Using RME to directly edit memory

Introduction:

RME (Remote Memory Editing) is an exploit in COD that allows the RGH/JTAG user to write data into the host's memory. This can change how the game works for individual clients or even every client.

There are two mayor ways to change memory with RME; Direct Memory Editing and sending DVAR commands.

To add any new RME functions one needs to edit the contents of "RME.cpp" and "RME.h" aswell as "Globals.h"

sending DVAR commands

Sending DVAR commands is far simpler than editing the memory directly as we don't need to calculate extra which memory address we want to edit. WE can simply use any of the functions that send a DVAR command and take it as a general template. Sending these commands, however, isn't guaranteed to work as a lot of commands are cheat protected. If we were to make the host send, basically in our name thanks to RME, a cheat protected command, the host would be kicked out of the game because he triggered the cheat protection. We would need to somehow bypass the cheat protection for the host to make these kinds of cheat protected commands work. As of today, this isn't possible as the cheat protection offset is above the range that RME is normally able to edit. One needs to find some way to either edit those offsets above the RME range, or one needs to find some other bypass to circumvent the cheat protection.

General look of a DVAR command through RME

```
char* buf = "cg_drawGun 0\n"; //this command is cheat protected

g_Netchan.write_string(decryptDWORD(Security->addrs.cmdInfoAddrs), buf);

g_Netchan.write_int(decryptDWORD(Security->addrs.cmdInfoSizeAddrs) + 0x8,
strlen(buf));
```

Direct Memory Editing

To edit memory from the host directly we need to use a formula to calculate where the offset, which we want to change, is on the host. The formula consists of "gClient_Size", "gClient", "clientNumber", "valueToInsert" and "restToOffset". The first three of these variables exist inside the code, the last one is the one that needs to be calculated. "restToOffset" is the most important value in any direct memory edit when using RME. This hexadecimal value will decide which memory address you're editing. If "restToOffset" is either too big or too small, it may result in a crash.

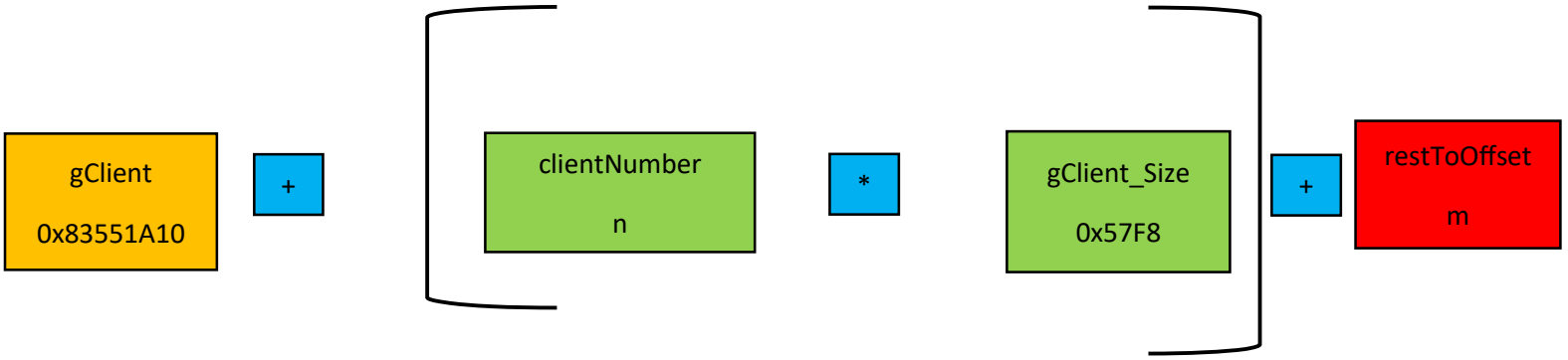
Size for each variable

gClient	clientNumber	gClient_Size	restToOffset
Hex: 0x83551A10 Decimal: 2203392528	Dynamic int between 0 and 17	Hex: 0x57F8 Decimal: 22520	Dynamic Hex number to get to the desired offset, if too big > edit wrong memory > possible crash

How a direct memory edit with RME looks

```
g_Netchan.write_short((gClient(i) + 0x54E0), 0xF2);
```

The general formula to get the correct memory STARTING addresses for each host



Where one starts editing memory from, if n = 0

0x83551A10

Where one starts editing memory from, if n = 1

0x83557208

Explanation

We need to calculate for each player the place where they have their data saved on the host. “Player state” can supposedly only be edited with the current way on how RME works.

Before we can edit any memory, we need to find an offset which we want to edit. If we found a value which we want to edit, then we need to calculate through the formula how much we need to get to the offset from our starting point.

The base starting address is “**0x83551A10**” if the client number is 0. If you want to edit the memory for a different client, you need to multiply the clientNumber*gClient_Size. gClient_Size is a static value and won’t need to be changed.

How to find offsets through XCE

To find values for editing we can use “XCE” to dump the memory from our game while being inside the running game. After the dump is done, which takes a few minutes, we can search for a certain value and will find any offset that has that value saved. Let’s assume we want to change our ammo amount; we would look for ammo we have currently, convert that number to hexadecimal and then search for that hexadecimal value in our dump. After finding all the offsets that have such a value stored, we would then shoot to change the ammo value from our gun. Afterwards we would search for the new value, now we would get all the results that went from the old value to the new value > will have way fewer results/just one. If we have only a few results remaining we would then use “Peek Poker” to change the offsets value to some other value and see if our ammo value, which we want to find, is changing or not.

How to find offsets through “Peek Poker”

Start “Peek Poker”, connect to your console and then go to the starting address for the RME Exploit while being in a custom game. After that just change any value of any of the offsets that you see before you “Poke” the change into the memory. See if anything changed that you wanted to change. Repeat until you found anything. If you find an offset that does something cool, then write down the exact offset that it is. Now you need to calculate “restToOffset” to make your direct RME edit complete. As said before, this value is the most important part. If it’s too high or too low, you will edit the wrong memory and may crash your console.

The general steps to calculate “restToOffset”

1. Convert the offset you want to edit to decimal
2. Convert the base starting address to decimal
3. Calculate: your_offset_base starting address
4. Convert the result into hexadecimal

Examples on calculating “restToOffset”

We want to edit the following offset, which is used for one ammo value:

Offset we want to edit: 0x83551E4E

1. 0x83551E4E = 2203393614
2. 0x83551A10 = 2203392528
3. 2203393614 – 2203392528 = 1.086
4. [Decimal] 1086 = 0x43E [Hexadecimal]

[input RME command]

Alternative way to calculate the “restToOffset” value

Use the script “restToOffset-Calculator.py” or “restToOffset.exe” to calculate the “restToOffset” faster.

Both of these scripts are open source, all code can be seen in the “Software-Source” folder.

Links

Hex to Decimal calculator: https://www.rapidtables.com/convert/number/hex-to-decimal.html