CS 460 Project: Wallhacks

He Yuchen (he34), Paul Pok Hym Ng (ppng2)

Introduction

We chose to pursue creating a simple wallhack for a game, CSGO. Although the version of the wallhack we wrote is definitely detectable (due to the memory library we are using), it is still a good proof of concept of how easy it is to hack a game albeit in a bannable and insecure manner. We sought to prove that learning to hack games has a low barrier to entry and is an epidemic which is hard to overcome in both Esports and online casual play.

Concept

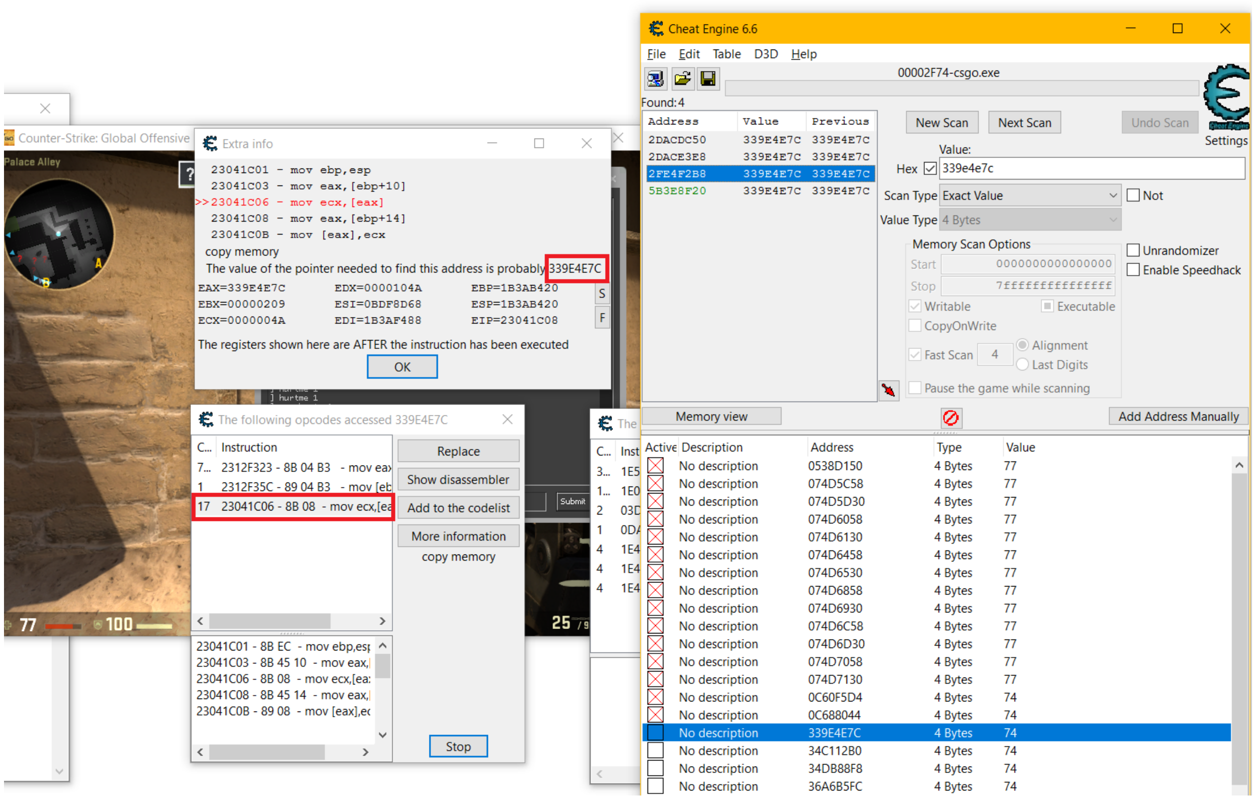
The concept behind this is to exploit the CSGO demo viewer which has capabilities such as glowing characters, lines of fire, sight lines, and other information. Although the demo viewer's capabilities can only be available to the spectator of a match or be used to view VODs post match, the data structures and capabilities are still loaded into the memory for every player on the server. We aim to edit values in the memory to enable the glow of a player, through walls, in order to create a rudimentary wallhack which is essentially given to us by the developers themselves.

Now the question is how do we find the location of these values and how do we edit them? The first question can be answered by using Cheat Engine to view the memory while the game is running and the second question can be answered by either writing your own memory access library or using someone else's (which we did).

Firstly, we learned that each struct and function used in a program has a signature and each of these signatures has a static pointer upon loading the game. Therefore even though the actual location of the value we need to change is dynamically allocated upon the start of a game, using static pointers we can walk our way back to the data structure.

Steps:

1. Find a way to change a value to and scan to find the new updated video
    a. In CSGO this would be searching for health 100, and using hurtme function to lower the health and rescan
2. Once we have narrowed down the number of possible memory addresses that have the updated health value try and observe the what assembly instructions show up. This means that something has written the new value either to screen or to the location
    a. New assembly instruction has been highlighted in red
    b. Possible pointer location has been circled in red



```
23041C01 - mov ebp,esp
23041C03 - mov eax,[ebp+10]
>>23041C06 - mov ecx,[eax]
23041C08 - mov eax,[ebp+14]
23041C0B - mov [eax],ecx
copy memory
The value of the pointer needed to find this address is probably 339E4E7C
EAX=339E4E7C      EDX=0000104A      EBP=1B3AB420
EBX=00000209      ESI=0BDF8D68      ESP=1B3AB420
ECX=0000004A      EDI=1B3AF488      EIP=23041C08
The registers shown here are AFTER the instruction has been executed
```

C...	Instruction
7...	2312F323 - 8B 04 B3 - mov ea
1	2312F35C - 89 04 B3 - mov [eb
17	23041C06 - 8B 08 - mov ecx,[ea

```
23041C01 - 8B EC  - mov ebp,esp
23041C03 - 8B 45 10  - mov eax,
23041C06 - 8B 08  - mov ecx,[ea
23041C08 - 8B 45 14  - mov eax,
23041C0B - 89 08  - mov [eax],ec
```

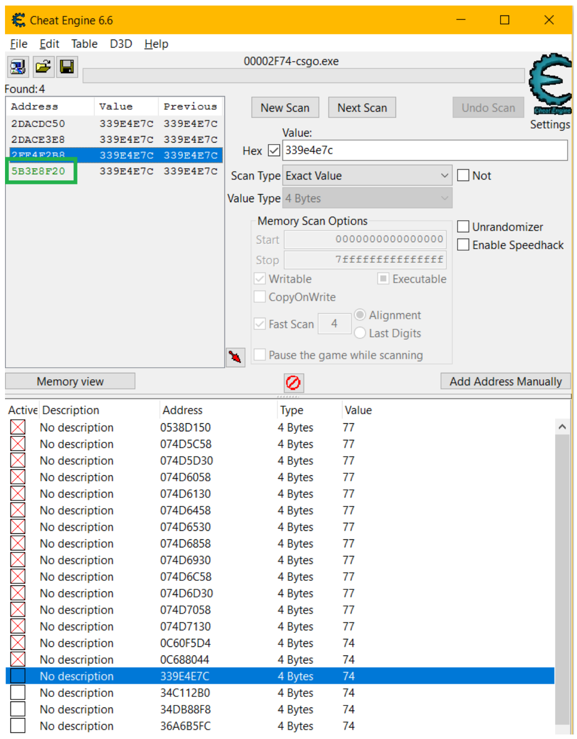
Found:4

Address	Value	Previous
2DACDC50	339E4E7C	339E4E7C
2DACE3E8	339E4E7C	339E4E7C
2FE4F2B8	339E4E7C	339E4E7C
5B3E8F20	339E4E7C	339E4E7C

Value: Hex [✓] 339e4e7c
Scan Type: Exact Value
Value Type: 4 Bytes
Memory Scan Options
Start 0000000000000000
Stop 7fffffffffffffff
[✓] Writable [■] Executable [ ] CopyOnWrite
[✓] Fast Scan 4 (● Alignment) Last Digits
[ ] Pause the game while scanning
[ ] Not [ ] Unrandomizer [ ] Enable Speedhack

Active	Description	Address	Type	Value
☒	No description	0538D150	4 Bytes	77
☒	No description	074D5C58	4 Bytes	77
☒	No description	074D5D30	4 Bytes	77
☒	No description	074D6058	4 Bytes	77
☒	No description	074D6130	4 Bytes	77
☒	No description	074D6458	4 Bytes	77
☒	No description	074D6530	4 Bytes	77
☒	No description	074D6858	4 Bytes	77
☒	No description	074D6930	4 Bytes	77
☒	No description	074D6C58	4 Bytes	77
☒	No description	074D6D30	4 Bytes	77
☒	No description	074D7058	4 Bytes	77
☒	No description	074D7130	4 Bytes	77
☒	No description	0C60F5D4	4 Bytes	74
☒	No description	0C688044	4 Bytes	74
☐	No description	339E4E7C	4 Bytes	74
☐	No description	34C112B0	4 Bytes	74
☐	No description	34DB88F8	4 Bytes	74
☐	No description	36A6B5FC	4 Bytes	74

3. After we have this pointer we can search for that pointer and see what static addresses access it.
   a. This would show up in green in the cheat engine listing



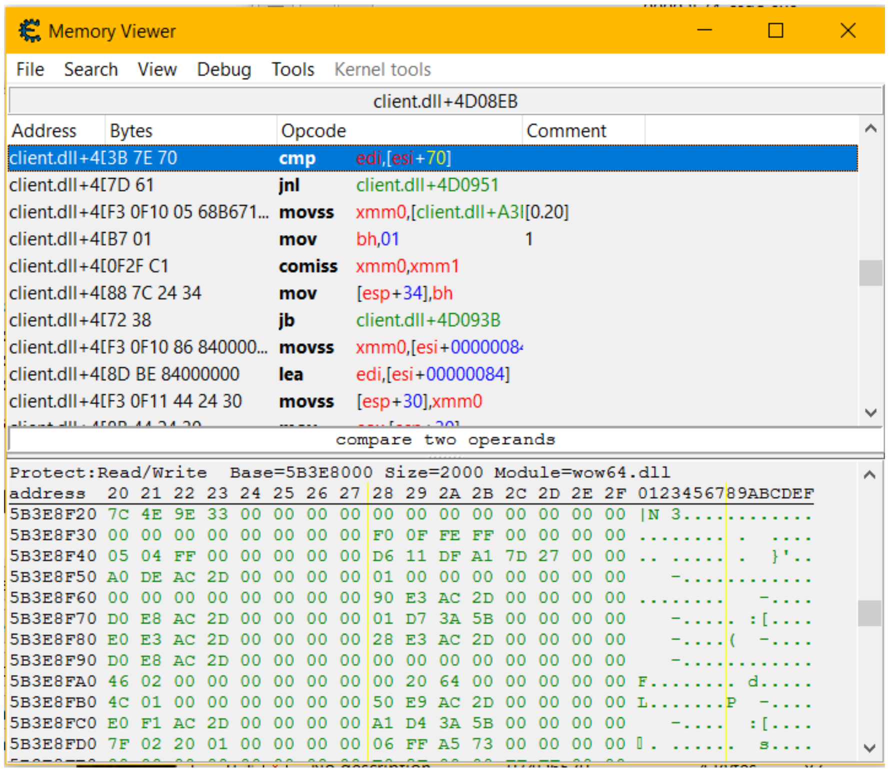
Found:4

Address	Value	Previous
2DACDC50	339E4E7C	339E4E7C
2DACE3E8	339E4E7C	339E4E7C
2FE4F2B8	339E4E7C	339E4E7C
5B3E8F20	339E4E7C	339E4E7C

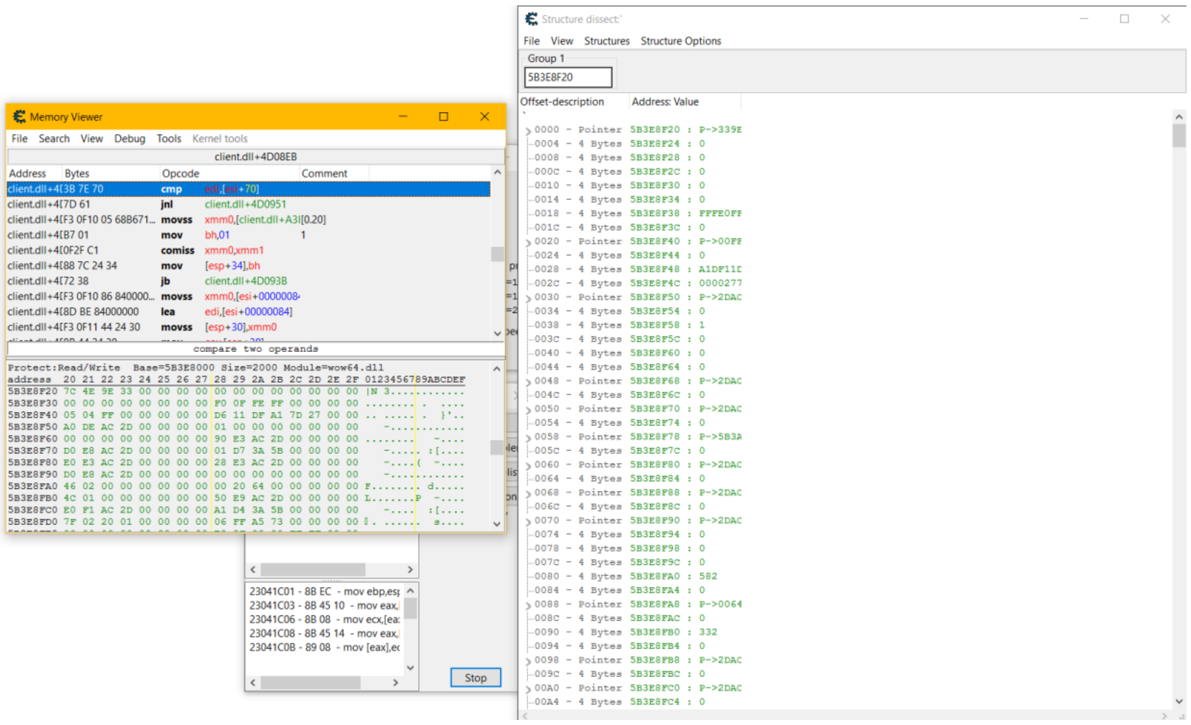
Value: Hex [✓] 339e4e7c
Scan Type: Exact Value
Value Type: 4 Bytes
Memory Scan Options
Start 0000000000000000
Stop 7fffffffffffffff
[✓] Writable [■] Executable [ ] CopyOnWrite
[✓] Fast Scan 4 (● Alignment) Last Digits
[ ] Pause the game while scanning
[ ] Not [ ] Unrandomizer [ ] Enable Speedhack

Active	Description	Address	Type	Value
☒	No description	0538D150	4 Bytes	77
☒	No description	074D5C58	4 Bytes	77
☒	No description	074D5D30	4 Bytes	77
☒	No description	074D6058	4 Bytes	77
☒	No description	074D6130	4 Bytes	77
☒	No description	074D6458	4 Bytes	77
☒	No description	074D6530	4 Bytes	77
☒	No description	074D6858	4 Bytes	77
☒	No description	074D6930	4 Bytes	77
☒	No description	074D6C58	4 Bytes	77
☒	No description	074D6D30	4 Bytes	77
☒	No description	074D7058	4 Bytes	77
☒	No description	074D7130	4 Bytes	77
☒	No description	0C60F5D4	4 Bytes	74
☒	No description	0C688044	4 Bytes	74
☐	No description	339E4E7C	4 Bytes	74
☐	No description	34C112B0	4 Bytes	74
☐	No description	34DB88F8	4 Bytes	74
☐	No description	36A6B5FC	4 Bytes	74

4. Finally we can then access the data structure by viewing the memory at that address
    a. First we go through the memory viewer and then we select the function dissect the data at this memory address
    b. Then we can use the interface to load everything into a human readable struct form with each variable taking 4096 bytes



```
client.dll+4D08EB
Address                 Bytes            Opcode   Comment
client.dll+4[3B 7E 70                    cmp      edi,[esi+70]
client.dll+4[7D 61                       jnl      client.dll+4D0951
client.dll+4[F3 0F10 05 68B671...        movss    xmm0,[client.dll+A3I[0.20]
client.dll+4[B7 01                       mov      bh,01              1
client.dll+4[0F2F C1                     comiss   xmm0,xmm1
client.dll+4[88 7C 24 34                 mov      [esp+34],bh
client.dll+4[72 38                       jb       client.dll+4D093B
client.dll+4[F3 0F10 86 840000...        movss    xmm0,[esi+0000008
client.dll+4[8D BE 84000000              lea      edi,[esi+00000084]
client.dll+4[F3 0F11 44 24 30            movss    [esp+30],xmm0
compare two operands
Protect:Read/Write  Base=5B3E8000 Size=2000 Module=wow64.dll
address   20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 0123456789ABCDEF
5B3E8F20 7C 4E 9E 33 00 00 00 00 00 00 00 00 00 00 00 00 |N 3............
5B3E8F30 00 00 00 00 00 00 00 00 F0 0F FE FF 00 00 00 00 ........ .   ....
5B3E8F40 05 04 FF 00 00 00 00 00 D6 11 DF A1 7D 27 00 00 .. ..... .  }'..
5B3E8F50 A0 DE AC 2D 00 00 00 00 01 00 00 00 00 00 00 00    -............
5B3E8F60 00 00 00 00 00 00 00 00 90 E3 AC 2D 00 00 00 00 ........    -....
5B3E8F70 D0 E8 AC 2D 00 00 00 00 01 D7 3A 5B 00 00 00 00    -.....  :[....
5B3E8F80 E0 E3 AC 2D 00 00 00 00 28 E3 AC 2D 00 00 00 00    -....(   -....
5B3E8F90 D0 E8 AC 2D 00 00 00 00 00 00 00 00 00 00 00 00    -............
5B3E8FA0 46 02 00 00 00 00 00 00 00 20 64 00 00 00 00 00 F........ d.....
5B3E8FB0 4C 01 00 00 00 00 00 00 50 E9 AC 2D 00 00 00 00 L.......P  -....
5B3E8FC0 E0 F1 AC 2D 00 00 00 00 A1 D4 3A 5B 00 00 00 00    -....   :[....
5B3E8FD0 7F 02 20 01 00 00 00 00 06 FF A5 73 00 00 00 00 . .  .....   s....
```


Memory Viewer
File Search View Debug Tools Kernel tools
Structure dissect
File View Structures Structure Options
Group 1
Stop

```
client.dll+4D08EB
Address                 Bytes            Opcode   Comment
client.dll+4[3B 7E 70                    cmp      edi,[esi+70]
client.dll+4[7D 61                       jnl      client.dll+4D0951
client.dll+4[F3 0F10 05 68B671...        movss    xmm0,[client.dll+A3I[0.20]
client.dll+4[B7 01                       mov      bh,01              1
client.dll+4[0F2F C1                     comiss   xmm0,xmm1
client.dll+4[88 7C 24 34                 mov      [esp+34],bh
client.dll+4[72 38                       jb       client.dll+4D093B
client.dll+4[F3 0F10 86 840000...        movss    xmm0,[esi+0000008
client.dll+4[8D BE 84000000              lea      edi,[esi+00000084]
client.dll+4[F3 0F11 44 24 30            movss    [esp+30],xmm0
compare two operands
Protect:Read/Write  Base=5B3E8000 Size=2000 Module=wow64.dll
address   20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 0123456789ABCDEF
5B3E8F20 7C 4E 9E 33 00 00 00 00 00 00 00 00 00 00 00 00 |N 3............
5B3E8F30 00 00 00 00 00 00 00 00 F0 0F FE FF 00 00 00 00 ........ .   ....
5B3E8F40 05 04 FF 00 00 00 00 00 D6 11 DF A1 7D 27 00 00 .. ..... .  }'..
5B3E8F50 A0 DE AC 2D 00 00 00 00 01 00 00 00 00 00 00 00    -............
5B3E8F60 00 00 00 00 00 00 00 00 90 E3 AC 2D 00 00 00 00 ........    -....
5B3E8F70 D0 E8 AC 2D 00 00 00 00 01 D7 3A 5B 00 00 00 00    -.....  :[....
5B3E8F80 E0 E3 AC 2D 00 00 00 00 28 E3 AC 2D 00 00 00 00    -....(   -....
5B3E8F90 D0 E8 AC 2D 00 00 00 00 00 00 00 00 00 00 00 00    -............
5B3E8FA0 46 02 00 00 00 00 00 00 00 20 64 00 00 00 00 00 F........ d.....
5B3E8FB0 4C 01 00 00 00 00 00 00 50 E9 AC 2D 00 00 00 00 L.......P  -....
5B3E8FC0 E0 F1 AC 2D 00 00 00 00 A1 D4 3A 5B 00 00 00 00    -....   :[....
5B3E8FD0 7F 02 20 01 00 00 00 00 06 FF A5 73 00 00 00 00 . .  .....   s....
```

```
23041C01 - 8B EC  - mov ebp,es
23041C03 - 8B 45 10  - mov eax,
23041C06 - 8B 08  - mov ecx,[ea
23041C08 - 8B 45 14  - mov eax,
23041C0B - 89 08  - mov [eax],ec
```

```
5B3E8F20
Offset-description   Address: Value
0000 - Pointer 5B3E8F20 : P->339E
0004 - 4 Bytes 5B3E8F24 : 0
0008 - 4 Bytes 5B3E8F28 : 0
000C - 4 Bytes 5B3E8F2C : 0
0010 - 4 Bytes 5B3E8F30 : 0
0014 - 4 Bytes 5B3E8F34 : 0
0018 - 4 Bytes 5B3E8F38 : FFFE0FF
001C - 4 Bytes 5B3E8F3C : 0
0020 - Pointer 5B3E8F40 : P->00FF
0024 - 4 Bytes 5B3E8F44 : 0
0028 - 4 Bytes 5B3E8F48 : A1DF11D
002C - 4 Bytes 5B3E8F4C : 0000277
0030 - Pointer 5B3E8F50 : P->2DAC
0034 - 4 Bytes 5B3E8F54 : 0
0038 - 4 Bytes 5B3E8F58 : 1
003C - 4 Bytes 5B3E8F5C : 0
0040 - 4 Bytes 5B3E8F60 : 0
0044 - 4 Bytes 5B3E8F64 : 0
0048 - Pointer 5B3E8F68 : P->2DAC
004C - 4 Bytes 5B3E8F6C : 0
0050 - Pointer 5B3E8F70 : P->2DAC
0054 - 4 Bytes 5B3E8F74 : 0
0058 - Pointer 5B3E8F78 : P->5B3A
005C - 4 Bytes 5B3E8F7C : 0
0060 - Pointer 5B3E8F80 : P->2DAC
0064 - 4 Bytes 5B3E8F84 : 0
0068 - Pointer 5B3E8F88 : P->2DAC
006C - 4 Bytes 5B3E8F8C : 0
0070 - Pointer 5B3E8F90 : P->2DAC
0074 - 4 Bytes 5B3E8F94 : 0
0078 - 4 Bytes 5B3E8F98 : 0
007C - 4 Bytes 5B3E8F9C : 0
0080 - 4 Bytes 5B3E8FA0 : 582
0084 - 4 Bytes 5B3E8FA4 : 0
0088 - Pointer 5B3E8FA8 : P->0064
008C - 4 Bytes 5B3E8FAC : 0
0090 - 4 Bytes 5B3E8FB0 : 332
0094 - 4 Bytes 5B3E8FB4 : 0
0098 - Pointer 5B3E8FB8 : P->2DAC
009C - 4 Bytes 5B3E8FBC : 0
00A0 - Pointer 5B3E8FC0 : P->2DAC
00A4 - 4 Bytes 5B3E8FC4 : 0
```

5. Observe the struct when you change something such as jumping and see if a flag has been changed
6. Repeat for all desired values

Testing

To begin with we decided to write a bunny hop script. A bunny hop abuses the falling acceleration of a player character and jumps immediately after landing in order to preserve this new found speed. This was used often in older games such as Quake which allowed for this type of movement. However in more recent game such as CSGO such movement has been deemed hacking and unfair.

This uses all the same ideas as the wall hack but is definitely much easier to implement as we only need to find one address the local player struct which stores a flag whether a player is jumping or not.

The basic idea is to check this flag. Whenever this flag is flipped we need to trigger the keyboard to change its status to not pressed and again when its flipped it will be flipped back to pressed.

The reason we need to do this is because the game was coded to prevent players from holding down spacebar and continually triggering the jump. Therefore in order to bunny hop one would need to press and let go of the space bar at very specific times in order to preserve the momentum which is incredibly hard thus a script.

Discussion Memory

This glow will always be consistently in the game as casters should have access to this feature. The reason why wallhacks continually need to be updated are not only because the developers move these offset around in memory in order to void previous working builds of wallhacks but also to patch the ways that were used to access memory. Simply accessing memory is dangerous as checking whether memory has been written is much easier than checking whether it's been read.

However developing an algorithm or library which allows us to read and write memory without being caught would a tall order in itself, and we would probably need much time and research in order to accomplish therefore in the interest of time we went with a known, detected, but working library known as VAMemory.

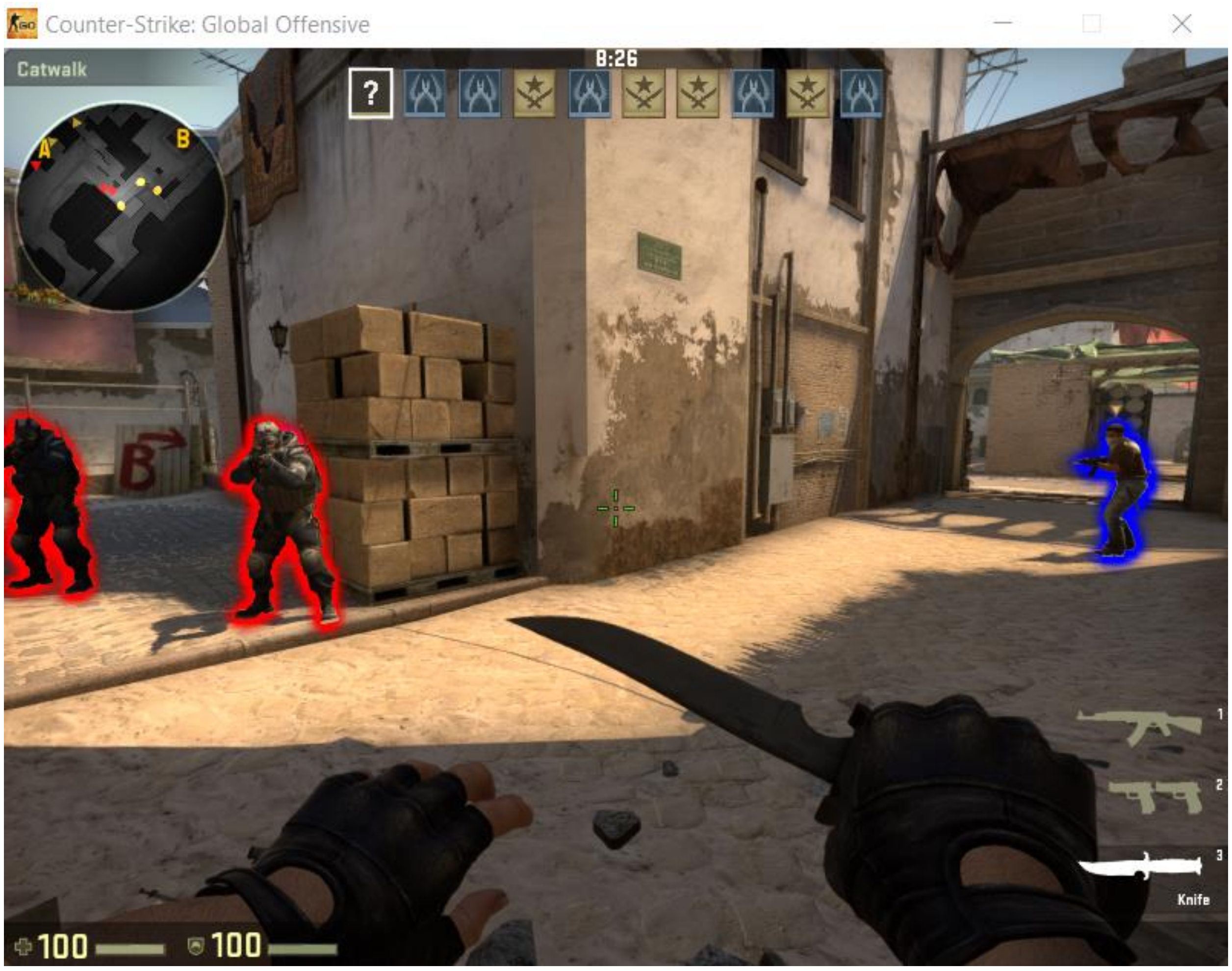
Proof of Concept

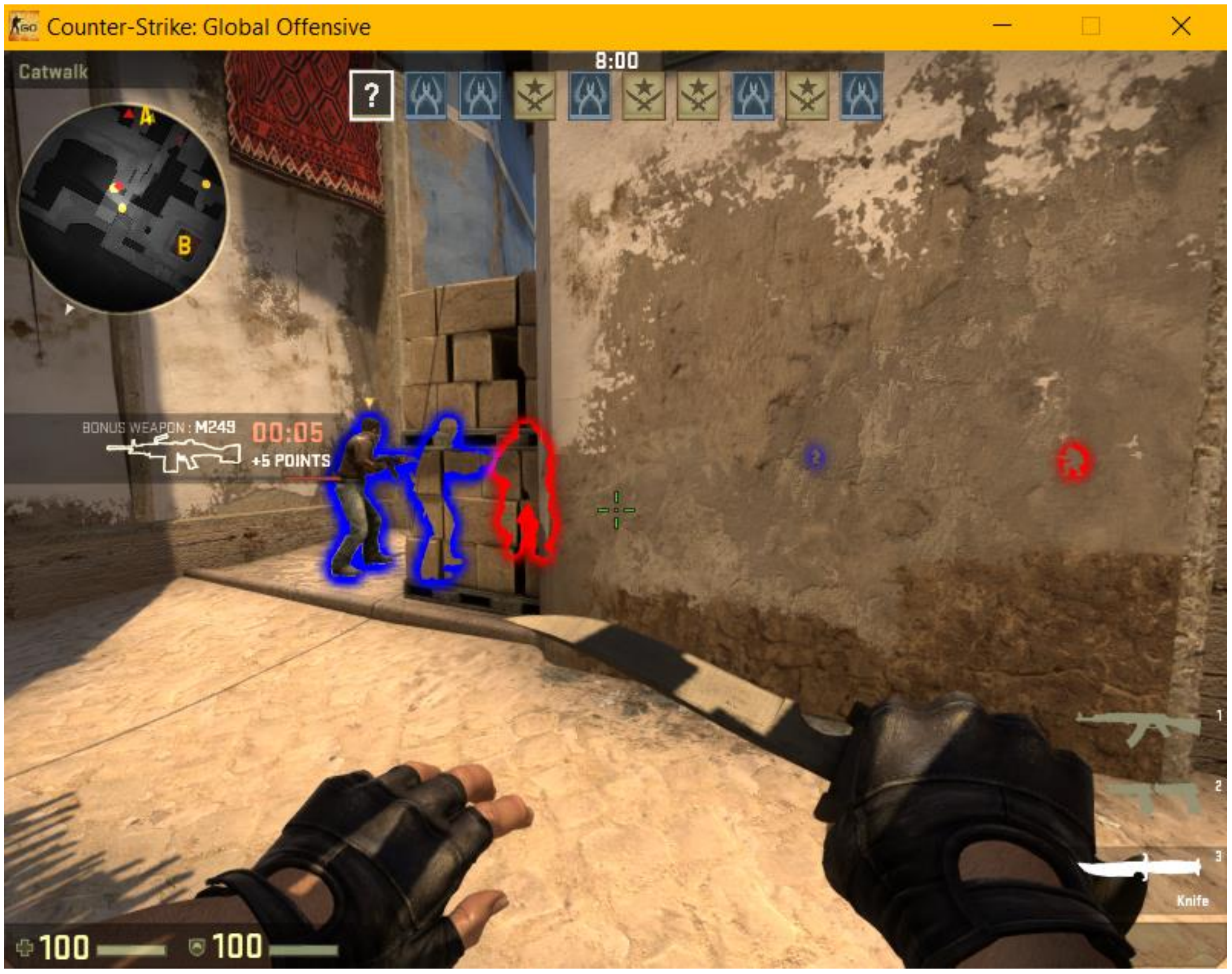
The major difference between the bunny hop experiment and the wall hack is that the wall hack requires access to more information. Instead of only the local player struct, it also needs access to …

1. Local Player: Your player information
2. Local Entity: Contains information about teams
3. Glow Struct: Toggles the glow effect

These were found using the same trial and error in the same way mentioned above. Local player is used to check which team you're on to compare to the others. Local Entity is used to check the team of all the other players on the server. If they match set the color of their outline to blue and if they don't match set the color of their outline to red.

Finally the glow struct is used to actually set the players colors.





Conclusion

We have shown that it is relatively easy to get into game hacking and create something that works. However what works is not necessarily impervious to detection. Therefore the art of game hacking is in figuring out how to access and write memory without being banned. It will always be race between the hackers and the developers leading to innovation and giving rise to an ever thriving industry.