour hooking, it replaced 6 static instruction bytes of the functions, and where the majority of the kernel32.dll's functions consists of two bytes of instruction(0xC3), it required to manually disassemble and calculate the true stack size of the instructions that are being replaced, and therefore, low level assembly(ASM) disassembly was needed where the limitation of C# as abstract and high level language came into place. Although existing hooks were replaced with native C# implementations of walking current thread and handle table for threads and handles created outside of the game's process.

- **4.0B Abnormal Assembly Loading Detection - Oscar Module**

Detecting abnormally loaded assembly is crucial to monitor and assess for any internal modification done by the forcefully injected/loaded assemblies. It is also crucial for monitoring known cheat assemblies, where signatures of the bytes are compared to the origin and known list of assemblies.

Essentially, it sets up two functions that are called on a tick based off of the system clock. First function takes the care of detecting the abnormal loadings, where it will enumerate through the assemblies list initialised on setup and the currently loaded assemblies list, then compares both of the assemblies and if it doesn't match, it raises a flag and generates a report of information consisting about the assembly and sends it to the server for further assessment.

Second function takes the care of detecting the known cheat assembly bytes signature, where it will execute the same process as the first function, however it takes the currently enumerating assembly's bytes signature and compares it against the known cheat assembly bytes signatures, and once it is detected, it displays a ban message and throws an exception, where will cause a crash on the game process, leading the game process to close and prevents further actions by the cheat assembly.

Below are the code example of the implementation:

Function 1: DetectAbnormalLoadings()

```
var appDomainASMList = AppDomain.CurrentDomain.GetAssemblies().ToList();

foreach (var asm in appDomainASMList)
{
    if (abnormalLoadedAssemblies.Contains(asm))
        continue;

    if (asm == appDomainASMList.Last() && loadedSafeAssemblies.Contains(asm))
        break;

    if (!loadedSafeAssemblies.Contains(asm))
    {
        Logger.Log($"Abnormal assembly: {asm} has been loaded.", Logger.LogSeverity.Warning);

        abnormalLoadedAssemblies.Add(asm);
        FlagRaiserService.RaiseFlag(Enums.DetectionFlags.ABNORMAL_LOADING_DETECTED, ConstructFlagInformationForAssembly(asm));
    }
}
```

Function 2: DetectKnownCheatSignature()

```

foreach(var asm in abnormalLoadedAssemblies)
{
    string currASMBytes = Convert.ToBase64String(File.ReadAllBytes(asm.Location));

    foreach(var detectedSignature in detectedKnownCheatByteSignatures)
    {
        if (currASMBytes.Contains(detectedSignature))
        {
            BanMessageDisplayer.DisplayBanMessage();
        }
    }
}

```

- 4.0C Detecting Running and Attached Debuggers and Known Cheat Processes - Kalidahs Module**

Detecting running and attached debuggers are crucial in order to prevent tampering and dumping of the game's process. This is implemented by calling the WinAPI's `IsDebuggerPresent` and `CheckRemoteDebuggerPresent` functions where `IsDebuggerPresent` checks for the debuggers that are physically attached to the game's process, while `CheckRemoteDebuggerPresent` checks for remote debuggers that are present in a separate and parallel process with debug privileges to the game's process. Additionally, it enumerates through all running processes and checks against their process name, windows handle title and icon bytes signature. On the other hand, detecting known cheat processes implies the similar techniques to detecting debuggers, however, however it takes the currently enumerating process's bytes signature and compares it against the known cheat process bytes signatures, and once it is detected, it displays a ban message and throws an exception, where will cause a crash on the game process, leading the game process to close and prevents further actions by the cheat process.

Below are the code example of the implementations:

Main Function for Detecting Debuggers: `DetectDebuggers()`

```

private void DetectDebuggers()
{
    if(IsDebuggerAttached() || IsRemoteDebuggerAttached()
        || IsDebuggerRunningPrcName() || IsDebuggerRunningHWND())
    {
        BanMessageDisplayer.DisplayBanMessage();
    }
}

```

Main Function for Detecting Known Cheat Softwares:

DetectKnownCheatApplication()

```
private void DetectKnownCheatApplication()
{
    if (IsCheatRunningPrcName() || IsCheatRunningHWND())
    {
        BanMessageDisplayer.DisplayBanMessage();
    }
}
```

Function for Detecting Known Cheat Software Byte Signatures:

DetectKnownCheatApplicationSignature

```
foreach (var proc in Process.GetProcesses())
{
    string currASMBytes =
        Convert.ToBase64String(File.ReadAllBytes(proc.MainModule.FileName));

    foreach (var detectedSignature in CheatProcessByteSignature)
    {
        if (currASMBytes.Contains(detectedSignature))
        {
            BanMessageDisplayer.DisplayBanMessage();
        }
    }
}
```

- **4.0D Client To Server / Server To Client Communication Implementation**

Server communication is crucial for anticheats, especially when an unknown cheat software is flagged, it provides an ease of access to the information collected by the anticheat module itself. For Goodwitch, the server handles all the incoming reports generated by the client and depending on the report type, it executes certain functions where reported information is taken care of.

It initially connects to the server with TCP client where the packets are encrypted with XOR bit shifting and decrypts it back when received, and generates a unique hardware ID(HWID) based on the hardware components identifiers, then registers in a JSON database on the server.

Below are the the implementation of the communications:

Function To Connect To The Server

```
private static Tuple<bool, string> ConnectToServer()
{
    try
    {
        ServerSocket = new TcpClient();
        var Res = ServerSocket.BeginConnect("127.0.0.1", 8971, null, null);

        var success = Res.AsyncWaitHandle.WaitOne(TimeSpan.FromSeconds(3));

        if (!success)
        {
            return new Tuple<bool, string>(false, $"Exception: Failed to connect to the server.");
        }

        NStream = ServerSocket.GetStream();

        return new Tuple<bool, string>(true, "");
    }
    catch (Exception Ex)
    {
        if (Ex.GetType() == typeof(SocketException))
            return new Tuple<bool, string>(false, $"Exception: Failed to connect to the server.");
        else return new Tuple<bool, string>(false, $"Exception: {Ex.Message}");
    }
}
```

Functions To Encrypt And Decrypt Packets

```
private static string EncryptPacket(string Packet)
{
    //XOR -> Bitshift

    string XORKey = "bbcd563cf93676e1312a34fc8347c993";

    char[] XORString = new char[Packet.Length];

    //XOR Operation
    for (int i = 0; i < Packet.Length; i++)
    {
        XORString[i] = (char)(Packet[i] ^ XORKey[i % XORKey.Length]);
    }

    char[] BitshiftedXORString = new char[XORString.Length];

    //Bitshifting XOR string
    for (int i = 0; i < XORString.Length; i++)
    {
        BitshiftedXORString[i] = (char)(XORString[i] << 4);
    }

    return new string(BitshiftedXORString);
}
```

```
private static string DecryptPacket(string Packet)
{
    string XORKey = "bbcd563cf93676e1312a34fc8347c993";

    char[] ReverseBitshiftedString = new char[Packet.Length];
    for (int i = 0; i < Packet.Length; i++)
    {
        ReverseBitshiftedString[i] = (char)(Packet[i] >> 4);
    }

    char[] ReverseXORString = new char[ReverseBitshiftedString.Length];
    for (int i = 0; i < ReverseBitshiftedString.Length; i++)
    {
        ReverseXORString[i] = (char)(ReverseBitshiftedString[i] ^ XORKey[i %
                                                                    XORKey.Length]);
    }

    return new string(ReverseXORString);
}
```

Functions To Handle Incoming Reports

```
//Main hub to handle all incoming reports besides from authentication
internal static void HandleIncomingReports(NetworkStream NStream)
{
    var resp = ServerTelemetry.ReadPacket(NStream);

    if (resp.Item1 && resp.Item2.StartsWith("GWReportType"))
    {
        if (resp.Item2.Contains("GWReportType: AbnormalLoadingDetected"))
            ReportHandling.AbnormalLoadingDetection.HandleDetection(resp.Item2);
        else if (resp.Item2.Contains("GWReportType: KnownCheatSoftwareDetected"))
            ReportHandling.KnownCheatSoftwareDetection.HandleDetection(resp.Item2);
    }
}
```

4.1 Implementing It To A Real World Example - Game

Now all the code level implementations are done, anticheat cannot work without a game. This is where the implementation of Goodwitch to a real work example comes into place. The game chosen for this demonstration purpose is C# Winform SpaceInvaders game by Kubikola, which is an open source C# game project. The project is hosted at <https://github.com/Kubikola/SpaceInvaders>.

Implementation of Goodwitch on the SpaceInvaders is as below:

```
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    var LoadGoodwitchAC = Goodwitch.Main.InitialiseGoodwitch();

    if (LoadGoodwitchAC.Item1)
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
    else
    {
        MessageBox.Show(LoadGoodwitchAC.Item2, "Goodwitch Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error);

        Environment.Exit(0);
    }
}
```

As seen in the code implementation, Goodwitch is loaded before the runtime creates the visual components and the main form is executed after everything is loaded properly. If any exception occurs while loading Goodwitch, it will output to a message box then when the “OK” button is clicked, it will exit the application’s environment.

4.2 Feature Demonstrations

Below video demonstrates the explored designs and implementations of Goodwitch, where it showcases abnormal assembly loading detection and reporting, known cheat assembly detection, known cheat process detection and debugger detection.

Youtube Video: <https://youtu.be/9VvwJfDfn2c>

Chapter 5.

Project Schema

5.1 Criteria to evaluate success

Area	Criteria	Justification and Analysis	Method of Evaluation
Function	1) Prevent third-party injection 2) Prevent tampering of the game. (Anti-Dumping, Anti-Debugging, Anti-Decompiling) 3) Prevent execution of third-party applications 4) Detection of known/detected cheats.	1) Third-party injection known as DLL injection is the most common base method of all game cheats. 2) Cheaters can reverse engineer the game or even tamper the game to acquire access to game’s source code and unwanted leak to the game for malicious use. 3) Cheaters often or majorly use specific tools to manipulate and hack the game and it’s memory. 4) Detection of game cheats is the core functionality of an AntiCheat to detect and ban the cheaters.	1) Software Reverse Engineers / Crackers 2) Gamers

Sustainability	1) Prevention of high resource usage	1) If the system were to use high resources as part of the execution, it is redundant and inefficient since the game may be graphic and processor intensive.	1) Load balance testing
Service of Others	1) To provide clean gaming experience without the interruption of illegal advantage players	1) Anticheat is made to provide services of preventing cheaters in the game, but also to serve the people in the game providing a clean and fair experience while playing.	1) User reports 2) Feedbacks 3) Surveys
Element of Uniqueness and Innovation	1) Full Scale C# based	1) Full scaled anticheat written in C# has never been tried in the market before, hence even if there are, it bounds to only support a small amount of features as anti cheat.	1) Market research 2) User experience
Impact On Individual, Society and The Environment	1) Individual - Clean and fair gaming environment 2) Society - Lessening the count of cheaters in games and to the world of game cheating. 3) Environment - N/A	1) Individual - People who play the game and also who cheats in the game can get clean and fair gaming environment through MDP. 2) Society - Socially, cheaters been a heat on media news especially infamous for huge amount of money being involved. Through MDP, issues outlined by the media can be lessened or even resolved	1) Individual - User reviews - Surveys - Game developer reviews 2) Society - Media research - World-wide impact research

5.2 Time & Action Plan

2019 - Term 4

WEEK	Action	Done By	Variation
Wk1	Project Ideas	18/10/2019	1 - 2 days
Wk2	Pmi analysis / Criteria eval success	25/10/2019	1 - 2 days
Wk3	Industry excursion	31/10/2019	1 - 2 days
Wk4	Sketches(Demo/Sample)	8/11/2019	1 - 2 days
Wk5	Collaboration	15/11/2019	1 - 2 days
Wk6	Project determination	22/11/2019	1 - 2 days
Wk7	Project Proposal	27/11/2019	1 - 2 days
Wk8	Create Survey for target market/audience	6/12/2019	1 - 2 days
Wk9	Send Out Survey	12/12/2019	1 - 2 days

2020 - Term 1

WEEK	Action	Done By	Variation
Wk1	Research anticheat methods & delegations Research methods of cheats and cheat system		
Wk2	Research anticheat methods & delegations Research methods of cheats and cheat system		
Wk3	Research anticheat methods & delegations Research methods of cheats and cheat system		
Wk4	CAMP		
Wk5	Research anticheat		

	methods & delegations Research methods of cheats and cheat system		
Wk6	Research anticheat methods & delegations Research methods of cheats and cheat system	14/02/2020	4 - 6 days
Wk7	Start development of anticheat	21/02/2020	
Wk8	Coding		
Wk9	Coding		
Wk10	Coding		

2020 - Term 2

WEEK	Action	Done By	Variation
Wk1	Coding		
Wk2	Coding		
Wk3	Coding		
Wk4	Coding		
Wk5	Coding	30/05/2020	4 - 6 days
Wk6	ASSEMBLY(Integration)		
Wk7	ASSEMBLY(Integration)		
Wk8	ASSEMBLY(Integration)	20/06/2020	4 - 6 days
Wk9	FINISH		
Wk10	FINISH		

5.3 Area of Investigation

Necessary Modelling, Prototyping and Mock-Up Techniques	<ul style="list-style-type: none"> - Software Prototyping - Feature Brainstorming
Materials	N/A
Design Techniques	<ul style="list-style-type: none"> - Diagramming
Shaping/Forming	<ul style="list-style-type: none"> - Prototyping - Alpha testing - Sandboxed Testing - Coding - Unit Testing - Window detection - Process Detection - Module Scan - Injection Scan/Detection - Cheat Scan/Detection
Assembly	<ul style="list-style-type: none"> - Compiling - Integrating - IDE(Integrated Development Environment) - Compiler - Error Correction(Syntax, Code Redundancy)
Finishing Process	<ul style="list-style-type: none"> - Development - Testing -> Production Testing
Testing and Experimenting	<ul style="list-style-type: none"> - Unit Testing - Window detection - Process Detection - Module Scan - Injection Scan/Detection - Cheat Scan/Detection - Load Balance Testing - Integration Testing - Acceptance Testing - Compatibility Testing

5.4 Financial Plan

Due to the project being a software development project, no material costs are required, and therefore considered as funds are redundant unconditionally.

Chapter 6.

Evaluation

6.1 Application of Evaluation & Analysis and Evaluation of Functional and Aesthetic Aspects of Design

- **Analysis of Preventing Third-Party Injection**

Third-party injection known as DLL injection is the most common base method of all game cheats, and with the utilisation of below code for abnormal assembly loading detection and report handling, third party injection is prevented. Although, while effectiveness is valid on the detected assembly loading detection, abnormal assembly loading detection is evaluated to be less effective due to only raising flags but not preventing. However, it is intended to be less effective due to the possibilities of false flagging and reporting of it.

Function 1: DetectAbnormalLoadings()

```
var appDomainASMLList = AppDomain.CurrentDomain.GetAssemblies().ToList();

foreach (var asm in appDomainASMLList)
{
    if (abnormalLoadedAssemblies.Contains(asm))
        continue;

    if (asm == appDomainASMLList.Last() && loadedSafeAssemblies.Contains(asm))
        break;

    if (!loadedSafeAssemblies.Contains(asm))
    {
        Logger.Log($"Abnormal assembly: {asm} has been loaded.", Logger.LogSeverity.Warning);

        abnormalLoadedAssemblies.Add(asm);
        FlagRaiserService.RaiseFlag(Enums.DetectionFlags.ABNORMAL_LOADING_DETECTED, ConstructFlagInformationForAssembly(asm));
    }
}
```

Function 2: DetectKnownCheatSignature()

```
foreach(var asm in abnormalLoadedAssemblies)
{
    string currASMBytes = Convert.ToBase64String(File.ReadAllBytes(asm.Location));

    foreach(var detectedSignature in detectedKnownCheatByteSignatures)
    {
        if (currASMBytes.Contains(detectedSignature))
        {
            BanMessageDisplay.DisplayBanMessage();
        }
    }
}
```

- **Analysis of Preventing Tampering of The Game (Anti-Dumping, Anti-Debugging, Anti-Decompiling) & Preventing Execution of Third-Party Applications**
Cheaters can reverse engineer the game or even tamper the game to acquire access to game's source code and unwanted leak to the game for malicious use. For the purpose of preventing such situations, below code is utilised to detect loading of debuggers and execution of it. While it is evaluated to be effective to a certain extent, being in the nature of ring3 provides varieties of loopholes to the kernel-mode debuggers to debug our ring3 game process.

Main Function for Detecting Debuggers: DetectDebuggers()

```
private void DetectDebuggers()
{
    if(IsDebuggerAttached() || IsRemoteDebuggerAttached()
        || IsDebuggerRunningPrcName() || IsDebuggerRunningHwnd())
    {
        BanMessageDisplay.DisplayBanMessage();
    }
}
```

- **Analysis of Detecting Known/Detected Cheats**

Detection of game cheats is the core functionality of an Anti-Cheat to detect and ban the cheaters and It also prevents further use of the same cheat repeatedly. With the utilisation of below codes, it is evaluated that, while it is effective in detecting known cheat processes, the same nature of tamper prevention applies here as well; where, if a cheat process resides in a ring0 layer, it is subjectively difficult or even impossible to detect the execution of it.

Main Function for Detecting Known Cheat Softwares:
DetectKnownCheatApplication()

```
private void DetectKnownCheatApplication()
{
    if (IsCheatRunningPrcName() || IsCheatRunningHWND())
    {
        BanMessageDisplay.DisplayBanMessage();
    }
}
```

Function for Detecting Known Cheat Software Byte Signatures:
DetectKnownCheatApplicationSignature

```
foreach (var proc in Process.GetProcesses())
{
    string currASMBytes =
        Convert.ToBase64String(File.ReadAllBytes(proc.MainModule.FileName));

    foreach (var detectedSignature in CheatProcessByteSignature)
    {
        if (currASMBytes.Contains(detectedSignature))
        {
            BanMessageDisplay.DisplayBanMessage();
        }
    }
}
```

References

[Source 1.1]: https://en.wikipedia.org/wiki/Protection_ring#/media/File:Priv_rings.svg

[Source 1.3]: <https://i.stack.imgur.com/H5OTp.png>