Cheating in online video-games

An example with CS:GO

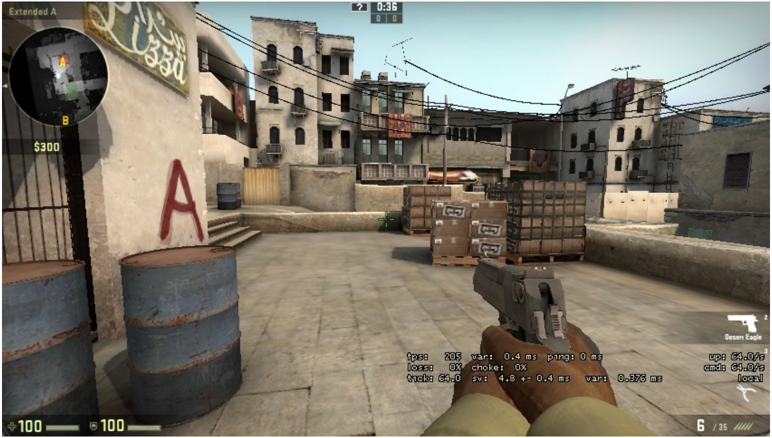
Adrien Garin

EPITA

July 18, 2015

Intro

- Counter Strike: Global Offensive (2012)
- Source Engine (2004)
- Prize Money Awarded: $5,269,708.88 (4 July 2015)



```
fps:   205  var:  0.4 ms  ping: 0 ms                 up: 64.0/s
loss:   0%  choke:  0%                               cmd: 64.0/s
tick: 64.0  sv:  4.8 +- 0.4 ms   var:  0.376 ms         local
```

Figure: CS:GO i

What do we want

- AIMBOT / Trigger BOT
- No Flash
- Wall hack
- ESP
- Radar Hack

Cheat theory

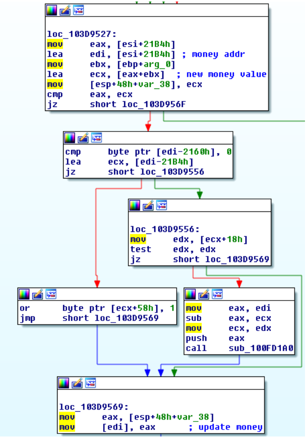
- Change code in .text
- Players attributes and info are in memory
- Find the good addresses
- Objects are very often dynamically allocated
- Hooking

CSGO code

- csgo.exe (resources loading, some checksums)
- client.dll (C_BaseEntity, EntityList, LocalPlayer, RadareBase. . .)
- engine.dll
- server.dll

Infinite money

- Current money is in memory
- Use Cheat Engine to find where
- Then find which instruction wrote to this address
- Patch it



```
loc_103D9527:
mov     eax, [esi+21B4h]
lea     edi, [esi+21B4h] ; money addr
mov     ebx, [ebp+arg_0]
lea     ecx, [eax+ebx]  ; new money value
mov     [esp+48h+var_38], ecx
cmp     eax, ecx
jz      short loc_103D956F
```

```
cmp     byte ptr [edi-2160h], 0
lea     ecx, [edi-21B4h]
jz      short loc_103D9556
```

```
loc_103D9556:
mov     edx, [ecx+18h]
test    edx, edx
jz      short loc_103D9569
```

```
or      byte ptr [ecx+58h], 1
jmp     short loc_103D9569
```

```
mov     eax, edi
sub     eax, ecx
mov     ecx, edx
push    eax
call    sub_100FD1A0
```

```
loc_103D9569:
mov     eax, [esp+48h+var_38]
mov     [edi], eax       ; update money
```

Figure: Money IDA

Why

- We need to read and write to game memory
- We also want to hook some stuff
- We have to inject code

Internal vs External

- You can access the game internally or externally from another process
- Internal cheats can call game functions

LoadLibrary

- Classical method is to call LoadLibrary
- Or LdrLoadDll
- allocate memory for the string my_module.dll in remote process
- Write the string at allocated address
- Create a new remote thread which will execute LoadLibraryA
- But it is not stealth

PEB

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  BYTE                          Reserved4[104];
  PVOID                         Reserved5[52];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutin
  BYTE                          Reserved6[128];
  PVOID                         Reserved7[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

Get PEB

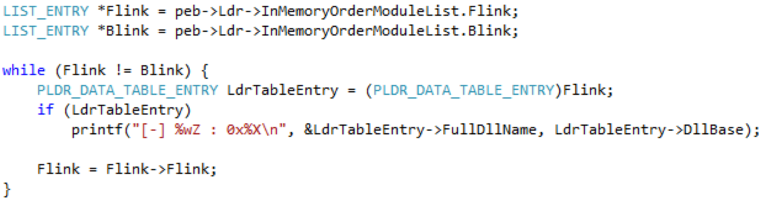
- PEB address is in segment register FS

Getting PEB

```
PPEB peb;

__asm
{
    mov eax, FS:[0x30]
    mov peb, eax
};
```

Iterate Ldr LinkedList



```
LIST_ENTRY *Flink = peb->Ldr->InMemoryOrderModuleList.Flink;
LIST_ENTRY *Blink = peb->Ldr->InMemoryOrderModuleList.Blink;

while (Flink != Blink) {
    PLDR_DATA_TABLE_ENTRY LdrTableEntry = (PLDR_DATA_TABLE_ENTRY)Flink;
    if (LdrTableEntry)
        printf("[-] %wZ : 0x%X\n", &LdrTableEntry->FullDllName, LdrTableEntry->DllBase);

    Flink = Flink->Flink;
}
```

Figure: PEB

Manual mapping

- Allocate enough space in remote process heap
- Patch relocations
- Load dependencies
- Patch imports
- Stealthier than LoadLibrary

Differences with ELFs

- Elfs use position independant code
- Dlls don't
- Dlls are always relocated by the kernel memory manager

PE

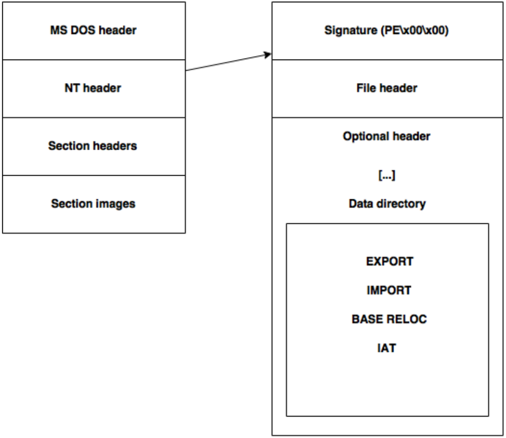


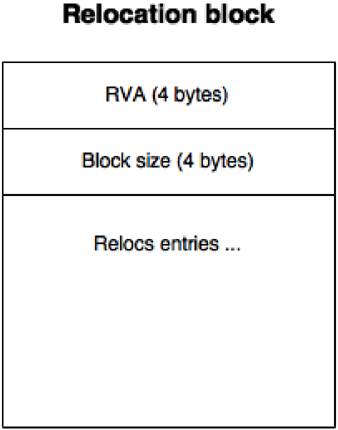
Figure: Portable Executable

Relocations

IMAGE_BASE_RELOCATION

- Num relocs = (SizeOfBlock - 8) / sizeof(WORD)
- The high 4 bits are a relocation type
- The bottom 12 bits are offsets

```
struct IMAGE_BASE_RELOCATION
{
    DWORD VirtualAddress;
    DWORD SizeOfBlock;
};
```



Relocation block

RVA (4 bytes)

Block size (4 bytes)

Relocs entries ...

Figure: Relocation block

Import and IAT

- Use LoadLibrary to load dependencies
- patch IAT

Getting offsets

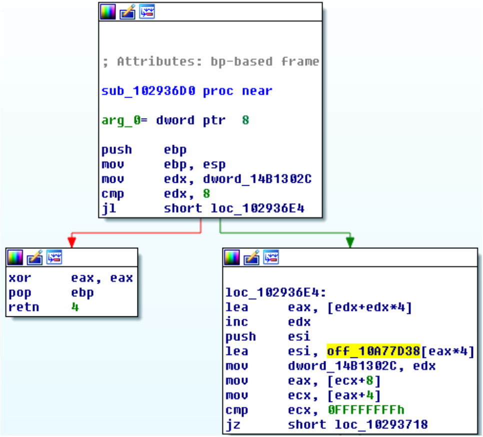
- We have a static pointer in .bss section
- We have to find its location

C_BasePlayer.cpp

```
static C_BasePlayer *s_pLocalPlayer = NULL;
```

Sigs

- Game client is often updated
- Offsets change
- We don't want to waste time



```
; Attributes: bp-based frame

sub_102936D0 proc near

arg_0= dword ptr  8

push    ebp
mov     ebp, esp
mov     edx, dword_14B1302C
cmp     edx, 8
jl      short loc_102936E4
```

```
xor     eax, eax
pop     ebp
retn    4
```

```
loc_102936E4:
lea     eax, [edx+edx*4]
inc     edx
push    esi
lea     esi, off_10A77D38[eax*4]
mov     dword_14B1302C, edx
mov     eax, [ecx+8]
mov     ecx, [eax+4]
cmp     ecx, 0FFFFFFFFh
jz      short loc_10293718
```

Figure: LocalPlayer offset

Instructions

```
lea esi, [eax * 4 + XXXX]
mov [XXXX], edx
mov eax, [ecx + 0x8]
mov ecx, [eax + 0x4]
```

Signature

```
const uint8_t sigs[] =
{
    0x8D, 0x34, 0x85, 0x00, 0x00, 0x00, 0x00,
    0x89, 0x15, 0x00, 0x00, 0x00, 0x00,
    0x8B, 0x41, 0x08,
    0x8B, 0x48, 0x00
};
```

Protecting offsets

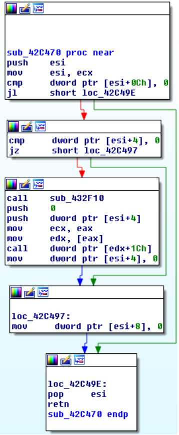
- Some cheaters want to protect offsets by packing their cheat
- IAT is destroyed you can't hook WPM
- But you can still hook the native API
- WriteProcessMemory uses NtWriteVirtualMemory in ntdll

VAC

- Several modules
- Not in FS
- Loaded by steamservices.exe
- Encrypted in .data
- manuel mapped into steam.exe
- Not loaded in the same time
- No information available when you get banned
- Bans are delayed
- No kernel module

Dump

1. Put a BP in steamservices.exe just before it injects steam.exe
2. Find where VAC is located in .data
3. Dump



```
sub_42C470 proc near
push    esi
mov     esi, ecx
cmp     dword ptr [esi+0Ch], 0
jl      short loc_42C49E
```

```
cmp     dword ptr [esi+4], 0
jz      short loc_42C497
```

```
call    sub_432F10
push    0
push    dword ptr [esi+4]
mov     ecx, eax
mov     edx, [eax]
call    dword ptr [edx+1Ch]
mov     dword ptr [esi+4], 0
```

```
loc_42C497:
mov     dword ptr [esi+8], 0
```

```
loc_42C49E:
pop     esi
retn
sub_42C470 endp
```

Figure: Dump VAC IDA

Behaviour

- Checksums game binaries
- Checks if you disabled DSE on Windows x64
- Checks if you hooked stuff in kernel32.dll
- It was checking your DNS cache
- Read memory of process which opened a handle on csgo.exe
- It looks for known public injectors

Loaders

- VAC uses mainly signatures
- If you release a public cheat it will be detected soon
- Use a loader to make generated code unique
- Junk code addition
- Change order of structs
- String encryption

String encryption

- VAC is also scanning your strings litterals
- Don't forget to encrypt them

How

- m_flFlashMaxAlpha a float between [0.0f, 255.0f]
- Create a thread which check whether its value is > 0.0f
- Write 0.0f

Internal

- Glow effect since Source SDK v2013
- Handled with a GlowManager
- 2 boolean values to set
- You can set the color
- The engine will use stencil buffer to show a glow effect around entities models

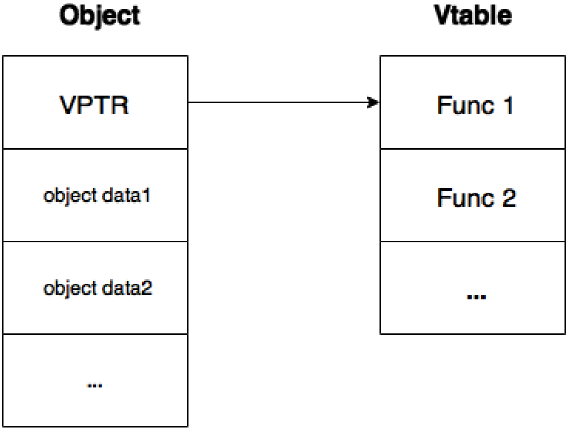


Figure: VMT

PaintTraverse hook

- g_pVGuiPanel object
- Method PaintTraverse
- Method 41 in the vtable of g_VGuiPanel
- thiscall calling convention

External

1. Run game in windowed mode
2. Open a transparent window
3. Draw your stuff at enemies position

TriggerBot

- m_inCrossHairId at offset 0x2410
- Attacks are handled with a boolean value

Conclusion and other stuff

- Don’t use existing toolz
- Make your cheat unique
- Kernel cheat ?