STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Využití reverzního inženýrství k tvorbě softwaru pro získávání výhod v online hrách a analýza jeho detekce

Filip Touš
Středočeský Kraj

Nymburk 2019

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Využití reverzního inženýrství k tvorbě softwaru pro získávání výhod v online hrách a analýza jeho detekce

Using reverse engineering to create software for gaining advantage in online games and an analysis of its detection

Autor: Filip Touš
Škola: Gymnázium Bohumila Hrabala v Nymburce, Komenského 779, 288 40 Nymburk
Kraj: Středočeský Kraj

Nymburk 2019

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Nymburce dne 12. 3. 2019 ……

Filip Touš

Anotace

Cílem práce bylo vytvořit software pro získávání neoprávněných výhod v počítačových online hrách (tzv. cheat), konkrétně pro hru Team Fortress 2 (TF2) na operačním systému Windows. V první části jsou popsány metody reverzního inženýrství použité pro pochopení vnitřních funkcí a struktury herního procesu, využití funkcí API Windows pro injekci vlastního kódu do procesu hry a je vysvětleno jak může cheat modifikovat běh hry. Část druhá se věnuje samotným funkcím cheatu a jejich algoritmům. V poslední části jsou popsány možné detekční vektory pro odhalení tohoto cheatu a analyzován anti-cheatovací software zodpovědný za ochranu hry TF2 před cheaty. Dále jsou uvedeny a v cheatu aplikovány alternativy používaných metod k obejití těchto detekcí a zároveň také představena možná vylepšení anti-cheatu hry TF2. Nakonec jsou zmíněny další možnosti tvorby těžce detekovatelných cheatů a reakce jiných, pokročilejších anti-cheatovacích softwarů na tyto cheaty. Na přiloženém CD je kompletní zdrojový kód cheatu a ukázky jeho funkcí.

Klíčová slova

cheat, reverzní inženýrství, injekce kódu, hookování funkcí, Windows API

Annotation

The goal of this work was to create a software for gaining unfair advantage in online computer games (a.k.a. "cheat"), for the game Team Fortress 2 (TF2) on the Windows operating system. The first part describes reverse engineering methods used for understanding the inner structure and functions of the game process, the usage of Windows API functions for injecting custom code into the process of the game and explains how the cheat can modify the behavior of the game. The second part is dedicated to the actual functions of the cheat and its algorithms. The last part describes possible detection vectors of the cheat and analyzes the anti-cheating software that is responsible for protecting TF2 from cheats. Alternatives of the used methods are listed and applied to the cheat for the purpose of evading its detection, and possible improvements of the anti-cheat software are presented. Lastly are mentioned other available options of creating harder-to-detect cheats and the reaction of other, advanced anti-cheat software to these methods. The attached CD contains the complete source code of the cheat and demonstrations of its functions.

Keywords

cheat, reverse engineering, code injection, function hooking, Windows API

Obsah

1 Úvod 6
2 Principy fungování cheatu 8
2.1 Virtuální adresní prostor v OS Windows 8
2.1.1 Bytová signatura 8
2.2 Použité nástroje pro tvorbu cheatu 9
2.2.1 Cheat Engine 9
2.2.2 ReClass.NET 9
2.2.3 IDA Pro a Hex-Rays Decompiler 10
2.2.4 x64dbg 10
2.3 Injekce kódu 11
2.4 Hookování funkcí 12
2.4.1 VMT hooking 12
2.4.2 Window procedure 14
2.5 Source engine 15
2.6 Běh cheatu od injektování po hookování herních funkcí 16
2.7 Používání funkcí Source enginu 17
2.7.1 NetVar a ConVar 18
3 Funkce cheatu 19
3.1 Aimbot 19
3.1.1 Skrytí používání aimbotu 21
3.1.2 Projektilový aimbot 22
3.2 Triggerbot 23
3.3 Anti-aim 23
3.4 Speedhack 25
3.5 Auto-airblast 26
3.6 Auto-Sticky detonation 26
3.7 ESP 27
3.7.1 Radar 28
3.8 Menu 28
3.9 Ostatní funkce 29
4 Detekovatelnost cheatu 30
4.1 Valve anti-cheat 30

4.2 Skenování paměti 31
4.2.1 Virtualizace 31
4.2.2 XOR šifrování 31
4.2.3 Mutace 32
4.3 Skrytí injekce kódu 32
4.3.1 Obcházení LoadLibrary 33
4.3.2 Přepsání PE File headeru 34
4.4 Skrytí souboru cheatu 34
4.5 Možná vylepšení VAC 34
4.6 Možné detekce v TF2 35
4.7 Co dál? 35
4.7.1 Manual mapping a thread hijacking 36
4.7.2 Kernelové cheaty 36
4.7.3 Hardwarové cheaty 37
4.8 Legální aspekt 37
5 Závěr 38
6 Použitá literatura 39
7 Seznam obrázků 41
8 Příloha 1: Ukázka zdrojového kódu - funkce triggerbot 42
9 Příloha 2: Zdrojový kód ldrloaddll injektoru 43

1 Úvod

Podvádění je součástí videoher od jejich samotného počátku, hry vycházely (a často stále vycházejí) s vestavěnými funkcemi pro ulehčení / modifikování herního zážitku, které se dali vyvolat například sekvencí stisknutí specifických kláves. Ve hře DOOM (1993) se po stisknutí kláves "IDDQD" aktivoval stav kdy herní postava nemůže být poškozena - stala se tedy nesmrtelnou. To mohlo hráči například pomoct překonat těžký úsek hry, nebo jednoduše přinést hráči lepší požitek ze hry. Hráči ale nebyli závislí pouze na tom, co jim vývojáři poskytli. Už v době 8-bitových počítačů bylo známou praxí modifikovat načtenou paměť hry za účelem podvádění (1).Tím se hráčům velmi rozšířily možnosti úpravy běhu hry. Dnes jsou pro většinu her na internetu volně dostupné desítky cheatů jako spustitelné soubory s různými funkcemi, které hráč může lehce nastavovat skrze uživatelské rozhraní nebo pomocí kláves.

S příchodem internetu a online her, kde hráči bojují proti sobě, se z podvádění stal velký problém a započala tak velká hra na kočku a myš mezi vývojáři cheatů a vývojáři her a anti-cheatů - softwaru který se snaží cheatovací software odhalit, zamezit v jeho působení a ve většině případů podvádějícího hráče penalizovat tzv. "banem", což mu zablokuje přístup do online části hry (v některých případech celé hry, nebo dokonce všem hrám na stejném enginu apod.) Cheateři (hráči používající cheatovací software k získání výhod nad ostatními) kazí herní zážitek ostatním hráčům a proto je cheatování v herní komunitě často diskutované téma. S příchodem pro-gamingu a turnajů s miliony dolarů ve výhrách (2) a miliardových obratech videoherního průmyslu je cheatování narůstající problém i pro herní vývojáře (3).

Na internetu jsou zdarma dostupné cheaty i pro některé online hry, ale jejich používání velmi často vede k rychlé penalizaci hráče banem, což vyvolalo poptávku po pokročilejších cheatech. Vytvoření pokročilého, feature-rich cheatu pro online hru, který proti anti-cheatu obstojí, je úkol vyžadující pokročilé znalosti z mnoha oblastí: programovacího jazyka, způsobu fungování virtuální paměti, reverzního inženýrství, jazyka assembly, architektury operačního systému (téměř výhradně Windows) atd. Z cheatů pro online hry se stal milionový byznys (4). Cheateři jsou ochotni platit desítky dolarů za možnost podvádět ve hrách a za nižší riziko odhalení.

Většina online her funguje na principu klient-server a proto principielně jednoduché funkce jako je nesmrtelnost nebo zvýšení rychlosti pohybu (tzv. speedhack) apod. jsou v online hrách až na výjimky neproveditelné, protože o herních akcích rozhoduje server, nikoliv klient. Úprava zdraví / životů herních postav v paměti klienta by tak na chod hry neměla žádný vliv, jelikož skutečnou hodnotu uchovává a upravuje server. Ve hře Team Fortress 2 hlídá rychlost postav i všechny další hodnoty a kontroluje, zda jsou v povolených mezích, server. Obecně se (nejen ve videohrách) používá design "Nikdy nevěř klientovi" (5). Stává se však, že i v takto koncipovaných hrách existují chyby, které lze zneužít, čehož jsou příkladem i některé funkce naprogramované v mém cheatu.

Mezi nejčastější funkce cheatů pro online hry dnes patří:

- zobrazování skrytých herních informací (například pozice schovaného nepřítele, jeho zdraví atp.)
- automatizace nebo asistence při vykonávání akcí (např. míření)
- umožnění hráči provádět akce které obyčejně nejsou proveditelné

Někteří cheateři se snaží cheaty používat nenápadně, aby získali malou nebo značnější výhodu nad soupeřem, která může nanejvýš vzbudit u ostatních hráčů podezření, že hráč cheatuje, ale nikoliv takovou, která by byla ostatním zřejmá nade vší pochybnost. Tomuto stylu podvádění se někdy říká "legit cheating" (legitimní podvádění). Naopak pojmem "rage" cheateři (běsnící cheateři) jsou označováni hráči, kteří využívají všech funkcí cheatu bez ohledu na to, že to ostatní hráči vidí. V TF2 mají hráči možnost hlasovat o vyloučení spoluhráče ze serveru, častěji než anti-cheat se tak o rage cheatery postarají sami hráči.

Technicky můžeme cheaty rozdělit na dvě skupiny:

- Externí - využívá vlastního procesu, popř. ovladače nebo i hardwaru pro získávání dat z paměti hry nebo modifikaci chování hry. Samotný cheat se tedy nenachází uvnitř virtuální paměti hry a neběží v herním procesu.
- Interní - injektuje vlastní kód do procesu hry, nejčastěji v podobě dynamické knihovny (soubor s koncovkou DLL), cheat je tedy uvnitř virtuální paměti hry.

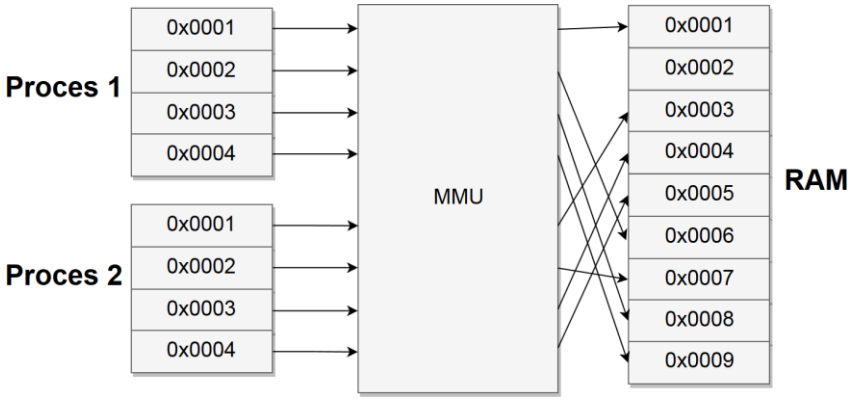
Cheat vytvořený v rámci této práce je interní a je sestaven jako DLL soubor. Hra Team Fortress 2 je vyvíjena společností Valve Corporation (Valve) a je distribuována na službě Steam (také od společnosti Valve), kde je od roku 2011 dostupná zdarma (resp. jako freemium, také označováno pojmem Free-to-play). Cheat je napsán v jazyku C++ ve vývojářském prostředí Visual Studio 2017.

Motivací pro tuto práci byla možnost vyzkoušet si znalosti v oboru IT v praxi a seznámit se s metodami, které jsou používané v oboru informační a počítačové bezpečnosti, zejména reverzním inženýrstvím a skrývání / detekování škodlivého kódu, pro další pokračování studia a přípravu pro profesní kariéru v této oblasti.

2 PRINCIPY FUNGOVÁNÍ CHEATU

2.1 Virtuální adresní prostor v OS Windows

Každý proces v OS Windows má přístup k vlastnímu virtuálnímu setu adres, tzv. Virtual adress space (VAS). VAS každého procesu je privátní a cizí procesy k němu nemají přístup. Virtuální adresa nereprezentuje skutečnou adresu v paměti počítače (RAM). Jednotka MMU (Memory management unit) v procesoru je zodpovědná za jejich překlad na fyzické adresy. Tento systém mimo jiné umožňuje, aby se paměť programu jevila jako lineárně uspořádaná i když je ve skutečnosti na různých místech ve fyzické paměti (6).



Obrázek 1: Virtuální adresní prostor v OS Windows

Ukazatel na objekt uchovává pouze počáteční adresu objektu v paměti, nikoli jeho skutečné hodnoty. Pokud je potřeba manipulovat např. s proměnnou třídy, může být rychlejší a jednodušší používat jen ukazatel na třídu samotnou a reverzním inženýrstvím zjistit offset hledané proměnné od prvního člena třídy. Offset je v tomto případě počet bytů, které je potřeba přičíst (nebo odečíst) k ukazateli na třídu k získání adresy hledané proměnné.

2.1.1 Bytová signatura

Cheat pro získání adres některých funkcí a tříd používá tzv. bytovou signaturu, protože nebyl nalezen žádný výchozí bod, kterého by mohl využít pro lokalizaci hledaného objektu. V takovém případě je pomocí reverzního inženýrství nalezena sekvence bytů, která se vždy objevuje kolem hledané adresy. Např. pro získání adresy funkce *EstimateAbsVelocity* cheat používá následující signaturu:

```
("E8 ? ? ? ? F3 0F 10 4D ? 8D 85 ? ? ? ? F3 0F 10 45 ? F3 0F 59 C9 56 F3 0F 59 C0 F3
0F 58 C8 0F 2F 0D ? ? ? ? 76 07")
```

Byty jsou zapsány v hexadecimálním formátu. Otazníky označují byty jejichž hodnota není pokaždé stejná. Následující kód funkce *dwFindSignature* postupně pročítá adresy od určené počáteční adresy do určené konečné adresy. Pokud hodnota na právě čtené adrese odpovídá prvnímu bytu signatury (nebo je v signatuře otazník), přečte hodnotu na adrese o jeden byte vyšší a testuje zda odpovídá druhému bytu v signatuře atd. Pokud další byte neodpovídá,

pokračuje na další adrese znovu od začátku signatury. Když se úspěšně dostane až na poslední byte v signatuře, vrátí adresu prvního odpovídajícího bytu.

```
#define SCOPE(x,a,b)    (x >= a && x <= b)
#define BITS( x )    (SCOPE((x&(~0x20)),'A','F') ? ((x&(~0x20)) - 'A' + 0xa) :
(SCOPE(x,'0','9') ? x - '0' : 0))
#define READ( x )    (BITS(x[0]) << 4 | BITS(x[1]))

DWORD CSignature::dwFindSignature(DWORD dwAddress, DWORD dwLength, const char*
szPattern)
{
	const char* pat = szPattern;
	DWORD firstMatch = NULL;
	for (DWORD pCur = dwAddress; pCur < dwLength; pCur++)
	{
		if (!*pat) return firstMatch;
		if (*(PBYTE)pat == '\?' || *(BYTE*)pCur == READ(pat))
		{
			if (!firstMatch) firstMatch = pCur;
			if (!pat[2]) return firstMatch;
			if (*(PWORD)pat == '\?\?' || *(PBYTE)pat != '\?') pat += 3;
			else pat += 2;
		}
		else {
			pat = szPattern;
			firstMatch = 0;
		}
	}
	return NULL;
}
```

2.2 Použité nástroje pro tvorbu cheatu

2.2.1 Cheat Engine[1]

Dříve zmíněný Cheat Engine je užitečným nástrojem pro skenování virtuální paměti procesu a hledání adres a ukazatelů na tyto adresy nebo textových řetězců. Může být také použit jako debugger pro zjištění které funkce pracují s danou adresou a přepis strojového kódu do assembly.

2.2.2 ReClass.NET[2]

ReClass funguje na podobném principu jako Cheat Engine, ale je designovaný pro práci se třídami a datovými strukturami (z čehož také plyne jeho název). Umožňuje přiřazovat hodnotám názvy a datové typy nebo ukazatele a zpětně tak vytvářet třídy a struktury bez původního zdrojového kódu pro zjednodušení psaní cheatu. Velmi užitečným nástrojem je pak schopnost generovat C++ nebo C# kód z vytvořených tříd, s názvem třídy a jejích členů,

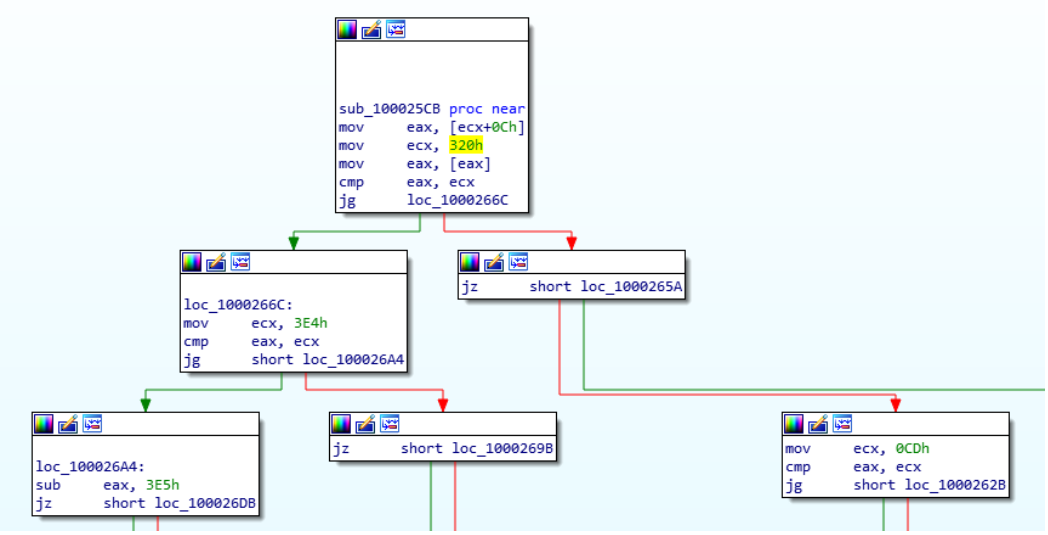
[1] Cheat Engine je zdarma a open-source. Dostupný z: https://www.cheatengine.org/
[2] Dostupné zdarma z: https://github.com/ReClassNET/ReClass.NET

včetně datového typu a offsetu (počtu bytů) od první položky. ReClass.NET je vylepšená verze původního open-source programu ReClass, přepsaná pro platformu .NET Framework.

2.2.3 IDA Pro a Hex-Rays Decompiler[3]

IDA (zktratka pro Interactive Dissasembler) je standard v oboru reverzního inženýrství. Společně s integrovaným programem Hex-Rays Decompiler tvoří pokročilý multi-platformní disassembler a debugger pro desítky typů procesorů. IDA nabízí krom obyčejného textového výpisu v assembly také diagramové znázornění běhu programu a tzv. Proximity view, který graficky znázorňuje propojení funkcí mezi sebou. Většina statické i dynamické analýzy hry byla provedena právě v IDA Pro.

IDA také podporuje tvorbu zásuvných modulů - pluginů. Jedním z nich je SigMaker[4], který je používán pro jednoduché vytváření bytových signatur.



Obrázek 2: Diagramové zobrazení v IDA Pro

2.2.4 x64dbg[5]

x64dbg (respektive jeho část x32dbg, program používá odlišná rozhraní podle architektury analyzovaného souboru) je open-source debugger pro Windows. Některé jednodušší problémy při vyvíjení cheatu byly řešeny v x64dbg namísto IDA Pro, spíše ale pro rychlost a přehlednost než jeho funkčnost.

---

[3] Kromě licencí pro školy se jedná placený software. Stránky vývojáře: https://www.hex-rays.com/
[4] SigMaker je open-source, dostupný zdarma z: https://github.com/ajkhoury/SigMaker-x64
[5] Zdarma dostupné z: https://x64dbg.com

2.3 Injekce kódu

Jak již bylo zmíněno, cheat je sestaven jako dynamická knihovna (DLL), nikoliv jako spustitelný soubor a je tedy za potřebí ještě program, který injektuje cheat do procesu hry, tzv. injektor. Ten provádí následující úkoly:

1. Zjistí aktuální PID herního procesu ("hl2.exe").
2. Vytvoří handle[6] pro přístup k hernímu procesu s přístupovými právy pro tvorbu nového vlákna, získávání informací o procesu, čtení a zapisování do paměti procesu a operace s pamětí procesu pro změnu ochrany jejích částí.
3. Alokuje paměť v procesu s ochranou umožňující zápis a čtení do paměti pro zapsání cesty k souboru cheatu.
4. Do alokovaného prostoru zapíše cestu k cheatu (soubor "cheat.dll").
5. Vytvoří v procesu nové vlákno které volá funkci *LoadLibrary* s ukazatelem jako argument. Ten ukazuje na alokovanou paměť, kde je zapsána cesta k souboru cheatu.
6. *LoadLibrary* načítá do herního procesu modul cheatu v podobě cheat.dll a automaticky volá jeho funkci *DLLMain*. Řízení herního procesu je teď v rukou cheatu.
7. Injektor uzavře handle a ukončí se.

Názvy procesu a DLL vepsány přímo v injektoru. Vyžaduje jen aby injektor a cheat byli ve stejné složce.

```
#include <iostream>
#include <Windows.h>
#include <TlHelp32.h>
int main()
{	DWORD dwPID;
	HANDLE hPID = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);
	PROCESSENTRY32 ProcEntry;
	ProcEntry.dwSize = sizeof(ProcEntry);
	do{
		if (!strcmp(ProcEntry.szExeFile, "hl2.exe"))
		{	dwPID = ProcEntry.th32ProcessID;
			CloseHandle(hPID);		}
	} while (Process32Next(hPID, &ProcEntry));
	char myDLL[MAX_PATH];
	GetFullPathName("cheat.dll", MAX_PATH, myDLL, 0);
	HANDLE hProcess = OpenProcess(PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION
| PROCESS_VM_READ | PROCESS_VM_WRITE | PROCESS_VM_OPERATION, FALSE, dwPID);
	LPVOID allocatedMem = VirtualAllocEx(hProcess, NULL, sizeof(myDLL), MEM_RESERVE
| MEM_COMMIT, PAGE_READWRITE);
	WriteProcessMemory(hProcess, allocatedMem, myDLL, sizeof(myDLL), NULL);
	CreateRemoteThread(hProcess, 0, 0, (LPTHREAD_START_ROUTINE)LoadLibrary,
allocatedMem, 0, 0);
	CloseHandle(hProcess);
	return 0;	}
```

[6] Handle v OS Windows je abstraktní reference na objekt pro práci s Windows API. Využitím handlu zůstává skutečná adresa objektu skryta.

2.4 Hookování funkcí

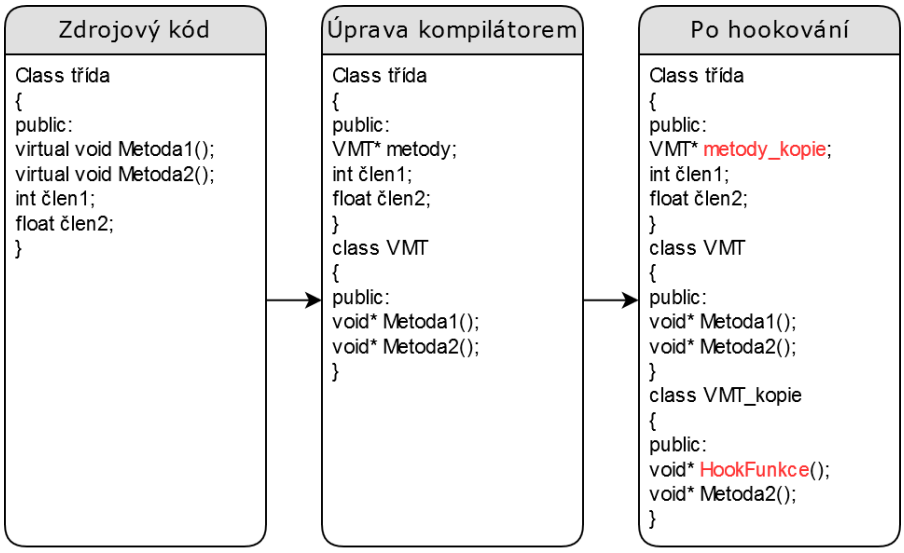
Hookování funkcí je nejpoužívanější metodou pro modifikaci běhu hry v interních cheatech. Rozumí se tím nahrazení funkcionality již existující funkce vlastním kódem. To je velmi silný nástroj, jelikož poskytuje přístup k parametrům původní funkce. Například funkce *CreateMove* je volaná každý tick[7] a skrze ukazatel na třídu *UserCmd* (user command) jako parametr posílá serveru povel k provedení akcí vyvolaných hráčem (pohyb, úhel pohledu, stisknuté klávesy atd.) Funkce cheatu která například asistuje při míření upravuje vektor *viewangles* (úhly pohledu), který je součástí *UserCmd* a proto je volána z hooknuté funkce *CreateMove*, zatímco funkce která kreslí na obrazovku (ve 2D) je volána z *PaintTraverse*. Veškeré funkce cheatu jsou volány skrze hooknuté funkce. Výhodou toho také je, že běh hry je v jejích rukou což přispívá k celkové optimalizaci cheatu. Nejjednodušší možností, jak hookovat funkci, je přepsat její instrukce v paměti. Knihovna Detours[8] od společnosti Microsoft, využívající tzv. Inline hooking, přepíše prvních několik bytů původní funkce nepodmíněným skokem na novou funkci. Přepsané byty se uloží do tzv. *trampoline* funkce. Na konci hook funkce se volá *trampoline* funkce a poté adresa původní funkce povýšená o počet přepsaných bytů.

2.4.1 VMT hooking

Přepisování instrukcí funkce mění její checksum a je tedy lehce detekovatelné. Cheat proto používá jinou metodu, tzv. Virtual Method Table hooking (VMT hooking). Implementace VMT se může podle kompilátoru lišit, Microsoft Visual C++ (MSVC) přidá na začátek třídy skrytou proměnnou, která ukazuje na Virtual Method Table. VMT je seznam ukazatelů na všechny virtuální metody definované ve třídě. Při spuštění programu jsou ukazatele nastaveny tak, aby ukazovali na správné metody. Každá má svůj index (podle pořadí ukazatele ve VMT), seřazeny jsou tak, jak jsou definovány ve třídě.

Cheat duplikuje VMT instance třídy a ukazatel na původní nahradí svým vlastním. Uvnitř nového VMT pak upraví ukazatel na hookovanou metodu, aby ukazoval na novou hook funkci. Původní metoda i VMT zůstávají tedy netknuté.

[7] Tick je akce kdy dojde k synchronizování klienta se serverem. V TF2 tato akce probíhá 66 krát za sekundu.
[8] Jedná se pouze o zdrojový kód, zdarma dostupný z: https://github.com/Microsoft/Detours



Obrázek 3: VMT Hooking

Hook funkce musí přijímat stejný počet argumentů jako původní funkce a musí používat stejnou volací konvenci. Funkce které cheat hookuje používají konvenci *thiscall* (v MSVC výchozí konvence pro členské funkce tříd), která přes ECX registr posílá *this* ukazatel na objekt (třídu) pod kterou volaná funkce patří (7). MSVC ale nepovoluje explicitně definovat funkce jako *thiscall*. S *thiscall* je kompatibilní *fastcall*, ta ale první dva argumenty posílá přes ECX a EDX registry a až ostatní ukládá na stack (zásobník), je tedy potřeba deklarovat parametr pro EDX i když se nepoužívá, aby se program nepokoušel číst první skutečný argument z EDX registru místo stacku.

```
bool __fastcall HookCreateMove(PVOID ClientMode, int edx, float
input_sample_frametime, CUserCmd* pCommand)
//PVOID ClientMode - this ukazatel v registru ECX, ukazuje na instanci třídy
ClientModeShared jejíž členem je funce CreateMove
//int edx - deklarace pro neexistující parametr v registru EDX
//argumenty 3 a 4 jsou ve skutečnosti argumenty 1 a 2 které funkce CreateMove přijímá
```

Další možností je konvence *stdcall*, která posílá argumenty pouze přes stack. *This* ukazatel, který *thiscall* posílá přes ECX registr, můžeme získat přes assembly kód:

```
void __stdcall HookDrawModelExecute(const DrawModelState_t &state, const
ModelRenderInfo_t &pInfo, matrix3x4 *pCustomBoneToWorld)
{
	DWORD thisptr;
	_asm mov thisptr, ECX //zkopíruje hodnotu v registru ECX do thisptr
}
```

Pro zjednodušení hookování využívá Cheat knihovnu VMThooks[9], zveřejněnou jako public domain. Následující kód hookuje funkci *KeyEvent* instance třídy *CHLClient*:

```
VMTBaseManager* clientHook = new VMTBaseManager();
```

[9] Soubory vmthooks.cpp a vmthooks.h, dostupné z: https://github.com/CasualX/LibEx/tree/master/public/toolkit

```
clientHook->Init(gInts.Client); //VMT které třídy bude hookováno? (Client - instance
třídy CHLClient)
clientHook->HookMethod(&HookKeyEvent, 20);  //určení adresy naší funkce a offset
(index) původní funkce ve VMT
clientHook->Rehook(); //hookuje funkci
```

Někde (ideálně uvnitř naší hook funkce) je potřeba zavolat původní funkci:

```
int __fastcall HookKeyEvent(PVOID CHLClient, int edx, int eventcode, int keynum, const
char *currentBinding)
{
	/*
	kód hook funkce
	*/

//získání adresy původního VMT instance třídy
VMTManager &hook = VMTManager::GetHook(CHLClient);
//volání původní funkce, hook funkce pak vrací hodnotu kterou vrátila původní funkce
return hook.GetMethod<int(__thiscall *)(PVOID, int, int, const char *)>(20)(CHLClient,
eventcode, keynum, currentBinding);
}
```

2.4.2 Window procedure

Každé okno v OS Windows registrované skrze Windows API je asociováno s vlastní callback funkcí *WindowProc*, která zpracovává všechny zprávy poslané oknu (stisk klávesy, pohyb kolečka myši apod.). Cheat tuto funkci využívá pro řešení inputu ve vlastním menu.

Funkce Windows API *SetWindowLongPtr* umožňuje upravit adresu *WindowProc* funkce specifického okna. Cheat nejprve vytvoří handle na okno hry pomocí funkce Windows API *FindWindow* podle jeho názvu (název okna se dá najít například v programu Process Hacker 2[10]) a poté nahradí adresu *WindowProc* funkce na adresu vlastní funkce *HookWindowProcedure*.

```
HWND thisWindow;
while (!(thisWindow = FindWindow("Valve001", NULL))) Sleep(500);
gMenu.windowProc = (WNDPROC)SetWindowLongPtr(thisWindow, GWLP_WNDPROC,
(LONG_PTR)&HookWindowProcedure);
```

HookWindowProcedure na konci volá původní funkci (pokud nebyla stisknuta klávesa Escape, aby nepůsobila na hru, když je otevřené menu cheatu.)

[10] Process Hacker 2 je open-source program, zdarma dostupný z: https://processhacker.sourceforge.io/

```
LRESULT __stdcall HookWindowProcedure(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
	switch (uMsg)
	{
	/*zkráceno*/
	case WM_KEYDOWN: // stisknutá klávesa
	case WM_SYSKEYDOWN: // alt + klávesa
		if (wParam > 255)
			break;
		gMenu.keys[wParam] = true, gMenu.last_key = wParam = gMenu.key = wParam;
		break;
	/*zkráceno*/
	}
// pokud je menu otevřené a byla stisknuta klávesy escape, nevolá původní funkci
if (gMenu.enabled && gMenu.keys[VK_ESCAPE])
		return 0;
return CallWindowProc(gMenu.windowProc, hWnd, uMsg, wParam, lParam);
}
```

2.5 Source engine

Source engine (vyvíjený také společností Valve), který Team Fortress 2 používá, je rozdělen na několik nezávislých rozhraní (renderování, herní entity, fyzika, zvuk atd.), které mezi sebou komunikují. Tato rozhraní jsou součástí různých DLL souborů. Každý takový DLL soubor má exportovanou funkci *CreateInterface*, která vrací ukazatel na tato rozhraní, stačí znát jeho název.

Jelikož se jedná o exportovanou funkci, není třeba hledat její adresu, ale lze využít funkce Windows API *GetProcAdress*. Ta jako argument přijímá handle na DLL modul (získatelný přes další funkci Windows API *GetModuleHandle*) a název exportované funkce. Získání adresy funkce *CreateInterface* která je součástí souboru client.dll pak vypadá takto:

```
ClientFactory =
(CreateInterfaceFn)GetProcAddress(gSignatures.GetModuleHandleSafe("client.dll"),
"CreateInterface");
```

Nyní můžeme skrze *CreateInterface* získat ukazatel na některé z rozhraní Source enginu:

```
gInts.Client = (CHLClient*)ClientFactory("VClient017", NULL);
```

Základ celého cheatu spočívá v hookování funkcí těchto rozhraní.

Source engine má také k dispozici SDK (vývojářský kit) určený pro tvorbu modů (modifikací hry) nebo nových her. Source SDK[11] a jeho Wiki stránka[12] rozšiřuje možnosti cheatů, jelikož zjednodušuje reverzování a používání Source funkcí v cheatech.

[11] Zdrojový kód nejnovější verze Source SDK je dostupný z: https://github.com/ValveSoftware/source-sdk-2013
[12] Wiki stránka Source enginu a jeho SDK je dostupná z: https://developer.valvesoftware.com/wiki/SDK_Docs

2.6 Běh cheatu od injektování po hookování herních funkcí

Následující zjednodušený výčet a zdrojový kód popisují běh programu cheatu od momentu kdy funkce *LoadLibrary* zavolá vstupní funkci *DLLMain* v injektnutém cheat.dll až po hookování herních funkcí.

1) *LoadLibrary* volá funkci *DLLMain*, ta v procesu vytvoří nové vlákno *dwMainThread*

```
BOOL APIENTRY DllMain(HMODULE hInstance, DWORD dwReason, LPVOID lpReserved)
{
if(dwReason == DLL_PROCESS_ATTACH)
{
CreateThread( NULL, 0, (LPTHREAD_START_ROUTINE)dwMainThread, NULL, 0, NULL );
}
return true;
}
```

2) Cheat získá adresy exportovaných funkcí *CreateInterface* pomocí *GetProcAdress*

```
DWORD WINAPI dwMainThread(LPVOID lpArguments)
{
ClientFactory =
(CreateInterfaceFn)GetProcAddress(gSignatures.GetModuleHandleSafe("client.dll"),"Creat
eInterface");
ClientFactory = /*zkráceno*/ ("client.dll"), "CreateInterface");
EngineFactory = /*zkráceno*/ ("engine.dll"), "CreateInterface");
VGUIFactory = /*zkráceno*/ ("vguimatsurface.dll"), "CreateInterface");
VGUI2Factory = /*zkráceno*/ ("vgui2.dll"), "CreateInterface");
CvarFactory = /*zkráceno*/ ("vstdlib.dll"), "CreateInterface");
SteamFactory = /*zkráceno*/ ("SteamClient.dll"), "CreateInterface");
MaterialSystemFactory = /*zkráceno*/ ("MaterialSystem.dll"), "CreateInterface");
```

3) Cheat získá ukazatele na následující rozhraní Source enginu pomocí funkcí *CreateInterface*

```
gInts.Client = (CHLClient*)ClientFactory("VClient017", NULL);
gInts.EntList = (CEntList*)ClientFactory("VClientEntityList003", NULL);
gInts.EntList2 = (CEntList2*)ClientFactory("VClientEntityList003", NULL);
gInts.Engine = (EngineClient*)EngineFactory("VEngineClient013", NULL);
gInts.Surface = (ISurface*)VGUIFactory("VGUI_Surface030", NULL);
gInts.RenderView = (CRenderView*)EngineFactory("VEngineRenderView014", NULL);
gInts.MatSystem = (CMaterialSystem*)MaterialSystemFactory("VMaterialSystem081", NULL);
gInts.EngineTrace = (IEngineTrace*)EngineFactory("EngineTraceClient003", NULL);
gInts.ModelInfo = (IVModelInfo*)EngineFactory("VModelInfoClient006", NULL);
gInts.cvar = (ICvar*)CvarFactory("VEngineCvar004", NULL);
gInts.ModelRender = (CModelRender*)EngineFactory("VEngineModel016", NULL);
gInts.EventManager = (IGameEventManager2*)EngineFactory("GAMEEVENTSMANAGER002", NULL);
gInts.Panels = (IPanel*)VGUI2Factory("VGUI_Panel009", NULL);
```

4) Cheat získá ukazatele na následující rozhraní přes vlastní bytové signatury

```
gInts.globals = *reinterpret_cast<CGlobals**>(gSignatures.GetEngineSignature("A1 ? ? ?
? 8B 11 68") + 8); //engine.dll
DWORD dwClientModeAddress = gSignatures.GetClientSignature("8B 0D ? ? ? ? 8B 02 D9
05"); //client.dll
gInts.ClientMode = **(ClientModeShared***)(dwClientModeAddress + 2);
```

5) Cheat hookuje VMT následujících funkcí, pomocí knihovny VMThooks

```
VMTBaseManager* clientHook = new VMTBaseManager();
VMTBaseManager* clientModeHook = new VMTBaseManager();
VMTBaseManager* panelHook = new VMTBaseManager();
VMTBaseManager* defRenderHook = new VMTBaseManager();
//PaintTraverse
panelHook->Init(gInts.Panels);
panelHook->HookMethod(&HookPaintTraverse,gOffsets.iPaintTraverseOffset); //41
panelHook->Rehook();
//KeyEvent
clientHook->Init(gInts.Client);
clientHook->HookMethod(&HookKeyEvent, gOffsets.iKeyEventOffset); //20
clientHook->Rehook();
//FrameStageNotifyThink
clientHook->HookMethod(&FrameStageNotifyThink, 35);
clientHook->Rehook();
//CreateMove
clientModeHook->Init(gInts.ClientMode);
clientModeHook->HookMethod(&HookCreateMove, gOffsets.iCreateMoveOffset); //21
clientModeHook->Rehook();
//DrawModelExecute
defRenderHook->Init(gInts.ModelRender);
defRenderHook->HookMethod(&HookDrawModelExecute, 19);
defRenderHook->Rehook();
```

6) Nakonec hookuje Window procedure okna hry pro práci s menu

```
HWND thisWindow;
while (!(thisWindow = FindWindow("Valve001", NULL)))
	Sleep(500);
gMenu.windowProc = (WNDPROC)SetWindowLongPtr(thisWindow, GWLP_WNDPROC,
(LONG_PTR)&HookWindowProcedure);
```

Z výše hooknutých funkcí jsou poté volány samotné funkce cheatu.

2.7 Používání funkcí Source enginu

Jednou z výhod interních cheatů je možnost přímo volat funkce samotné hry, v tomto případě funkce jednotlivých rozhraní Source enginu. Adresy tříd do kterých funkce patří cheat získává skrze *CreateInterface*, nebo na třídu přijímá ukazatel některá z hooknutých funkcí. Reverzním inženýrstvím je zjištěn index funkce ve VMT třídy, datový typ který funkce vrací, její parametry a potvrzena volací konvence *thiscall* (až na výjimky). Reverzované funkce, struktury a třídy Source enginu jsou definované v souboru SDK.h ve zdrojovém kódu cheatu, jako například funkce *DrawSetColor* rozhraní *ISurface*:

```
class ISurface
{
public:
	void DrawSetColor(int r, int g, int b, int a)
	{
		typedef void(__thiscall* OriginalFn)(void*, int, int, int, int);
		getvfunc<OriginalFn>(this, 11)(this, r, g, b, a);
	}
}
```

Když cheat zavolá funkci *DrawSetColor*, funkce *getvfunc* volá původní funkci, která má ve VMT *ISurface* index 11 a předává jí parametry, včetně přidaného ukazatele *this*, jelikož používá konvenci *thiscall*. *This* je ukazatel na instanci mateřské třídy funkce.

```
inline const void** getvtable( const void* inst, size_t offset = 0 )
{
	return *reinterpret_cast<const void***>( (size_t)inst + offset );
}
template< typename Fn >
inline Fn getvfunc( const void* inst, size_t index, size_t offset = 0 )
{
	return reinterpret_cast<Fn>( getvtable( inst, offset )[ index ] );
}
```

Getvtable vrací adresu VMT třídy za ukazatelem *this*, *getvfunc* adresu funkce.

2.7.1 NetVar a ConVar

NetVar (networked variable) jsou proměnné sdílené mezi klientem a serverem. Samotný NetVar mívá více členských proměnných. Například NetVar "DT_BasePlayer", třídy *CBaseEntity* obsahuje proměnnou "m_lifeState", jejíž hodnota je 0, pokud je entita živá, 1 pokud je neživá. Funkce *IsAlive*, kterou cheat používá pro zjištění, zda je daná hráčská postava živá, zjišťuje jestli se hodnota "m_lifeState" rovná nule a vrací logickou hodnotu tohoto porovnání.

```
bool IsAlive()
{
return *(byte*)(this + gNetVars.get_offset("DT_BasePlayer", "m_lifeState")) == 0;
}
```

Samotný NetVar může být dělen na další části a ty až můžou obsahovat členské proměnné. NetVar má tedy stromovou strukturu. Funkce Source enginu *GetAllClasses* vrací ukazatel na první třídu *ClientClass* obsahující tabulku NetVarů (včetně jejich názvů) a ukazatel na další třídu *ClientClass*. Cheat tak postupně vytvoří stromovou strukturu všech NetVarů. Funkce *get_offset* (viz výše) rekurzivně prochází strukturu NetVarů podle jejich názvů, než se dostane ke konečné členské proměnné, jejíž akumulovaný offset od počátku mateřské třídy vrátí. Funkce *GetValue* a *SetValue* čtou nebo přepisují hodnoty na adrese členské proměnné NetVaru.

ConVar je proměnná přístupná z vývojářské konzole ve hře. Hráči vývojářskou konzoli využívají nejčastěji pro změnu nastavení hry, protože změna hodnoty ConVaru může být mnohdy rychlejší než procházení menu hry. Konzole také poskytuje možnost jednoduchého skriptování. Některé ConVary změnit nelze, stejně tak některé příkazy skrze konzoli hra neprovede, protože by mohli hráči poskytnout výhodu. Jejich změna je povolená pouze pokud má server na kterém hráč hraje nastavený ConVar "sv_cheats" na hodnotu 1. Cheat ale může tento ConVar lokálně upravit pomocí funkcí Source enginu a obejít tak vývojářskou konzoli a nastavení serveru. Hra hodnoty ConVarů se serverem nesdílí. Toho lze využít k cheatování, čehož jsou důkazem některé funkce cheatu. Source funkce *FindVar* vrací ukazatel na hledaný ConVar podle jeho názvu. Funkce *SetValue* a *GetValue* nastavují nebo získávají hodnoty ConVaru.

3 FUNKCE CHEATU

3.1 Aimbot

Aimbot (složenina slov "aim" - mířit a "robot") asistuje nebo kompletně automatizuje míření hráče na nepřítele. Cheat nejprve najde nejvhodnější cíl a poté nejvhodnější část těla cíle, na kterou může zamířit a vypočítá vektor z pozice hráčových očí do pozice nejvhodnější části těla cíle, který poté převede na tři Eulerovy úhly a patřičně upraví úhly pohledu hráčovy postavy tak, aby mířila na cíl.

Pozice všech entit včetně těch hráčských je v Source enginu reprezentována 3-dimenzionálním vektorem s počátkem ve středu mapy (herního světa). Úhly pohledu hráče (*viewangles*) jsou reprezentovány třemi Eulerovými úhly: X - náměr, Y - vodorovné otočení a Z - náklon (ten je v TF2 vždy nulový).

Pro nalezení nejvhodnějšího cíle nejprve funkce cheatu *GetBestTarget* prochází ve for smyčce všechny hráčské entity a u každé kontroluje, zda je naživu, v nepřátelském týmu, není momentálně nesmrtelný a je zasažitelný (tedy zda není za překážkou a zbraň kterou postava lokálního hráče[13] drží je schopna ho zasáhnout).

```
int CAimbot::GetBestTarget(CBaseEntity* pLocal, CUserCmd* pCommand)
{
	for (int i = 1; i <= gInts.Engine->GetMaxClients(); i++)
	{
	//Smyčka ignoruje zbytek instrukcí a pokračuje znovu s inkrementovaným i, pokud
	platí  některá z následujících podmínek
	if (i == me) //aktuální index odpovídá indexu lokálního hráče
		continue;
	CBaseEntity* pEntity = GetBaseEntity(i);//(ukazatel na i-tou hráčskou entitu)
	if (!pEntity) //pEntity neukazuje na platnou entitu
		continue;
	if (pEntity->GetLifeState() != LIFE_ALIVE)
		continue; //pEntity neukazuje na živou entitu
	if (pEntity->GetTeamNum() == pLocal->GetTeamNum())
		continue; //pEntity ukazuje na hráče který je spoluhráč lokálního hráče
	int iBestHitbox = GetBestHitbox(pLocal, pEntity);
	if (iBestHitbox == -1)
		continue; //GetBestHitbox nevrátila žádnou část těla jako zasažitelnou
	if (pEntity->GetCond() & TFCond_Ubercharged ||
	    pEntity->GetCond() & TFCond_UberchargeFading ||
	    pEntity->GetCond() & TFCond_Bonked)
		continue; //hráč je v nesmrtelném stavu
```

Poté cheat vypočítá úhly od pozice očí hráčovy postavy do pozice ideální části těla pro zásah vrácené funkcí *GetBestHitbox* a vypočítá jejich odchylku od aktuálních úhlů pohledu (viewangles) hráčovy postavy. Pokud je odchylka menší než aktuálně uložená v proměnné *flDistToBest* z minulých iterací for smyčky, přepíše ji novou odchylkou a index aktuální hráčské entity (tedy hodnotu iterační proměnné ve for smyčce) uloží do proměnné *bestTarget*. Pokud existuje více zasažitelných hráčů, cheat tedy prioritizuje toho, který je nejblíže

[13] Pojmem lokální hráč se rozumí hráč který cheat používá.

k zaměřovači lokálního hráče. Po skončení for smyčky funkce *GetBestTarget* vrací hodnotu v *bestTarget*. V případě, že nebyl nalezen žádný je hodnota *bestTarget* -1.

Funkce *GetBestHitbox* preferuje zásah hlavy. Pokud není možné hlavu zasáhnout, postupuje k ostatním částem těla. Pro zjištění, zda je část těla zasažitelná používá funkci Source enginu *TraceRay* (funkce *TraceRay* je dále popsána v kapitole 3.2 Triggerbot).

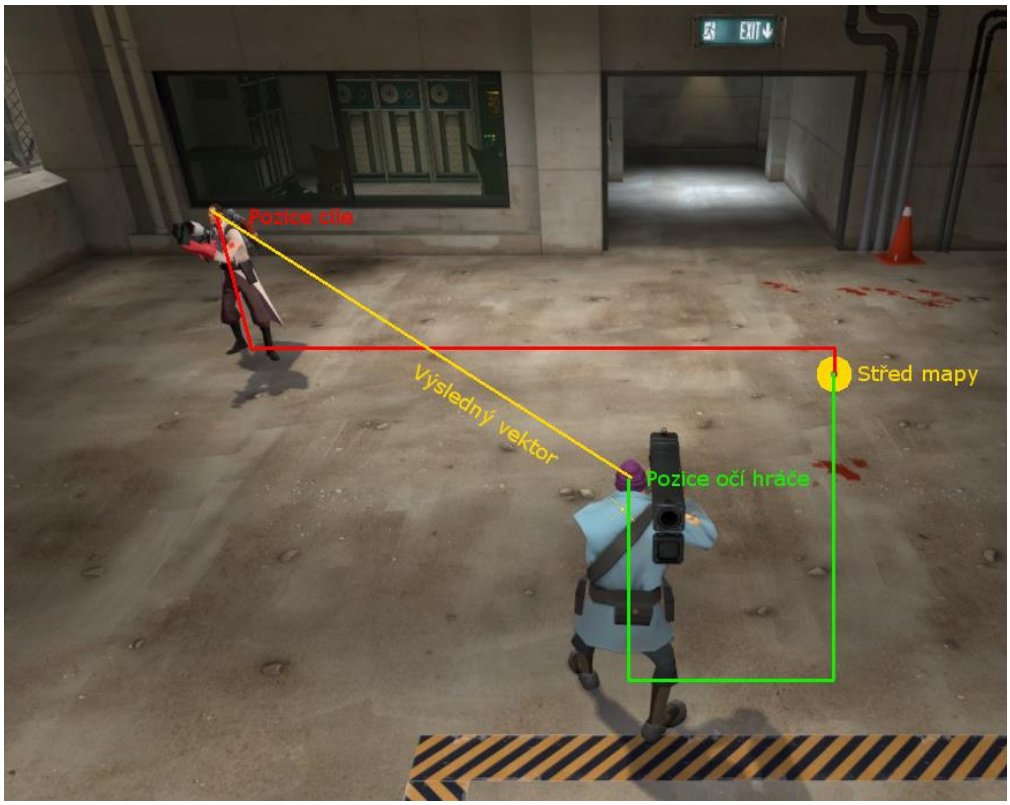
Následující rovnice (1.1, 1.2, 1.2, 1.4) popisují jak cheat vypočítá úhly *viewangles* (náměr x a vodorovné otočení y) ve stupních z vektoru *local* (pozice očí hráčovy postavy) do vektoru *target* (cíl). *Delta* je pozice cíle relativní k *local*. *Hyp* je přepona (délka) vektoru *delta*. Hodnoty funkcí arkus sinus a arkus tangens jsou převáděny z radiánů na stupně násobením zlomkem $\frac{180}{\pi}$.

$$\overline{delta} = \overline{target} - \overline{local} \tag{1.1}$$

$$hyp = \sqrt{(delta_x)^2 + \left(delta_y\right)^2} \tag{1.2}$$

$$viewangles_x = \arcsin(delta_z \div hyp) \cdot (\frac{180}{\pi}) \tag{1.3}$$

$$viewangles_y = \operatorname{arctg}\left(delta_y \div delta_x\right) \cdot (\frac{180}{\pi}) \tag{1.4}$$



Obrázek 4: Pozice hráčů a výsledný vektor aimbotu

Před nastavením nových *viewangles* ještě cheat úhly normalizuje, aby hodnoty byly v povolených mezích (-89° až +89° pro vertikální náměr a -180° až +180° pro horizontální otočení, náklon je vždy 0).

```
inline void NormalizeAngle(Vector& qaAng)
{
	while (qaAng.x > 89)
		qaAng.x -= 180;
	while (qaAng.x < -89)
		qaAng.x += 180;
	while (qaAng.y > 180)
		qaAng.y -= 360;
	while (qaAng.y < -180)
		qaAng.y += 360;
	while (qaAng.z != 0)
		qaAng.z = 0;
}
```

Funkce aimbotu je volána z *HookCreateMove*. Jedním z jejích parametrů je ukazatel na třídu *CUserCmd* (User command). Členem této třídy je právě vektor *viewangles*, který cheat upraví na vypočítané hodnoty. Výsledným efektem tedy je, že cheat instantně zamíří na nejbližší hráčskou postavu (ideálně na její hlavu) a bude na ni neustále mířit dokud nezemře nebo už nebude zasažitelná. Cheat také upravuje pohyb (chůzi) hráče, aby upravené vodorovné otočení nemělo vliv na směr pohybu. Pohyb upravuje přes členské proměnné třídy *CUserCmd forwardmove*, *sidemove* a *upmove*.

3.1.1 Skrytí používání aimbotu

Výše popsané fungování aimbotu je sice ideální pro rage cheatování, ale je velmi nápadné. Cheat poskytuje několik nastavení pro snížení šance, že jeho chování bude ostatním hráčům podezřelé.

- "Aim key" - aimbot bude aktivní jen když hráč drží nastavenou klávesu (nebo tlačítko myši)
- "Hitbox" (head, body, all) - nastavení body donutí aimbot mířit na tělo, nikoli hlavu
- "FOV" (1°-180°) - Aimbot bude mířit jen na hráče dostatečně blízko k zaměřovači podle nastaveného zorného pole ve stupních (field of view)
- "Ignore cloaked" - kromě nesmrtelnosti bude kontrolovat i neviditelnost cíle
- "Ignore disguised" - aimbot bude ignorovat nepřátelské hráče přestrojené za spoluhráče
- "Smooth"(0 -30) - umožňuje nastavit jak plynule bude aimbot měnit *viewangles*. Vyšší hodnota znamená pomalejší zamíření.

  ```
  //vDelta = aktuální - vypočítané vieawngles (vAngs)
  Vector vDelta(pCommand->viewangles - vAngs);
  AngleNormalize(vDelta); //normalizuje vDelta
  //nové vAngs = aktuální - (vDelta dělená nastaveným faktorem smooth)
  vAngs = pCommand->viewangles - vDelta / smooth.value;
  ```

- Silent - *viewangles* upraví jen při výstřelu. Fungování aimbotu tak může být viditelné jen zlomek vteřiny.

Nutno podotknout, že *viewangles* v *CUserCmd* a úhel pohledu, který je renderován na hráčův monitor, není totéž. Když je zapnuté nastavení Silent, aimbot tyto lokální úhly nemění. Z pohledu hráče tak k žádné změně *viewangles* nedojde, ale střela přesto zasáhne cíl.

Cheat má také nastavení "Autoshoot", které určuje, aby aimbot po zamíření i automaticky vystřelil a "Autozoom", které automaticky přiblíží pohled. Mechanika přiblížení pohledu je jen u zbraně "Sniper rifle". Po přiblížení je poškození, které zbraň udílí, postupně zvyšováno až na trojnásobek. Nastavení "Wait for charge" porovnává HP (health points) cíle s aktuálním poškozením zbraně a nedovolí, aby Autoshoot vystřelil, pokud by rána nebyla smrtící.

```
if (pLocal->szGetClass() == "Sniper" && pLocal->GetActiveWeapon()->GetSlot() == 0)
{
float damage = pLocal->GetActiveWeapon()->GetChargeDamage();
if ((autozoom.value) && (!(pLocal->GetCond() & tf_cond::TFCond_Zoomed)))
	{
		pCommand->buttons |= IN_ATTACK2;
	}
if (zoomedonly.value)
	{
		if (!(pLocal->GetCond() & tf_cond::TFCond_Zoomed))
			return -1;
		if (damage < 10.f)
			return -1;
	}
if (waitforcharge.value)
	{
	if (waitforcharge.value && BASE_DMG + damage < pEntity->GetHealth()) return -1;
	}
}
```

3.1.2 Projektilový aimbot

Výše zmíněný aimbot funguje pouze pro tzv. hitscan zbraně, které ihned po výstřelu udělí poškození cíli v jejich trajektorii. Jinými slovy má kulka vystřelená z hitscan zbraně nekonečnou velkou rychlost. V TF2 jsou tyto střely emitovány z hráčova zaměřovače (tedy středu monitoru), který odpovídá pozici očí hráčovy postavy. Některé zbraně v TF2 jsou ale tzv. projektilové (vystřelují fyzické projektily). Projektily se na rozdíl od hitscan kulek nepohybují instantně, ale mají předdefinovanou rychlost a na některé působí gravitace. Aimbot tedy musí započítat rychlost a gravitaci projektilu a pohyb nepřítele - předpovídá, kde se bude nacházet za čas, který uběhne, než projektil urazí cestu od hráče k nepříteli. Pokud je nepřítel ve vzduchu tak také zjišťuje, zda jeho je předpovězená pozice pod úrovní pevného povrchu pod ním. Pokud ano, tak předpovězenou výškovou polohu upraví aby se aimbot nesnažil mířit pod zem nebo podlaží. K získání rychlosti a směru pohybu nepřítele používá cheat funkci *EstimateAbsVelocity*. Vzdálenost k prvnímu pevnému objektu pod nepřítelem vrací funkce *TraceRay* (funkce *TraceRay* je dále popsána v následující kapitole).

3.2 Triggerbot

Triggerbot nechává míření na hráči, avšak automaticky vystřelí, když zjistí, že by střela zasáhla cíl. Používání triggerbotu může být oproti aimbotu velmi nenápadné. Opět se dá nastavit, aby se aktivoval pouze při stisku klávesy, ignoroval neviditelné hráče a spoluhráče a také aby střílel pouze na hlavu. Triggerbot v tomto cheatu nepracuje s projektily, protože využívá funkce Source enginu *TraceRay*, která vysílá paprsek z jednoho bodu do druhého a vrací první věc do které narazí (podle nastavení bitové masky, viz níže). Cheat vyšle paprsek z pozice hráčových očí, ve směru jeho úhlu pohledu. Pokud paprsek trefí entitu hráče, znamená to, že na něj hráč míří a vystřelí.

Funkce *TraceRay* vyžaduje mimo jiné argument určující, co může paprsek trefit. Je ve formátu bitové masky, proto cheat může použít více nastavení současně. Cheat používá takovou masku, aby paprsek trefil vše, co by ve hře vystřelená kulka mohla zasáhnout (musí být vyloučen zásah např. abstraktních součástí scény)

```
#define CONTENTS_SOLID            0x1 //pevné objekty, stěny, zem...
#define CONTENTS_WINDOW           0x2 //průhledné, pevné objekty (okna, sklo)
#define CONTENTS_MOVEABLE      0x4000 //entity typu MOVETYPE_PUSH (dveře)
#define CONTENTS_DEBRIS     0x4000000 //3d sutiny, trosky
#define CONTENTS_MONSTER    0x2000000 //nehráčské postavy (NPC)
#define CONTENTS_HITBOX    0x40000000 //hitboxy hráčů
//výsledná maska:          0x46004003
```

Triggerbot má také funkci Auto Backstab. Jedna z herních postav, Spy, dokáže svým nožem zabít nepřítele jedinou ranou, pokud ho zasáhne do zad - tzv. backstab. Cheat volá funkci *CanBackStab*, která vrací logickou hodnotu NetVaru "m_bReadyToBackstab", aby zjistil, zda může backstab provést.

Zdrojový kód funkce Triggerbot je nejkratší ze všech funkcí cheatu a byl proto jako ukázkový kód okomentován a jeho kompletní podoba je dostupná v Příloze 1.

3.3 Anti-aim

Anti-aim upravuje *viewangles* hráče tak, aby snížil šanci, že bude hráč zasažen. Mění *viewangles* na falešné hodnoty podle nastavení cheatu a pouze během střelby je rychle upraví na pravé hodnoty, aby střela zasáhla cíl a nikdy neupravuje pohled renderovaný na monitor hráče (podobně jako Aimbot se zapnutou Silent funkcí). Cheat kromě jednoduchého pohledu nahoru nebo dolů a počtu stupňů otočení doprava / doleva umožňuje nastavit horizontální otočení na "random", kdy každý tick je vygenerován jiný úhel, nebo "spin", tedy nepřetržité točení dokola s nastavitelnou rychlostí. To samo o sobě velmi ztěžuje zasažení hlavy nebo jiné části těla, ale šikovným nastavením Anti-aimu lze některé postavy natočit tak, že je hlava schovaná za zbytkem těla, nebo dokonce využít rohu zdi a schovat téměř celý model hráčovy postavy a přitom mít výhled na nepřítele. Anti-aim také upravuje pohyb hráče, aby kompenzoval otočení pohledu a hráč tak například místo dopředu nešel do strany, protože nastavil Anti-aim na "right".

Zajímavější možností Anti-aimu pak jsou nastavení "Fake up" a "Fake down", které využívají chyby vývojářů při ošetření úhlů mimo povolený rozsah, který je -89° (dolů) až +89° (nahoru). Pokud serveru pošleme úhel nižší než -89°, upraví hodnotu na nejnižší povolenou, tedy -89° a postava ve hře bude zobrazena, že míří do země, ale pro hitbox (neviditelný model používaný pro vyhodnocování zásahů) toto ignoruje a s nižším úhlem hodnota "přeteče" a hitbox tak naopak míří stále výš. To tedy způsobí, že renderovaný model a hitbox, který rozhoduje o tom, zda byl hráč skutečně zasažen, jsou rozdílné. Například hodnota -271°, kterou pro viditelný model server upraví na -89° (přímo dolů) otočí hitbox vzhůru. U některých postav se i přesto hitbox s renderovaným modelem částečně překrývají, v některých případech ale může hráč dosáhnout skutečné "nezasažitelnosti" (viz **Chyba! Nenalezen zdroj odkazů.**) nebo alespoň zamezit zásahu hlavy.

Anti-aim je velmi nápadný a využívá se tak zejména při rage cheatování nebo na HvH (hack vs hack) serverech, kde cheateři bojují proti sobě a vyhrává lépe naprogramovaný cheat.

Obrázek 5: Rozdíl mezi vykreslovaným modelem a hitboxem (modré kvádry)

3.4 Speedhack

Viewangles v Source enginu mají i třetí úhel "Z", určující náklon (roll). V TF2 se sice náklon nevyužívá, ale experimentování s upravováním tohoto úhlu přineslo zajímavé výsledky. Při pohybu vzad (který je normálně pomalejší než pohyb vpřed) se v závislosti na úhlech pohledu hráče někdy zvyšovala rychlost pohybu a měnil směr chůze.

Po hlubším zkoumání zdrojového kódu Source enginu a dalším experimentování bylo dosaženo následujících závěrů:

Hra při převádění *viewangles* na vektory směru chůze *vForward* a *vRight* započítává i hodnotu náklonu (8). Server omezuje rychlost pohybu na danou maximální hodnotu *flMaxSpeed*, která odpovídá normální rychlosti chůze vpřed, ale pro pohyb vzad (tedy záporné hodnoty *CUserCmd->forwardmove*) aplikuje stejný limit. Pokud je postava skrčená (to chůzi značně zpomalí), je aplikován jiný limit rychlosti, avšak pouze pro chůzi dopředu. Skrčená chůze vzad je (chybně) limitována na původní *flMaxSpeed* platící pro vzpřímenou chůzi vpřed. Nelze tedy zrychlit obyčejnou chůzi vpřed nad limit aplikovaný serverem, ale je možné zrychlit chůzi vzad na rychlost vzpřímené chůze vpřed a to i během skrčení postavy, což už vytváří značnou rychlostní výhodu oproti normálnímu skrčenému pohybu.

Úpravou náklonu na 90° se směr chůze hráčovy postavy nemění, ale stále platí zrychlení pohybu. Zrychlení pohybu vzad není příliš praktické, ale horizontálním otočením postavy o 180° ($viewangles_y$) a úpravou *forwardmove* na číslo opačné sice postava "couvá", avšak směr pohybu je stejný jako kdyby šla dopředu. Výsledná funkce je tedy funkční speedhack pro skrčený pohyb. Pokud hráč právě střílí, cheat žádné hodnoty neupravuje, aby střela nemířila za hráče (podobně jako Anti-aim).

```
if (speedhack.value && !(pCommand->buttons & IN_ATTACK) && (pCommand->buttons &
IN_DUCK)) //kontroluje zda je funkce zapnutá, hráč nestřílí a je skrčený
{
	Vector vLocalAngles = pCommand->viewangles; //uloží aktuální viewangles
	float speed = pCommand->forwardmove; //speed = hodnota forwardmove
	if (fabs(speed) > 0.0f) { //kontroluje zda se hráč pohybuje vpřed (speed > 0)
		pCommand->forwardmove = -speed; //forwardmove upraví na číslo opačné,
		tedy pohyb dozadu
		pCommand->sidemove = 0.0f; //anuluje sidemove (pohyb do strany)
		pCommand->viewangles.y = vLocalAngles.y;
		pCommand->viewangles.y -= 180.0f; //otočí postavu hráče o 180°, aby
		couvala dopředu
		//normalizuje úhel otočení
		if (pCommand->viewangles.y < -180.0f) pCommand->viewangles.y += 360.0f;
		pCommand->viewangles.z = 90.0f; //nastaví náklon na 90°
	}
}
```

Jak již bylo zmíněno v úvodu práce, je vzácné aby se pro novodobou online hru podařilo vytvořit funkční speedhack. Pro Team Fortress 2 momentálně žádný jiný (známý) speedhack neexistuje.

3.5 Auto-airblast

Jedna z herních postav ("Pyro") dokáže alternativním útokem svého plamenometu "odfouknout" stlačeným vzduchem nepřátelské projektily a poslat je opačným směrem (resp. tam kam hráč míří). Tato akce se nazývá *airblast*. Je to instantně provedená akce, která má velmi omezený dosah, a proto vyžaduje správné načasování a rychlé reflexy.

Auto-airblast ve smyčce projde všechny entity z *CEntList*, u každé zjišťuje, zda se jedná o objekt který lze odfouknout (podle názvu entity), jestli ho vystřelil nepřítel a je v dosahu *airblastu* (185 hu[14]). Pokud takový objekt existuje, přidá do aktuálního *UserCmd->buttons* operátorem *bitwise or* "IN_ATTACK2" (0x800) pro provedení alternativního útoku a odfouknutí projektilu. Server ale u *airblastu* nekompenzuje zpoždění mezi serverem a hráčem a cheat tak musí započítávat kromě rychlosti projektilu také příchozí a odchozí latenci:

```
// získání ukazatele na INetChannelInfo obsahující informace o připojení k serveru
auto net = reinterpret_cast<INetChannelInfo*>(gInts.Engine->GetNetChannelInfo());
// získání aktuálního spoždení mezi serverem a hráčem (odchozí + příchozí latence)
float latency = net->GetLatency(FLOW_OUTGOING) + net->GetLatency(FLOW_INCOMING);
// uložení rychlosti projektilu do vektoru
Vector velocity;
velocity = EstimateAbsVelocity(ent);
// předpoví, kde se bude projektil nacházet; k absolutní (relativní ke světu)
// pozici projektilu přičte vektor rychlosti skalárně násobený latencí
Vector predicted_proj = ent->GetAbsOrigin() + (velocity * latency);
// vypočítá absolutní vzdálenost od hráče k předpovězené pozici projektilu
float dist = predicted_proj.DistToSqr(g_pLocalPlayer->GetAbsOrigin());
```

3.6 Auto-Sticky detonation

Jiná z postav, Demoman, může vystřelovat lepivé bomby (tzv. sticky bomby), které se přilepí na jakýkoli povrch a hráč je může kdykoli odpálit. Dají se tak vytvářet pasti nebo je lze jen rozhazovat na nepřátele a ihned odpalovat (tzv. "sticky spam"). Funkce cheatu Auto-Sticky detonation, jak název napovídá, automaticky odpálí vystřelené bomby, pokud je alespoň jedna (z 8 max. položených) v dostatečné blízkosti od nepřítele, aby jej poškodila. Tím tak tvoří ze sticky bomb miny, které samy reagují na přítomnost soupeře. Hráč se tak musí soustředit jen na rozhazování bomb do blízkosti nepřátel nebo může připravit past, která se sama odpálí i když on bude na druhé straně mapy.

Cheat nejprve ve for smyčce prochází všechny entity z *CEntList* rozhraní a pokud narazí na sticky bombu, která byla vytvořena lokálním hráčem, hledá v další for smyčce hráče, který je v nepřátelském týmu, není neviditelný nebo nesmrtelný a je v dostatečné blízkosti k bombě (<144 *hu*). Nakonec ještě vysílá paprsek (*TraceRay*) od bomby ke hráči. Pokud paprsek nenarazí na nic, co by blokovalo poškození od bomby, do *UserCmd->buttons* opět přidá IN_ATTACK2 pro alternativní útok, čímž všechny položené bomby odpálí.

---

[14] Jednotkou vzdálenosti je v Source enginu *hammer unit* (hu), 1 hu odpovídá zhruba 1,9 cm

3.7 ESP

Pojmem ESP (extrasensory perception, nebo jednodušeji sixth sense - šestý smysl) bývají v cheaterské komunitě označovány funkce které hráči poskytují jinak nepřístupné informace. Funkce ESP prochází ve for smyčce všechny hráče a podle nastavení cheatu kreslí na obrazovku zjištěné informace o nich. Cheat nabízí mimo jiné následující nastavení:

- Box - kolem postav kreslí obdélníky, čímž odhaluje jejich pozici i když nejsou viditelní.
- Name - píše vedle postavy jméno hráče
- Class - píše vedle postavy její název (v TF2 je celkem 9 postav, také zvaných "třídy")
- Status - píše vedle postavy všechny její aktuální "stavy" (neviditelnost, nesmrtelnost apod.)
- Health (bar, value, both) - zobrazuje zdraví (životy) postavy jako text nebo barevný sloupec
- Objects - kreslí box a dodatečné informace také kolem objektů postavených hráči, beden s municí a lékarniček
- Bones - vykresluje "kosti" postav
- Weapon - píše vedle postavy název její aktuálně držené zbraně

Po zhotovení funkce Anti-aim (resp. Fake anti-aim) bylo také vytvořeno jednoduché opatření pro zjištění pozice hlavy skutečného modelu (hitboxu). Nastavení ESP "viewlines" vykresluje správný vektor úhlů pohledu (viewangles) z korektní pozice očí. Cheat pro zjištění správných informací čte NetVary "m_vecViewOffset" a "m_angEyeAngles", které uchovávají hodnoty pro hitbox, nikoli renderovaný model.

Informace cheat získává skrze funkce Source enginu nebo čtením hodnot NetVarů. Za samotné kreslení jsou zodpovědné Source funkce z rozhraní *ISurface* jako *drawString*, *drawRectangle* apod. Před kreslením je však třeba přenést pozici postavy ve 3D světě na 2D monitor hráče. Cheat používá matici poskytovanou grafickým API Direct3D 9 pro přepočet 3D koordinátů na 2D obrazovku hráče, závisle na aktuálních úhlech pohledu. Více viz online dokumentace Direct3D 9 (9) a (10).

Obrázek 6: Funkce ESP s nastavením Box, Bones, Name, Class, Health (both) a Weapon

3.7.1 Radar

Funkce radar vykresluje do rohu obrazovky čtvercové pole s tečkami reprezentujícími ostatní hráče. Lokální hráč je vždy uprostřed a pozice spoluhráčů i nepřátel jsou přepočítávány na pozice relativní k lokálnímu hráči. Radar je orientován podle aktuálního vodorovného otočení postavy lokálního hráče (resp. $viewangles_y$), takže hráči nacházející se před ním jsou na radaru nahoře, ty za ním jsou dole. Radar pracuje ve 2D a nezobrazuje tedy výškovou polohu hráčů.

3.8 Menu

Cheat má jednoduché uživatelské rozhraní určené k nastavování funkcí cheatu, aktivuje se klávesou Insert. Menu je vykreslované stejnými funkcemi rozhraní *ISurface* které používá ESP. Stisk klávesy cheatu oznamuje hooknutá Windows procedure okna hry (viz kapitola). Menu je rozděleno na několik karet podle funkcí cheatu, ty pak obsahují zaškrtávací políčka, posuvníky, výběrové seznamy atd. Ty jsou vykresleny také funkcemi *ISurface* jako *DrawRect, DrawLine* apod. Menu je ovládáno myší (to umožňuje funkce Source enginu *GetCursorPosition*, která vrací pozici kurzoru) a menu lze také přesouvat.

Obrázek 7: Menu cheatu otevřené na kartě Aimbot

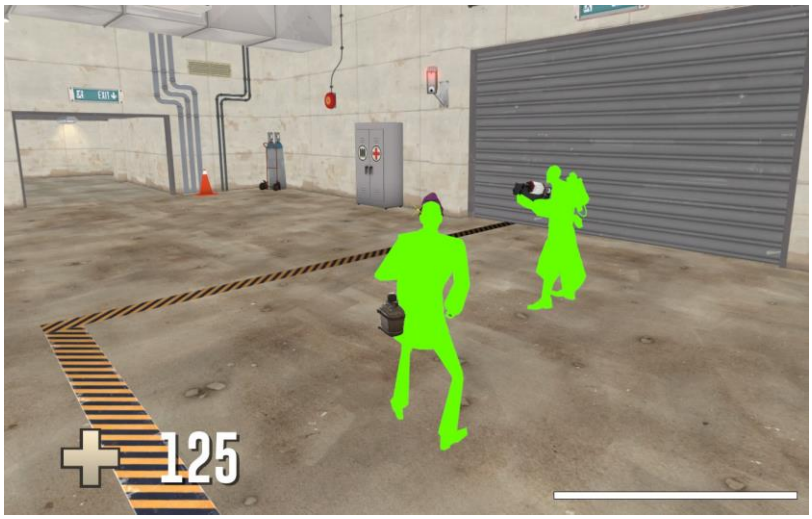
3.9 Ostatní funkce

Přepsání ConVaru "*sv_cheats*" umožňuje upravovat hodnoty většiny ostatních ConVarů a používat normálně nepovolené příkazy ve vývojářské konzoli. Nejjednodušším příkladem je funkce cheatu Thirdperson, tedy pohled z třetí osoby, aktivovaný příkazem "thirdperson". Cheat dále umožňuje například vypnout stíny, čímž se celý herní svět vykresluje jako pod přímým světlem ("mat_fullbright 1"), nebo vypnout renderování vody, protože je špatně průhledná ("mat_drawwater 0").

Úpravou některých NetVarů cheat umožňuje hráči například nastavit šířku viditelného zorného pole nad maximálně povolených 90° ("m_iFOV") nebo obejít zákaz některých modifikací hry ("sv_pure").

Cheat také hookuje funkci *DrawModelExecute*. Díky tomu může měnit "materiál" modelů herních postav a jeho vlastnosti, za účelem zlepšení jejich viditelnosti a vykreslování postav i za zdmi. Na rozdíl od funkce ESP nic nekreslí, nemusí ve smyčce procházet všechny entity (funkci *DrawModelExecute* hra volá při vykreslování každé postavy) ani nepřepočítává pozici postav na 2D obrazovku.

Funkce Fake Lag blokuje odesílání paketů na server a odešle jen každou x-tou (počet neodeslaných paketů "x" lze nastavit v menu cheatu) čímž se hráčova postava pro ostatní hráče při pohybu "zasekává" nebo téměř teleportuje. Takovýto pohyb ztíží nepřátelům míření, jelikož je velice nepředvídatelný. Neodesílání paketů také někdy způsobuje nezaznamenávání zásahů od nepřátel (hlavně pro hitscan zbraně a backstab), nebo různé desynchronizace herních akcí. Při nižším počtu nepřijatých paketů server pro lokálního hráče tyto výpadky kompenzuje a v mnohém věří klientovi, pro něj je tedy míření i pohyb téměř bezproblémový. To však platí pouze pokud je počet konsekutivně neodeslaných paketů nižší než 15.



Obrázek 8: Pohled z třetí osoby, změna materiálu a barvy postav a vypnutí stínování

4 DETEKOVATELNOST CHEATU

4.1 Valve anti-cheat

Valve vyvinulo svůj vlatní anti-cheat, nazvaný Valve Anti-Cheat (VAC), který používá pro ochranu svých her (včetně TF2) před cheatery a poskytuje ho i cizím společnostem. Na službě Steam ho k 12. 3. 2019 využívá přes 500 her. Když VAC odhalí používání cheatu, hráč dostane na hře ve které chatoval VAC ban a bude mu zakázano se připojovat na servery se zapnutou VAC ochranou. V rámci služby Steam může také následovat uzamčení herního inventáře, aby hráč nemohl své věci prodat, nebo si je poslat na jiný účet. Jediná možnost, jak VAC ban obejít, je vytvoření nového Steam účtu a znovu zakoupení hry.

VAC tvoří desítky modulů ve formě DLL souborů, do hry je načítají procesy služby Steam. Valve se snaží komplikovat reverzování a obcházení VAC různými prostředky. Nikdy nenačte do hry všechny moduly, aby nikdo neměl přístup ke kompletnímu VAC. Často je načítá se zpožděním a postupně. Druhá verze VAC (VAC2) nejprve stáhla DLL soubory do složky %temp% (C:\Users\<*aktuální uživatel*>\AppData\Local\Temp) a poté pomocí *LoadLibrary* načítala do hry. Nejnovější verze VAC (VAC3) ale zašifrované moduly streamuje ze svých serverů přímo do hry a *LoadLibrary* kompletně obchází pomocí manual mappingu - byte po bytu manuálně zapisuje obraz DLL souboru do paměti hry a poté volá jeho prvotní funkci (ta se v tomto případě ani nemusí jmenovat *DLLMain*). Samotné bany pak také často přichází se zpožděním, v případě objevení nového cheatu nebo použití nové metody odhalování cheatů až o 2 týdny, aby vývojáři nemohli poznat, co způsobilo detekci jejich cheatu. Bany vydává hromadně, naráz, aby cheateři neměli čas reagovat na bany ostatních hráčů a včas přestali používat svůj software. VAC je tedy spíše pasivní a soustředí se na veřejně dostupné (placené i neplacené) cheaty. Snaží se získávat o cheatech co nejvíce informací a poté všechny uživatele daného cheatu zasáhnout.

4.2 Skenování paměti

VAC skenuje paměť hry a porovnává její části se svou databází cheatů. Snaží se tak objevit funkce, které cheaty používají, nebo celé cheaty v paměti. Jedna z metod, jak snížit riziko odhalení, je tedy vytvoření vlastního privátního cheatu, který nebude s nikým sdílen. To ale jen může zamezit poznání specifického cheatu. Spousta funkcí v cheatech může být podobná nebo i identická s jinými cheaty. Pomoci tak může přidání tzv. junk kódu (instrukce, které ve výsledku nic užitečného nedělají, ale obfuskují kód cheatu).

4.2.1 Virtualizace

Pro ochranu cheatu proti rozpoznání jeho funkcí v databázích VAC byl použit program VMProtect[15]. Ten spustitelné soubory a dynamické knihovny chrání před reverzováním pomocí virtualizace kódu. Namísto obyčejného šifrování, kdy kód musí být za běhu programu dekryptován, je celý kód přepsán pro běh na vlastním virtuálním procesoru s vlastním setem instrukcí. Cheat tak běží uvnitř tzv. virtual machine (VM) a jeho instrukcím se dá jen těžko porozumět, hlavně se ale velmi liší od instrukcí původních. Architektura VM je údajně odlišná pro každý soubor, který VMProtect upraví (11). Takováto obfuskace kódu má ale i nechtěné dopady - velikost souboru cheatu se zvětšila více než 17 krát a jeho běh je pomalejší. Jediná instrukce původního kódu údajně může být přepsána do více než 50 instrukcí ve VM (11). Virtualizací také schováme importované funkce z knihoven Windows API a jakékoli textové řetězce zanechané kompilátorem. Téměř celý soubor tvoří jen jediný segment, jehož název se dá změnit. VMProtect zanechá v segmentu .text (část pro instrukce) jen to nejnutnější a v segmentu .rdata se zdají být uloženy všechny Unicode charaktery. Při běhu programu se z nich pravděpodobně sestavují textové řetězce potřebné např. pro import DLL souborů.

4.2.2 XOR šifrování

Stále jsou ovšem jednoduše čitelné všechny vlastní, pevně naprogramované textové řetězce (veškerý text napsaný ve zdrojovém kódu v uvozovkách). Názvy všech upravovaných ConVarů a NetVarů, bytové signatury, CreateInterface a názvy rozhraní a především text menu. V paměti jsou tak lehce čitelné řetězce jako "Aimbot key", "Radar" apod. VMProtect neposkytuje žádnou možnost pro automatické šifrování textových řetězců, takže byla použita vlastní implementace tzv. XOR šifrování.

Jednoduché XOR šifrování spočívá v použití operátoru *bitwise XOR* (exclusive or) mezi dvěma charaktery. Výsledek *bitwise XOR* je 1 pokud odpovídající bity jeho dvou operandů jsou opačné.

[15] VMProtect je placený software, jeho webová stránka je dostupná z: https://vmpsoft.com/

```
char a = 'a'; //znak UNICODE 'a' je v binárním zápisu 01100001
char Z = 'Z'; //znak UNICODE 'Z' je v binárním zápisu 01011010
char x = a ^ Z; //výsledek (a bitwise-XOR Z) je       00111011 → UNICODE znak ';'
```

Výhodou XOR šifrování je, že stejným algoritmem, kterým je šifrováno, může být posléze zpět dešifrováno. Pro praktické použití je tedy definován charakter, kterým jsou řetězce zašifrovány a poté je provedena operace *bitwise-XOR* pro každý charakter v řetězci. Všechny textové řetězce jsou zašifrovány ještě před kompilací pomocí definovaného klíče a dešifrovány jsou na stack až za běhu cheatu, když jsou používány. Nikdy tak nezůstane dešifrovaný text v paměti.

```
char* XOR(char text[])//XOR dešifrovací funkce
{
char xorKey = 'P'; //definuje klíč pro dešifrování
int len = strlen(text); //získání délky dešifrovávaného řetězce pomocí funkce strlen
//pro každý charakter řetězce provede operaci bitwise-XOR s klíčem 'P'
for (int i = 0; i < len; i++)
	{
		text[i] = text[i] ^ xorKey;
	}
	return text; //vrátí dešifrovaný řetězec
}
char* chCheats = XOR("#&_3851$#"); //volá funkci XOR s šifrovaným "sv_cheats" pomocí
klíče 'P' jako argument, vrácenou hodnotu uloží do vytvořené proměnné chCheats
ConVar* sv_cheats = gInts.cvar->FindVar(chCheats); //volání funkce FindVar s
argumentem chCheats, tedy "sv_cheats"
```

4.2.3 Mutace

Další metodou, jak zvýšit jedinečnost kódu cheatu, je mutace jeho funkcí. Toto VMProtect zvládne sám. Pokud je s .DLL souborem sestaven i .MAP soubor (ten poskytuje informace o relativních offsetech a názvech funkcí v kompilovaném souboru), může být u jednotlivých funkcí určeno, aby je VMProtect mutoval. Mutace ve VMProtect přepíše funkce ještě na úrovni fyzického procesoru, tedy v původním setu instrukcí a přidá junk kód. Je to účinná metoda ochrany právě před automatizovaným vyhledáváním signatur.

4.3 Skrytí injekce kódu

Spousta legitimních programů může injektovat vlastní kód do procesu se hrou, např. komunikační programy jako je Mumble nebo Discord, programy jako Fraps, které měří výkon hry (počet snímků za sekundu, které počítač stíhá renderovat apod.) nebo software poskytovaný k ovladačům grafických karet (GeForce Experience od společnosti Nvidia). Samotné použití *LoadLibrary* tedy samo o sobě k odhalení cheatu nestačí. Nicméně je to rozhodně akce, která upoutá pozornost VAC, který se pak bude snažit dále zjistit, odkud injektovaný kód / modul pochází. Následující metody (ale i hookování funkcí zmíněné v kapitole 2.4) jsou často využívány při tvorbě malwaru. Není tedy výjimkou, že některé anti-virové programy označují cheaty jako škodlivý software, jedná se však o tzv. false-positive (falešně pozitivní označení cheatu za malware). Anti-cheat však oproti antiviru má velkou nevýhodu - tzv. "výhoda pochybnosti" ("benefit of the doubt") je na straně hráče. Když antivirový program odhalí potenciálně nebezpečný software, může zablokovat spuštění

programu, nebo soubor uzamkne do karantény. Ban od anti-cheatu může znamenat peněžní ztrátu, ztrátu několika set hodinového postupu ve hře, v případě známé osobnosti poškodí jeho image a pro profesionální hráče znamená konec kariéry, pro tým ztrátu výher, fanoušků atd... Anti-cheat si tak musí být 100% jistý, že hráč cheatuje, než mu udělí ban. Následující části vysvětlují, jak lze toto rozhodování ještě více ztížit.

4.3.1 Obcházení LoadLibrary

Obejití samotné funkce *LoadLibrary* nám může pomoci za předpokladu, že VAC tuto funkci hookuje, nebo sleduje volání Windows API funkcí.

Sledování injektoru v programu API Monitor v2 odhaluje, že funkce Windows API *LoadLibrary* dále volá funkci *LdrLoadDll*. Doposud zmíněné funkce Windows API byly součástí Win32 API (soubor kernel32.dll). Funkce *LdrLoadDll* společně s ostatními Loader (Ldr) funkcemi je součástí Native API, tedy nativních funkcí OS Windows (soubor ntdll.dll). NTDLL je nejnižší rozhraní mezi kernelem a Uživatelským prostředím. Microsoft k nativním funkcím neposkytuje žádnou dokumentaci, ale v průběhu let byly analyzovány a reverzovány.

Nový injektor funguje podobně jako ten původní, ale nejprve pomocí funkce *GetProcAdress* získá adresu funkce *LdrLoadDll* uvnitř ntdll.dll a do alokované paměti v herním procesu vepíše data pro *LdrLoadDll* a poté vlastní funkci, která *LdrLoadDll* zavolá. Stejně jako můžeme obejít *LoadLibrary*, lze podobný postup uplatnit pro obcházení *CreateRemoteThread* funkce, která vytváří nové vlákno v procesu hry. Injektor místo ní volá nativní funkci *NtCreateThreadEx*, jež je také součástí ntdll.dll. Nejen že se vyhýbá používání API funkcí vysoké úrovně při manipulaci s herním procesem, *NtCreateThreadEx* navíc umožňuje nově vytvořené vlákno skrýt (12), pokud nastavíme její parametr CreateFlags na:

```
THREAD_CREATE_FLAGS_HIDE_FROM_DEBUGGER (0x00000004)
```

Zdrojový kód nového injektoru je v Příloze 2.

LdrLoadLibrary však stále zanechává nechtěné stopy. Všechny načtené moduly procesu jsou zaznamenány v PEB (Process Environment Block), který záznamy uchovává ve čtyřech seznamech:

- InLoadOrderModuleList
- InMemoryOrderModuleList
- InInitializationOrderModuleList
- LdrpHashTable

PEB je uložen v paměti přístupné z Uživatelského prostředí a je tedy možné záznam o modulu cheat odstranit.

Po těchto opatřeních jsou všechny záznamy o injekci cheatu, které jsou čitelné z Uživatelského prostředí (ve kterém VAC operuje), odstraněny.

4.3.2 Přepsání PE File headeru

Každý spustitelný soubor nebo dynamická knihovna pod Windows má na svém začátku File header, který se skládá z MS-DOS záhlaví a krátké MS-DOS aplikace, signatury identifikující formát souboru jako portable-executable (PE) soubor, COFF záhlaví a nepovinný Optional header. Anti-cheat by záhlaví mohl využít pro vytvoření signatury modulu a rozpoznání cheatu. Můžeme tedy záhlaví vymazat (přepsat na nuly), byty záhlaví randomizovat nebo využít jiného načteného .DLL souboru v procesu hry a maskovat záhlaví cheatu za jiný, legitimní DLL soubor. VMProtect sice částečně záhlaví upravuje, ale analýza souboru cheatu v programu PEview ukázala, že prvních 500 bytů souboru stále tvoří záhlaví, které je možné přepsat. Následující kód, přidaný do funkce *DLLMain* v cheatu přepíše prvních 500 bytů načteného modulu cheatu na prvních 500 bytů modulu kernel32.dll, který je načtený ve většině procesech ve Windows.

```
void* kernel32 = reinterpret_cast<void*>(GetModuleHandleA("kernel32.dll"));
memcpy(hInstance, kernel32, 0x01F4);
```

4.4 Skrytí souboru cheatu

Analýza jednoho z modulů VAC v IDA Pro odhaluje používání mnoha funkcí Win32 API, mimo jiné *GetModuleHandleA, GetModuleFileName, CreateFileA* a *VirtualQueryEx*.

Metody popsané v kapitole 4.3.1 a 4.3.2. zamezí najití modulu cheatu v paměti a tím jeho DLL souboru. Pro jistotu je však žádoucí, aby samotný soubor cheatu (a injektor) pokud možno nikdy nebyli na disku. Jednou z možností je injektovat z flash disku a randomizovat názvy souborů, nebo vytvořit loader, který cheat před injektováním stáhne z vlastního serveru. Také je možné místo vytváření procesu injektoru využít cizí proces - injektovat kód do jiného procesu, který poté injektuje cheat do procesu hry. Ideálně se jedná o proces, který také injektuje do procesu hry vlastní legitimní DLL soubor, např. komunikační program Mumble. Schovávání záznamů o injektování cheatu ani jeho maskování za legitimní programy však nikdy nebude 100 % nedetekovatelné, ale pro potřeby jedince jsou použité metody popsané v předchozích kapitolách dostatečné, protože VAC nemá s čím porovnat injektovaný kód.

4.5 Možná vylepšení VAC

VAC funguje spíše pasivně, snaží se tedy cheaty pouze odhalit, nikoli zamezit jejich fungování. Výjimkou je kontrola bezpečnostních funkcí Windows Driver Signature Enforcement (DSE), které zakazuje instalování necertifikovaných ovladačů (využití ovladačů je popsáno v kapitole 4.7.2) a kontrola Data Execution Prevention (DEP), které blokuje spouštění kódu z částí paměti neoznačených jako Executable. Právě skenování pro Executable (spustitelné) části virtuální paměti by mohl VAC využít odhalení cheatu vytvořeného v rámci této práce.

V případě že VAC najde pomocí *VirtualQueryEx* spustitelnou část virtuální paměti, ale ta nebude patřit žádnému modulu (funkce *GetModuleHandle* nebo *GetModuleFileName* nevrátí

platnou hodnotu) ani vlastnímu procesu hry, může potenciální cheat zablokovat (anulovat tuto část paměti, nebo jednoduše ukončit proces hry). Tato metoda spoléhá na to, že neexistuje legitimní důvod pro schovávání modulu v paměti. VAC by také mohl vytvořit tzv. whitelist povolených legitimních modulů a hledat tak i spustitelné části paměti, které nějakému modulu patří, ale nejedná se o známý, povolený modul. (Tedy opak skenování pro zakázaný kód cheatů). Cheat by se však mohl vydávat za legitimní modul, protože testování integrity všech povolených modulů by bylo problematické a výpočetně náročné. To je další nevýhoda anti-cheatovacího softwaru oproti anti-viru, protože si nemůže dovolit zajišťovat fair-play na úkor výkonu hry.

Další možností je kontrolování integrity ukazatelů uvnitř VMT. Zaznamenáváním upravených ukazatelů by bylo možné i rozpoznat specifický cheat.

4.6 Možné detekce v TF2

Účel VAC je být anti-cheatovým řešením pro jakoukoli hru, některé z funkcí cheatu lze však detekovat přímo hrou nebo serverem, např. upravování *viewangles* mimo povolené hodnoty funkcí Anti-aim nebo úprava ConVaru "sv_cheats". Při zavedení adekvátních opatření na serveru při normalizování úhlů *viewangles* by aktuální verze funkce Fake Anti-aim zkrátka fungovat nemohla, ale lokální upravovaní ConVarů by šlo nahradit jiným řešením, např. funkce Thirdperson může místo obcházení "sv_cheats" upravit NetVar "m_nForceTauntCam".

Hráči většinou dokážou odhalit používání Aimbotu (záleží na jeho nastavení) a proto společnost Valve ve své hře Counter-Strike: Global Offensive (CS:GO) zavedla systém zvaný "Overwatch", kde hráči sledují záznam části hry z pohledu podezřelého hráče a reportují, jestli je průkazné, že hráč používá cheat. Valve také vyvíjí umělou inteligenci, kterou cvičí v rozpoznávání mezi lidským mířením a Aimbotem (13). Opatrné používání funkcí ESP apod. je však téměř nemožné rozpoznat.

4.7 Co dál?

V případě, že by cheat měl být veřejně distribuován, bylo by za potřebí ověřit kompatibilitu cheatu s jinými verzemi Windows a provést další optimalizaci cheatu. V aktuálním stavu některé funkce snižují výkon hry (zejména kvůli velkému počtu iterací for smyček).

Pro zajištění nedetekovatelnosti cheatu poskytovanému více hráčům by bylo nutné poskytovat co nejvíce odlišná sestavení cheatu jednotlivým klientům. Toho je možné docílit polymorfismem kódu, použitím pokročilé randomizované mutace, úpravou velikosti cheatu, přidáním různého počtu junk kódu, randomizováním File headeru, randomizováním názvu sekcí souboru atd...

Pro snížení šance rozpoznání cheatu metodou kontrolování ukazatelů ve VMT (viz kapitola 4.5) by cheat mohl hookovat další funkce pro porušení konzistence upravených ukazatelů, nebo využívat také jiné varianty VMT hookingu, popřípadě jiné metody hookování funkcí.

Přiblížení se opravdové nedetekovatelnosti by však vyžadovalo přepsání cheatu a změnu základních principů jeho fungování, jak je uvedeno v následujících kapitolách.

4.7.1 Manual mapping a thread hijacking

Manual mapping (manuální vepsání modulu cheatu do paměti hry) by kompletně obešlo načítání DLL souboru funkcemi Windows API. Princip je jednoduchý - přepsat byty DLL souboru do paměti manuálně a zavolat prvotní funkci. Žádný soubor by tak ani nemusel existovat, stačilo by byty streamovat z vlastního serveru, nebo je pevně vepsat do injektoru. V praxi se však jedná o velmi komplexní úkol. Kompletně manuální manual mapping vyžaduje provedení následujících kroků:

- Alokovat paměť pro DLL
- Vepsat File header
- Upravit protekci paměti a vepsat jednotlivé sekce DLL
- Relokovat kód závislý na *ImageBase* pro fungování na nových adresách
- Mapování všech dalších DLL souborů které načítá samotný cheat
- Zavolání vstupního bodu (*DLLEntryProc*)

Druhá zmíněná metoda, thread hijacking, odstraňuje nutnost vytvářet v procesu nové vlákno. Injektor může vepsat do procesu vlastní shellcode[16], suspendovat některé z aktivních vláken v procesu, upravit jeho EIP (Instruction Pointer - ukazatel na další instrukci) na adresu shellcode který načte DLL, upravit EIP vlákna na původní stav a znovu aktivovat vlákno.

4.7.2 Kernelové cheaty

Cheat běžící na úrovni Kernelu má přímý přístup k hardwaru počítače (včetně celé paměti RAM), ke všem kernelovým funkcím (čímž se také může kompletně vyhnout používání Windows API) a pro anti-cheat může být velmi těžce detekovatelný, zvláště pokud je externí a anti-cheat běží pouze na úrovni Uživatelského prostředí (jako VAC). Kernelový cheat také může manipulovat se samotným anti-cheatem nebo prostředím Windows.

K přístupu ke Kernelu v OS Windows je zapotřebí vlastního ovladače, jejich certifikování je ale drahé a vývojáři anti-cheatů by mohli podle certifikátů ovladače cheaty blokovat. Alternativou je využití exploitovatelného existujícího, certifikovaného ovladače (v tom případě ale může anti-cheat stále tyto cheaty blokovat, pokud odhalí které ovladače jsou používány) nebo obejití DSE (viz kapitola) ve Windows, například využitím programu DSEFix[17].

---

[16] Shellcode je set instrukcí napsaný ve strojovém kódu, určený k provedení po injektování do procesu

[17] DSEFix, oficiálně nazván "Windows x64 Driver Signature Enforcement Overrider" je open-source program, zdarma dostupný z: https://github.com/hfiref0x/DSEFix

Některé pokročilejší anti-cheaty, jako například BattleEye nebo Easy Anti-Cheat používají vlastní ovladače pro ochranu před manipulováním s hrou z úrovně kernelu.

4.7.3 Hardwarové cheaty

Článek na internetovém blogu společnosti E-Sports Entertainment Association (ESEA), provozující populární herní ligu ESEA League, která je známá také díky svému vlastnímu pokročilému anti-cheatu, popisuje nejnovější trendy ve tvorbě cheatů využívajících hardware místo tradičních metod získávání a modifikování dat (14). Zmiňuje například používání mikrokontrollerů jako je Arduino nebo Raspberry Pi pro zaznamenávání a dešifrování herních paketů posílaných serverem. Na konci roku 2013 se na sociální síti YouTube objevilo video demonstrující produkt využívající tuto metodu pro vytvoření radaru (podobnému jako v tomu v kapitole) vykreslovaném na mobilním telefonu (15). ESEA v článku tvrdí, že zamezila cheaterům v používání této metody a dokonce odhalila a potrestala hráče a vývojáře těchto cheatů. Jako novější metodu podvádění uvádí Direct Memory Access (DMA), tedy fyzické čtení (nebo psaní) paměti RAM přes PCI Express rozhraní. Data jsou posílána jinému počítači, který z nich získává potřebné informace a opět může například vytvořit radar nebo jiné funkce k cheatování. ESEA na odhalování těchto cheatů aktuálně pracuje.

4.8 Legální aspekt

Hry bývají chráněny proti cheatům v rámci licenční smlouvy s koncovým uživatele (EULA), díky které může hráč dostat ban, pokud používá programy k získání neoprávněných výhod proti ostatním hráčům. Pro většinu herních společností tímto legální boj proti cheatům končí, ale např. společnost Blizzard Entertainment se úspěšně soudila s poskytovateli cheatů na základě porušování autorských práv (16). Vzhledem k počtu poskytovatelů cheatů a absenci zákonů postihujících podvádění ve videohrách je legální boj pro herní společnosti nákladný a neefektivní. Společnosti prodávající cheaty navíc někdy oficiálně operují ze zemí jako Rusko nebo Čína, čímž jsou legální možnosti herních společností prakticky nulové. Výjimkou je Jižní Korea, kde je tvorba a distribuce cheatů pro videohry postižitelná pokutou nebo trestem odnětí svobody až na 5 let (17).

5 ZÁVĚR

Byl vytvořen vlastní cheat pro hru Team Fortress 2, který ani po několika měsících testování nebyl detekován. V této práci byly popsány principy pomocí kterých cheat funguje a také jeho funkce. Dále byly popsány použité metody pro snížení možností odhalení tohoto cheatu a uvedena možná vylepšení Valve Anti-Cheatu. Nakonec byly zmíněny další možnosti tvorby nedetekovatelných cheatů.

Metody popsané v této práci (zejména reverzní inženýrství a hookování funkcí) a problematika maskování a detekování škodlivého kódu jsou témata adekvátní nejen pro cheaty, ale především malware a ochranu před ním. Tato práce je tedy současně náhledem do světa počítačové bezpečnosti se záměrem dalšího studia v této oblasti jako průprava na budoucí profesní kariéru autora.

Všechny cíle práce byly splněny. Navíc byla nalezena dosud neznámá chyba herních vývojářů, což umožnilo vytvoření patrně jediného speedhacku pro hru Team Fortress 2. Autor o této chybě včetně detailů cheatu a doporučení změn pro VAC společnost Valve, která VAC i TF2 vyvíjí, informoval.

Na tuto práci by mohlo navázat studium využití manual mappingu, thread hijackingu a kernelového prostředí pro tvorbu cheatů či malwaru se zaměřením na jejich odhalování nebo využití externího hardwaru pro tyto účely.

6 POUŽITÁ LITERATURA

1. Cheating in video games: History. In: *Wikipedia* [online]. [cit. 2019-03-12]. Dostupné z: https://en.wikipedia.org/wiki/Cheating_in_video_games#History

2. VALENTINE, Rebekah. The International brings record-breaking prize pool: Dota 2 tournament's $24.8 million pool once again largest in esports history. In: *Gameindustry* [online]. 20. Srpna 2018 [cit. 2019-03-12]. Dostupné z: https://www.gamesindustry.biz/articles/2018-08-20-the-international-again-brings-record-breaking-prize-pool

3. GRANADOS, Nelson. Report: Cheating Is Becoming A Big Problem In Online Gaming. In: *Forbes* [online]. Apr 30, 2018 [cit. 2019-03-12]. Dostupné z: https://www.forbes.com/sites/nelsongranados/2018/04/30/report-cheating-is-becoming-a-big-problem-in-online-gaming/#372567ce7663

4. MAIBERG, Emanuel. Hacks! An investigation into the million-dollar business of video game cheating. In: *PC Gamer* [online]. April 30, 2014 [cit. 2019-03-12]. Dostupné z: https://www.pcgamer.com/hacks-an-investigation-into-aimbot-dealers-wallhack-users-and-the-million-dollar-business-of-video-game-cheating/

5. Cheating in online games: Implementation of cheats. In: *Wikipedia* [online]. [cit. 2019-03-12]. Dostupné z: https://en.wikipedia.org/wiki/Cheating_in_online_games#Implementation_of_cheats

6. Virtuální paměť: Výhody virtualizace. In: *Wikipedia* [online]. [cit. 2019-03-12]. Dostupné z: https://cs.wikipedia.org/wiki/Virtu%C3%A1ln%C3%AD_pam%C4%9B%C5%A5#V%C3%BDhody_virtualizace

7. x86 calling conventions. In: *Wikipedia* [online]. [cit. 2019-03-12]. Dostupné z: https://en.wikipedia.org/wiki/X86_calling_conventions

8. LUDWIG, Joe a Jørgen P. TJERNØ. Mathlib_base.cpp. In: *GitHub* [online]. Dec 3, 2013 [cit. 2019-03-12]. Dostupné z: https://github.com/ValveSoftware/source-sdk-2013/blob/master/sp/src/mathlib/mathlib_base.cpp#L919

9. KENNEDY, John a Michael SATRAN. Transforms (Direct3D 9). In: *Microsoft Docs* [online]. 05/31/2018 [cit. 2019-03-12]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/direct3d9/transforms

10. KENNEDY, John a Michael SATRAN. View Transform (Direct3D 9). In: *Microsoft Docs* [online]. 05/31/2018 [cit. 2019-03-12]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/direct3d9/view-transform

11. CHEVET, Samuel. *Inside VMProtect* [online]. 16 January 2015 [cit. 2019-03-12]. Soubor ve formátu PDF. Dostupné z: lille1tv.univ-lille1.fr/telecharge.aspx?id=d5b2487e-cacc-4596-ab37-dab2b362cb9e

12. Ntpsapi.h. In: *SourceForge* [online]. [cit. 2019-03-12]. Dostupné z: https://processhacker.sourceforge.io/doc/ntpsapi_8h_source.html

13. GDC 2018: John McDonald (Valve) - Using Deep Learning to Combat Cheating in CSGO. In: *YouTube* [online]. 31. 3. 2018 [cit. 2019-03-12]. Video. Dostupné z: https://www.youtube.com/watch?v=ObhK8lUfIlc

14. ESEA TEAM. Cat And Mouse — ESEA Anti-Cheat. In: *ESEA.net Official Blog* [online]. 23 October 2018 [cit. 2019-03-12]. Dostupné z: https://blog.esea.net/esea-hardware-cheats

15. ORGANNERVIDEOS. HardwareRadar - The First Undetectable Hardware Hack. In: *YouTube* [online]. 17. 11. 2013 [cit. 2019-03-12]. Video. Dostupné z: https://www.youtube.com/watch?v=QUuuTPlNMyY

16. WESSING, Taylor. Cheating in games: legality and market trends: Cheating in multiplayer games is an annoyance for gamers who play by the rules and is increasingly becoming a revenue issue for games developers and publishers. In: *Lexology* [online]. December 14 2017 [cit. 2019-03-12]. Dostupné z: https://www.lexology.com/library/detail.aspx?g=e8271099-771c-4854-946e-41f7bf37671b

17. CHALK, Andy. Creating hacks for online games could now earn you jail time in South Korea: That aimbot could land you in the slammer. In: *PC Gamer* [online]. December 07, 2016 [cit. 2019-03-12]. Dostupné z: https://www.pcgamer.com/south-korea-makes-cheating-in-online-games-an-actual-crime/

7 SEZNAM OBRÁZKŮ

Obrázek 1: Virtuální adresní prostor v OS Windows 8
Obrázek 2: Diagramové zobrazení v IDA Pro 10
Obrázek 3: VMT Hooking 13
Obrázek 4: Pozice hráčů a výsledný vektor aimbotu 20
Obrázek 5: Rozdíl mezi vykreslovaným modelem a hitboxem (modré kvádry) 24
Obrázek 6: Funkce ESP s nastavením Box, Bones, Name, Class, Health (both) a Weapon 28
Obrázek 7: Menu cheatu otevřené na kartě Aimbot 29
Obrázek 8: Pohled z třetí osoby, změna materiálu a barvy postav a vypnutí stínování 30

8 PŘÍLOHA 1: UKÁZKA ZDROJOVÉHO KÓDU - FUNKCE TRIGGERBOT

```
#include "Triggerbot.h"
#include "Util.h"
CTriggerbot gTrigger;
void CTriggerbot::Run(CBaseEntity* pLocal, CUserCmd* pCommand)
{
//získá hráčovu aktivní zbraň
CBaseCombatWeapon* pWep = pLocal->GetActiveWeapon();
//pokud hráč není naživu, nepokračuje
if (pLocal->GetLifeState() != LIFE_ALIVE)
	return;
//pokud je zapnutá funkce Auto Backstab, zjistí zda hráč může backstab provést, pokud ano
tak vystřelí (volá funkci CanBackStab která vrací hodnotu NetVaru m_bReadyToBackstab)
if (backstab.value)
	if (pLocal->szGetClass() == "Spy" && pWep->GetSlot() == 2 && pWep->CanBackStab())
	pCommand->buttons |= IN_ATTACK;
//pokud Triggerbot není zapnutý, nepokračuje
if (!enabled.value)
	return;
//pokud neni stistknutá klávesa pro aktivaci triggerbotu, nepokračuje
//při možnosti "None" (žádná) vrací funkce pravdivou hodnotu
if (!Util->IsKeyPressedMisc(key.value))
	return;
//získání vektoru aktuálních úhlů pohledu
Vector vAim;
gInts.Engine->GetViewAngles(vAim);
//?
Vector vForward;
AngleVectors(vAim, &vForward);
//získá ItemDefinitionIndex aktuální zbraně
auto pClass = pWep->GetItemDefinitionIndex();
//pokud je zbraň pouze na blízko,
if (pWep->GetSlot() == 2)
	vForward = vForward * 8 + pLocal->GetEyePosition();
//pokud je aktuální postava "Pyro" a zbraň třídy tf_weapon_flamethrower, násobí vForward
17krát (dosah plamenů)
else if (pLocal->szGetClass() == "Pyro" && pClass != 1178 && pWep->GetSlot() == 0)
	vForward = vForward * 17 + pLocal->GetEyePosition();
//jinak předpokládáme hitscan zbraň, vForward násobíme co nejvyšší hodnotou
else vForward = vForward * 9999 + pLocal->GetEyePosition();
Ray_t ray; //inicializace paprsku
trace_t trace; //inicializace třídy CGameTrace, do které TraceRay vrátí výsledek
CTraceFilter filt; //inicializace filtru pro TraceRay
filt.pSkip = pLocal; //filtr ignoruje hráčovu postavu
//nastavení paprsku, počátek v pozici očí, konec podle vForward
ray.Init(pLocal->GetEyePosition(), vForward);
//vysílá paprsek, výsledek vrací na adresu trace
gInts.EngineTrace->TraceRay(ray, 0x46004003, &filt, &trace);
if (!trace.m_pEnt)
	return; //pokud TraceRay nevrátil hráčskou entitu, nepokračuje
if (trace.hitgroup < 1)
	return; //pokud netrefil žádnou část hráčského těla, nepokračuje
if (cloaked.value && trace.m_pEnt->GetCond() & TFCond_Cloaked)
	return; //pokud má Triggerbot ignorovat neviditelné hráče a trefený hráč je
neviditeln, nepokračuje
if (headonly.value && trace.hitbox != 0)
	return; //pokud má Triggerbot střílet pouze na hlavu a TraceRay netrefil hitbox
hlavy (0), nepokračuje
if (ignorefriendly.value && trace.m_pEnt->GetTeamNum() == pLocal->GetTeamNum())
	return; //pokud má Triggerbot ignorovat spoluhráče a týmy lokálního hráče a hráče
kterého trefil TraceRay jsou identické, nepokračuje
pCommand->buttons |= IN_ATTACK; //do příštího UserCmd přídá, že hráč stisknul spoušť -
vystřelí
}
```

9 PŘÍLOHA 2: ZDROJOVÝ KÓD LDRLOADDLL INJEKTORU

```
#include "Injektor.h"
#define THREAD_CREATE_FLAGS_HIDE_FROM_DEBUGGER 0x00000004
struct LDRLOADDLL_DATA //data pro LdrLoadLibrary
{
	f_LdrLoadDll  pLdrLoadDll; //ukazatel na funkci LdrLoadLibrary
	HANDLE		Out;
	UNICODE_STRING	pModuleFileName; //název DLL souboru
	BYTE		Data[MAX_PATH * 2]; //cesta k DLL souboru
};
void __stdcall LdrLoadDllShell(LDRLOADDLL_DATA * pData);
DWORD LdrLoadDllInject(const char * szDllFile, HANDLE hProc)
{
	LDRLOADDLL_DATA data{ 0 }; //vytvoří instanci třídy LDRLOADDLL_DATA
	data.pModuleFileName.szBuffer = reinterpret_cast<wchar_t*>(data.Data);
//velikost LDRLOADDLL_DATA.pModuleFileName = Data (520)
	data.pModuleFileName.MaxLength = MAX_PATH * 2;
//LDRLOADDLL_DATA.pModuleFileName max length = 520
	size_t len = _strlen(szDllFile); //získá délku názvu DLL souboru
	mbstowcs_s(&len, data.pModuleFileName.szBuffer, len + 1, szDllFile, len);
	data.pModuleFileName.Length = (WORD)(len * 2) - 2;
	HINSTANCE hNTDLL = GetModuleHandleA("ntdll.dll"); //otevření handlu na ntdll.dll
	FARPROC pFunc = GetProcAddress(hNTDLL, "LdrLoadDll"); //získání adresy funkce
LdrLoadDLl
	data.pLdrLoadDll = reinterpret_cast<f_LdrLoadDll>(pFunc); //ukazatel na funkci
LdrLoadDll vloží do dat pro LdrLoadLibraryShell
	//alokuje v procesu hry paměť o velikost LDRLOADDLL_DATA + 200 bytů, vrátí
ukazatel na alokovanou paměť
	BYTE * pArg = reinterpret_cast<BYTE*>(VirtualAllocEx(hProc, nullptr,
sizeof(LDRLOADDLL_DATA) + 0x200, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE));
	//do alokovaného prostoru vepíše LDRLOADDLL_DATA
	WriteProcessMemory(hProc, pArg, &data, sizeof(LDRLOADDLL_DATA), nullptr);
	//do alokovaného prostoru za LDRLOADDLL_DATA vepíše vlastní funkci
LdrLoadDllShell
	WriteProcessMemory(hProc, pArg + sizeof(LDRLOADDLL_DATA), LdrLoadDllShell,
0x100, nullptr);
	//získání adresy funkce NtCreateThreadEx
	auto _thread =
reinterpret_cast<f_NtCreateThreadEx>(GetProcAddress(GetModuleHandleA("ntdll.dll"),
"NtCreateThreadEx"));
	HANDLE hThread = nullptr;
	//vytváří nové skryté vlákno, zavolá funkci LdrLoadDllShell s LDRLOADDLL_DATA
jako argumenty
	_thread(&hThread, THREAD_ALL_ACCESS, nullptr, hProc, pArg +
sizeof(LDRLOADDLL_DATA), pArg, THREAD_CREATE_FLAGS_HIDE_FROM_DEBUGGER, 0, 0, 0,
nullptr);
	WaitForSingleObject(hThread, INFINITE); //čeká až vlákno dokončí práci
	CloseHandle(hThread); //ukončí vlákno
	VirtualFreeEx(hProc, pArg, 0, MEM_RELEASE); //vymaže alokovanou paměť
	return 0;
}
void __stdcall LdrLoadDllShell(LDRLOADDLL_DATA * pData)
{
	pData->pModuleFileName.szBuffer = reinterpret_cast<wchar_t*>(pData->Data);
	pData->pLdrLoadDll(nullptr, 0, &pData->pModuleFileName, &pData->Out);
}
```