Control Flow Analysis

R5AC is planting control flow enumeration helpers into important game mode.

Just because a module is signed, doesn't mean that R5AC will give it a pass.

It follows a strict whitelist which at the time of this writing, consisted of the following ranges.

r5apex_dx12.exe	(+1000 → +13d2000) ; '.text'
KERNEL32.DLL	(+1000 → +7f4bb) ; '.text'
ntdll.dll	(+1000 → +119f1e) ; '.text'
ntdll.dll	+11a000 → +11a592) ; 'PAGE'
ntdll.dll	(+11b000 → +11b1f9) ; 'RT'
steamnetworkingsockets.dll	(+1000 → +2f63a8) ; '.text
steam_api64.dll	(+1000 → +243de) ; '.text'
EOSSDK-Win64-Shipping.dll	(+1000 → +d8117c) ; '.text

Their whitelist is using a relatively primitive structure for describing what they deem a whitelisted memory range:

> struct WHITELISTED_REGION
>
> {
>
> u64 Begin;
>
> u64 End;
>
> };

> In more recent builds of**Apex Legends**, this list is now more complex to interpret and seems to have switched to using some computed hash of whitelisted module bases, and checking for that instead. R5AC only uses RtlCaptureStackBackTrace for examining control flow.

> In 2026, R5AC has switched to manual stack tracing via RtlCaptureContext + RtlLookupFunctionEntry + RtlVirtualUnwind.

An example of this instance of detection can be found in C_ViewRender::GetMostRecentClipTransform.

```
__int64 C_ViewRender::GetMostRecentClipTransform(__int64 a1, char frameIndex)
{
  v152 = frameIndex;  // Prime RNG (seems unrelated to stack walker)
  R5AC::Prime = 214013 * R5AC::Prime + 2531011;  // Early exit: check stack walker enablement
  if (!R5AC::IsStackWalkerEnabled)
    return *(_QWORD *)(a1 + 8i64 * v152 + 1155920);  __int64 stackFrames[2] = { 0 };
  __int128 stackData[2] = { 0 };  unsigned short numFrames = R5AC::Imp::RtlCaptureSBT(1, 5, stackData);
  if (!numFrames)
    return *(_QWORD *)(a1 + 8i64 * v152 + 1155920);  // Compute simple hash of the captured stack
  int hashA = 5381, hashB = 0;
  char* p = reinterpret_cast<char*>(stackData);
  for (size_t i = 0; i < 8 * numFrames; ++i)
  {
    unsigned char val = *p++;
    hashA = val + 33 * hashA;
    hashB = val + 65599 * hashB;
  }
  int stackHash = hashA ^ hashB;  // Lock to check hash
  MEMORY[0x7FFE8D571760](&R5AC::SoemLock, 0);
  char hasChanged = R5AC::HashmapHasChangedData((__int64)&R5::HashedCodeExecutionFrames, &stackHash);
  MEMORY[0x7FFE8D571920](&R5AC::SoemLock);  if (hasChanged != -1)
  {
    // If hash is known, return cached transform
    return *(_QWORD *)(a1 + 8i64 * v152 + 1155920);
```

```
    }    // Iterate captured frames
    for (unsigned int i = 0; i < numFrames; ++i)
    {
        __int64 frameAddr = *reinterpret_cast<__int64*>(stackData + i);
        if (!frameAddr)
            break;        // Check if frame is in whitelisted regions
        bool inWhitelist = false;
        for (int j = 0; j < R5AC::WhitelistedSectionCount; ++j)
        {
            _WHITELISTED_SEGMENT_* region = &R5AC::WhitelistedSections[j];
            if (frameAddr >= region->Begin && frameAddr <= region->End)
            {
                inWhitelist = true;
                break;
            }
        }        if (inWhitelist)
            continue;        // If not whitelisted, fetch mapped module and push
violation
        char moduleName[272] = { 0 };
        ((void(__fastcall*)(__int64, __int64*, char*, __int64))0)(-1, frameAddr,
moduleName, 260);        if (moduleName[0])
        {
            R5AC::PushViolation(nullptr, 1, frameAddr,
reinterpret_cast<__int64>(moduleName));
            return *(_QWORD *)(a1 + 8i64 * v152 + 1155920);
        }
    }    // If no violations, store new hash
    MEMORY[0x7FFE8D5790A0](&R5AC::SoemLock,
R5AC::WhitelistedSectionCount, numFrames, R5AC::WhitelistedSections);
    sub_2345E0(&R5::HashedCodeExecutionFrames, &stackHash);
    MEMORY[0x7FFE8D562C70](&R5AC::SoemLock);    return *(_QWORD *)(a1 +
8i64 * v152 + 1155920);
}
```

A little fun Fact

Some cheat developers have actually been using the aforementioned functionality against apex. See, because the API respawn uses for stack walking is resolved into an unchecked pointer in .data, anyone can just swap it out and fake the back trace buffer, or even cheaper: pretend like nothing on the stack was unwindable. As if that wasn't enough, respawn also have decided to making calls by referencing it's value across the game's entire code base.

What makes this much worse is the fact that this vulnerable pointer is referenced by numerous critical game functions. It's literally a backdoor so handy that you don't even need to hook CreateMove anymore, for example. You can just swap this pointer, check if caller (return address) is **C_Input::CreateMove**, and and that's it!

> “
> unsigned short __fastcall hk_capture_stack_back_trace(unsigned long frames_to_skip, unsigned long frames_to_capture, void **trace, void *trace_hash)
> {
> auto retaddr = BASE_OF(_ReturnAddress()) - gctx->game_base;
> g.r5ac_stack_walk_last_call = retaddr;
>
> if(g.conf.anon_mode)
> {
> *reinterpret_cast<uptr*>(gctx->game_base + gctx->offsets.string_dict_user_names) = 0;
> }
>
> if (retaddr == gctx->offsets.ret_addrs.create_move)
> {
> g.inputs.cmove.tick_begin();
>
> features::on_anonymizer();
> if (is_valid_session())
> {
> auto me = local_player();
> if (me.is_alive())
> {
> if (g.conf.rcs_strength > 0)
> {
> float rcs_strength = (static_cast<float>(g.conf.rcs_strength) / 1000.f);
> g.conf.rcs_internal_mouse_scale = rcs_strength;
> }
>
> if (g.conf.aim_assist_strength > 0)
> {
> float aa_strength = (static_cast<float>(g.conf.aim_assist_strength) / 100.f);
> g.conf.aim_assist_internal_mouse_scale = (gfloats.one - aa_strength);
> }

```
features::on_movement(me);
features::on_trigger(me);
}
}

g.inputs.cmove.tick_end();
}

return 0; /* no traces are available */
}
```

```
bool disarm_stack_walker()
 {
    size_t num_handled = 0;
    uint64_t needle = gctx->winapis.capture_stack_back_trace;
    if (needle)
    {
       uintptr_t offset = 0;
       auto data = (u8*)gctx->game_data_base;
       auto end = (u8*)(gctx->game_data_base + gctx->game_data_len);

       while (true)
       {
          auto result = stl::search(data + offset, end, (uint8_t *)&needle,
sizeof(uintptr_t));
          if (result != 0)
          {
             #if LOGGING_IS_ENABLED == 1
                LMsg(256, "[R5AC] disarming stack walk at index %i and pointer
%p", num_handled, result);
             #endif
             *reinterpret_cast<void **>(result) = hk_capture_stack_back_trace;
             ++num_handled;
             offset = result - (uintptr_t)data + 1;
          }
          else
          {
             break; // No more occurrences found
          }
       }
    }
```

```
    g.r5ac_num_stack_walks_disarmed = num_handled;

    return (num_handled > 0);
}
```

Above code was tested on retail apex legends, on both windows and linux. Since the flaw is in R5AC itself, platform mostly does not matter.

Revision #24
Created 2026-02-12 23:12:59 UTC by Admin
Updated 2026-05-06 16:13:57 UTC by DROF