Anti-Cheat Networking

Understanding how an anti-cheat communicates will help you navigate around it more efficiently.

- Client>Server Communication

Client>Server Communication

There is no singular function that is responsible for this task. In Apex, we rather have multiple of those working together.

> It is very unlikely that respawn is using just one function for their telemetry exchange. I have encountered at least 1 different function that also submits telemetry in a similar way to PushViolation. If you're watching their violation queues, you need to be aware that in certain scenarios they will send a NetMessage directly. In such a case, your logging would be incomplete.

Let's get started with the most common function, **R5AC::PushViolation:**

```
void __fastcall R5AC::PushViolation(
        const __m128i *pszIdentifierStr,
        char Severity,
        __int64 ExtraDataPtr,
        __int64 ExtraDataLen);
```

In case of client-side anomalies being detected, it will be invoked like this:

```
LBL_ON_ABNORMALITY_FOUND:
            v119 = v116();
            v120 = 0;
            v121 = 0i64;
            do
            {
              v122 = *(_BYTE *)(v121 + 41489128); // these are just data and key, and the RVA can be computed with analysis of a runtime dump.
              ++v120;
              v123 = *(_BYTE *)(v121 + 27928337);
              v188[++v121 + 15] = v123 ^ v122;
            }
            while ( v120 < 0xC ); // C-String encryption using simple xor
            sprintf_s_2(Buffer, 0x104ui64, Format, v119);
```

```
        }
      }
      R5AC::PushViolation(v200, 1, v177, (__int64)Buffer);
```

It's purpose is to enqueue a new violation record into a list that is fetched and emptied by numerous callers, all originating from **r5apex_dx12.exe.** These are to be considered slaves because they simply check if there is any messages pending, use CLC_AntiCheat virtual method table to setup a new netmessage in source engine, and then puts the data of the anti-cheat violation/message into the netmsg and sends it to the game server.

```
if ( gpNetChan )
  {
    v14 = vft::CLC_AntiCheatMsg;
    v15 = 0;
    v17 = 0i64;
    v5 = 5;
    v16 = 1;
    // send all of them in bulk
    for ( i = 0; (unsigned __int8)R5AC::PopAnticheatMsg(v20, &i); --v5 )
    {
      if ( !v5 )
        break;
      v18 = v20;
      v19 = i;
      C_NetChan::SendNetMsg(v4, &v14, 0, 0);
      C_NetChan::SendDatagram(v4, 0i64);
    }
  }
```

Additionally, there is a couple more ways for the anti-cheat to phone back home. Let me introduce you **R5AC::TrySendViolationOrQueue.** It behaves almost identically to it's predecessor, with the twist that if your game client has an active network channel instance (or in other words: an active connection to a game server is present), it will bypass it's usual queue-based system and directly send it to the game server.

> It should be noted that they no longer use any readable strings for the violation/message identifier. Instead, they will use a numeric, hexadecimal value (64-bit). Example: 0xA7F3C91D5B82E46F. This might be unfortunate because you could use these strings to automatically resolve certain game functions and also interface pointers. Sure, you could

leverage old dumps and still find most stuff through manual reversing, but I would say grabbing them automatically like this did have an appeal.

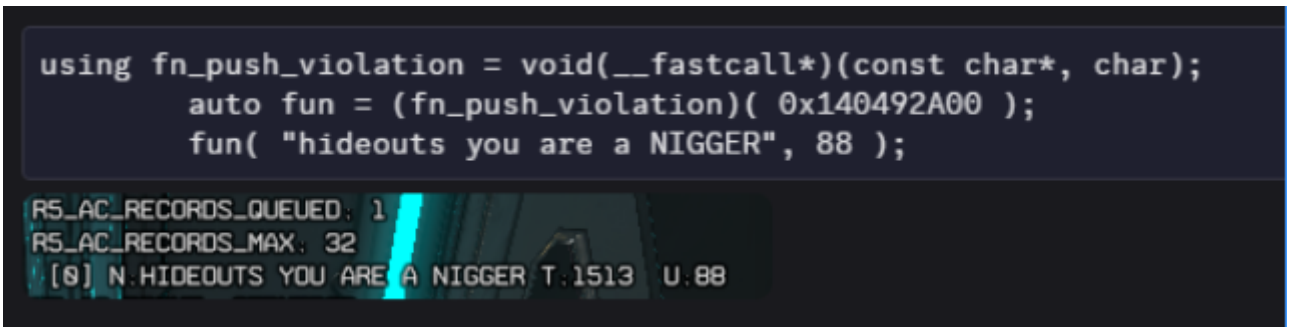
```
void __fastcall R5AC::TrySendViolationOrQueue(
        const __m128i *id_str,
        __int64 a2,
        __int64 extraData,
        const __m128i *extraDataLen)
{
  --*(_DWORD *)(v4 + 68);
  v5 = gpNetChan;
  if ( gpNetChan && (unsigned int)MEMORY[0x7FFE8D2B5550](id_str, 0i64) ==
dword_594427C )
  {
    v28 = 0;
    v9 = id_str;

    do
    {
      v10 = v9->m128i_i8[0];
      v9->m128i_i8[pszBuffer - (char *)id_str] = v9->m128i_i8[0];
      v9 = (const __m128i *)((char *)v9 + 1);
    }
    while ( v10 );

    idStrLength = -1i64;
    do
      ++idStrLength;
    while ( id_str->m128i_i8[idStrLength] );
    v12 = &pszBuffer[idStrLength + 1];
    if ( extraDataLen )
    {
      idx = -1i64;
      do
        ++idx;
      while ( extraDataLen->m128i_i8[idx] );
      R5::AllocAndCopyStruWithHdr(
        &mem,
        extraDataLen,
        idx,
        (__int64 (__fastcall **)(_QWORD, unsigned __int64, __int64))off_1AA6728);
      extraDataPtr = mem;
    }
    else
```

```
    {
      extraDataPtr = 0i64;
      mem = 0i64;
    }
    *(_DWORD *)v12 = 1;
    v12[4] = 0;
    *(_QWORD *)(v12 + 5) = extraData;
    sse_memcpy((__m128i *)(v12 + 13), extraDataPtr, (unsigned
int)(extraDataPtr[-1].m128i_i32[3] + 1));
    v15 = (_DWORD)v12 - ((unsigned int)&v27 + 880) + 14 + extraDataPtr[-
1].m128i_i32[3];
    someKey = 0x42A1110F96B5E116i64;
    v16 = 0;
    if ( v15 != 2 )
    {
      v17 = pszBuffer;
      do
      {
        ++v17;
        v18 = v16++ & 7;
        *(v17 - 1) ^= *((_BYTE *)&someKey + v18);
      }
      while ( v16 < v15 - 2 );
    }
    v26 = v15;
    *(_QWORD *)&v21 = vft::CLC_AntiCheatMsg;
    v22 = 0;
    v25 = &v28;
    v24 = 0i64;
    v23 = 1;
    C_NetChan::SendNetMsg(v5, &v21, 0, 0);
    C_NetChan::SendDatagram(v5, 0i64);
    sse_memset((__int64)&v28, 0, 0x400ui64);
    *(_QWORD *)&v21 = &off_173A878;
    if ( _InterlockedExchangeAdd(&extraDataPtr[-1].m128i_i32[2], 0xFFFFFFFF)
== 1 )
      (*(void (__fastcall **)(__int64))(extraDataPtr[-1].m128i_i64[0] +
8))(extraDataPtr[-1].m128i_i64[0]);
  }
  else
  {
    R5AC::PushViolation(id_str, 0, extraData, (__int64)extraDataLen);
  }
```

I have tried sending random stuff through this function before their switch to numeric ID's. I don't know if it ever arrived, probably not if they realistically filter for garbage. I can say that it was successfully pushed to the telemetry queue, but whether it was sent: no idea. The idea seems to be funny that you could send messages of your choice through it. What i also found interesting was the question how they process these anti-cheat messages on the server side. I obviously can't know it but i'd imagine its saved somewhere, so they have evidence and can read the raw telemetry or feed it to automation.



> “