



Another Way to Talk with Browser : Exploiting Chrome at Network Layer

Rong Jian, Guang Gong
360 Vulnerability Research Institute

Whoami

Rong Jian (@__R0ng)

- Security Research for 360 Vulnerability Research Institute
- Mainly focus on browser security
- Winner of Chrome category in Tianfu Cup contest 2020 / 2021



Agenda

Introduction

- ◆ Browser Networking
- ◆ Resource Loading

Code Caching in Chrome

- ◆ How code caching works
- ◆ The design flaws of code caching
- ◆ Renderer REC

QUIC Transport

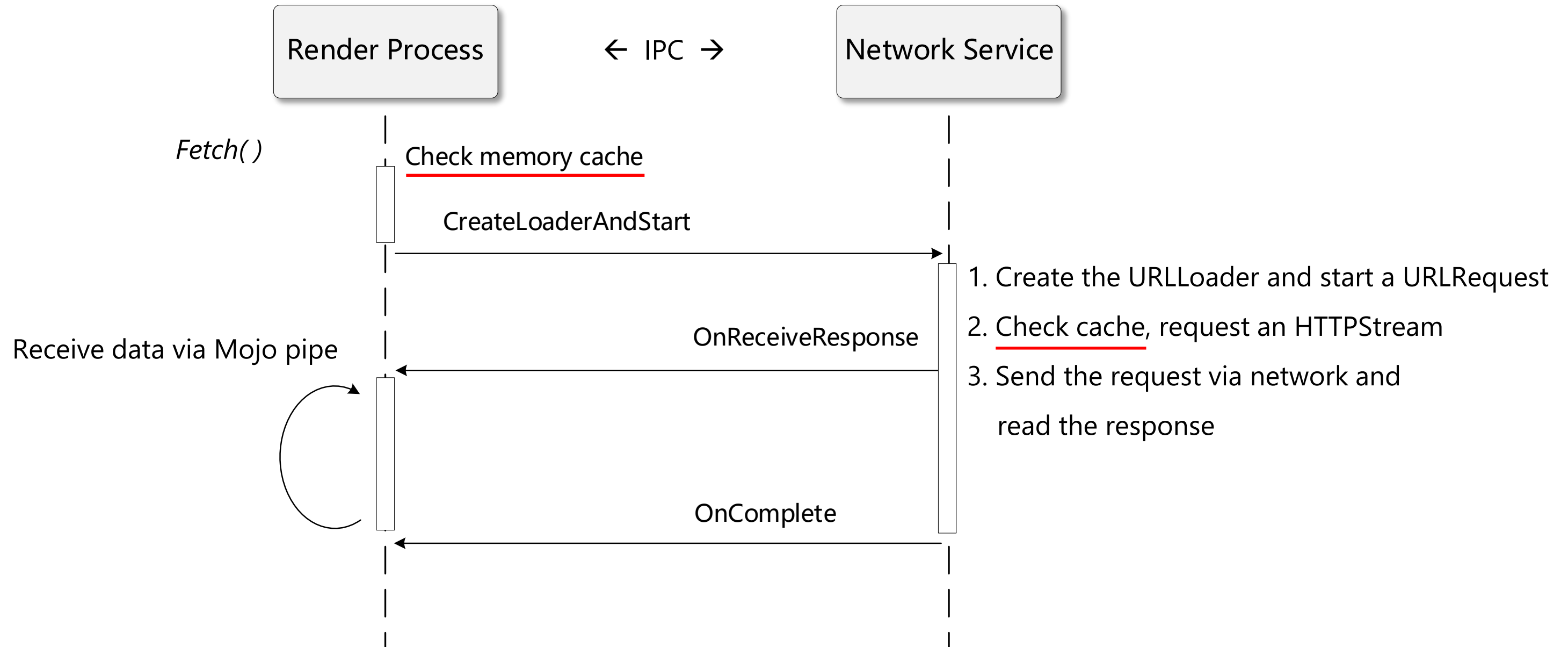
- ◆ Overview of QUIC
- ◆ UAF bug caused by unexpected server responding
- ◆ Sandbox Escape

Conclusions

Browser Networking

- ◆ Play a critical role for resource loading
- ◆ Range from high-level JavaScript APIs to management of every sockets
- ◆ Hard to parsing and processing complex and untrustworthy inputs correctly and safety

URLRequest : A simple example



Caches Everywhere

- ◆ Multiple caches differ in how they acquire, store and retain content
- ◆ Different requests can get matched by resources in different caches

Wrong cache hit?

cached img



actual img



How about Code Cache ?

Code Caching in Chrome

WebAssembly Code Caching

Cold Run

I want to fetch a .wasm resource



Compile it !



Here it is, I download it from remote server



Done



WebAssembly Code Caching

Hot Run

I want to fetch and compile that .wasm resource again !



- ◆ Fetch .wasm resource ➡ Resource Cache
- ◆ Compile the .wasm file ➡ Code Cache

Cache hit ! No need to download and compile



Name	Status	Type	Initiator	Size	Time
<input type="checkbox"/> example.wasm	200	fetch	poc.html:11	65.8 kB	42 ms
<input type="checkbox"/> example.wasm	200	fetch	poc.html:11	(disk cache)	1 ms
<input type="checkbox"/> example.wasm	200	fetch	poc.html:11	(disk cache)	2 ms
<input type="checkbox"/> example.wasm	200	fetch	poc.html:11	(disk cache)	3 ms
<input type="checkbox"/> example.wasm	200	fetch	poc.html:11	(disk cache)	1 ms

WebAssembly Code Caching

Key Questions

- a. When will the code cache be generated?
- b. What content is being cached?
- c. How a cache hit occurs?

When

WasmStreamingClient::OnModuleCompiled

```
Platform::Current()->CacheMetadata(  
   mojom::CodeCacheType::kWebAssembly,  
    KURL(response_url_),  
    response_time_,  
    serialized_data.data(), serialized_data.size());
```

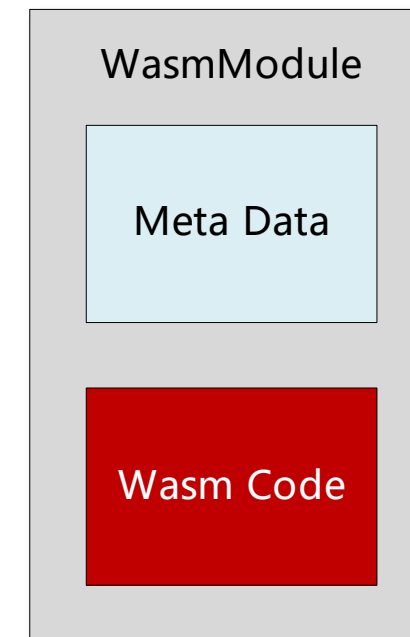
- ◆ Send data to the CodeCacheHost (in browser process)
- ◆ Data will also be stored as <Key, Value> pair in a map (in render process)

What

NativeModuleSerializer::Write

```
bool NativeModuleSerializer::Write(Writer* writer) {  
    DCHECK(!write_called_);  
    write_called_ = true;  
    WriteHeader(writer);  
    for (WasmCode* code : code_table_) {  
        if (!WriteCode(code, writer)) return false;  
    }  
    return true;  
}
```

- ◆ Not all data in WasmModule is serialized



How

ResourceFetcher::CachedResource

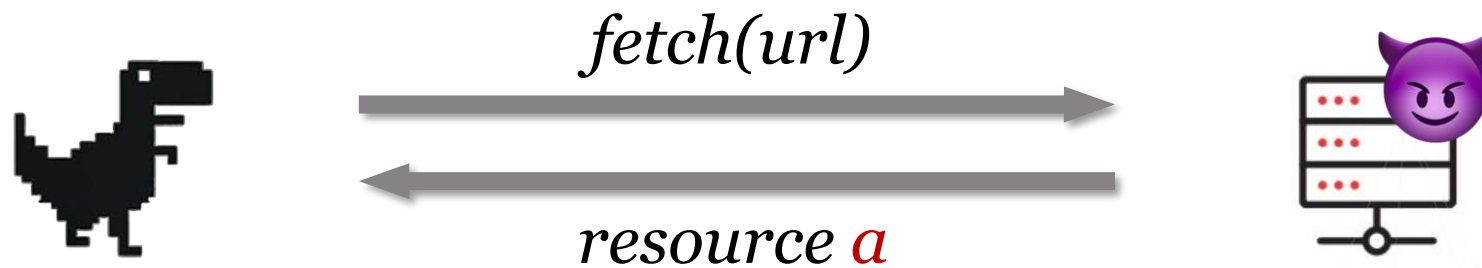
```
Resource* ResourceFetcher::CachedResource(const KURL& resource_url) const {
    if (resource_url.IsEmpty())
        return nullptr;
    KURL url = MemoryCache::RemoveFragmentIdentifierIfNeeded(resource_url);
    const WeakMember<Resource>& resource = cached_resources_map_.at(url);
    return resource.Get();
}
```



- ◆ Code Cache is associated with Resource Cache
- ◆ Happens when compiling the .wasm resource
- ◆ Simply do a map lookup by key

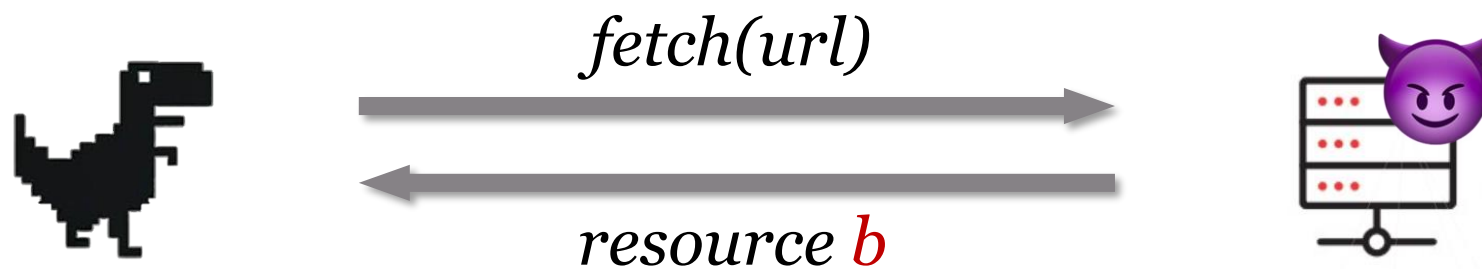
Code Cache Confusion (CVE-2020-16015)

Step 1. Fetch the wasm resource but do not compile it



◆ No code cache

Step 2. Fetch again and compile

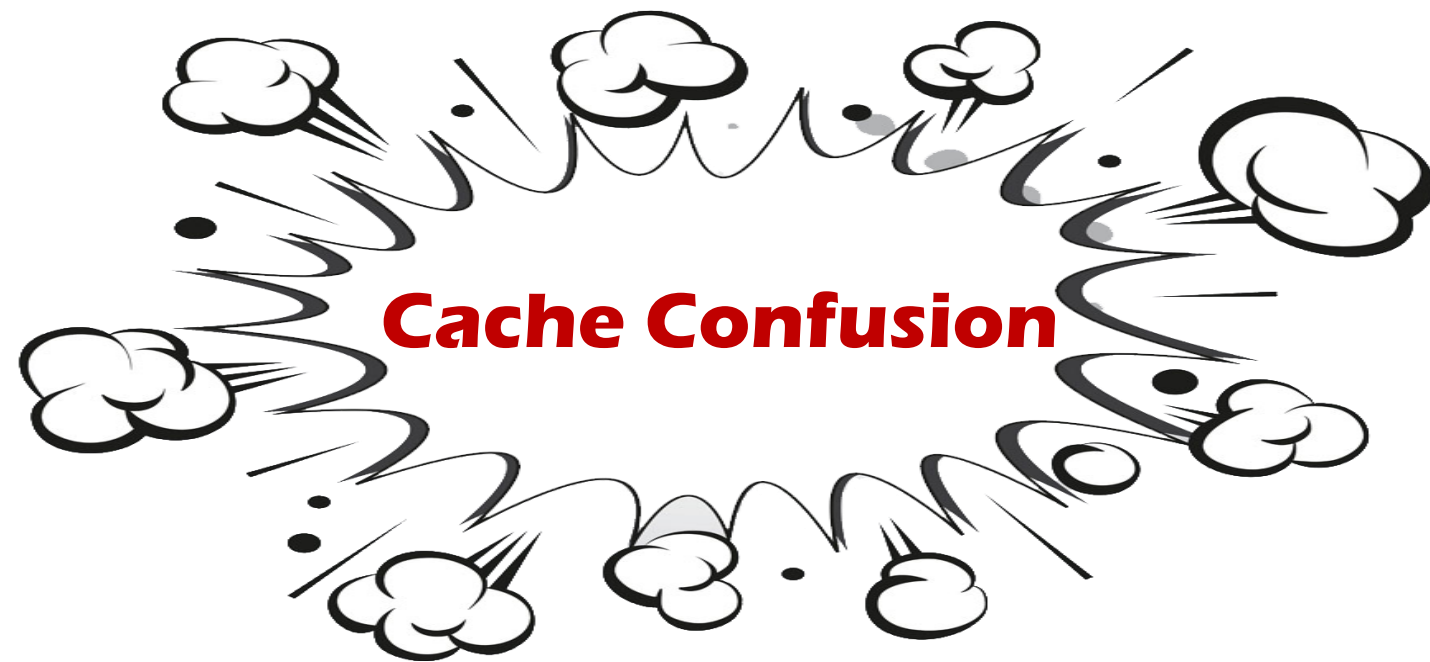


◆ Map url to cache of
resource b

Code Cache Confusion (CVE-2020-16015)

Step 3. Compile *resource a*

Because *resource a* has the same url with *resource b* ...



Fix (<https://crrev.com/c/2534570>)

- ◆ Changed the timing of determining code cache hit
 - Before: Happens when compiling the .wasm resource
 - After: Happens when responding resource request
- ◆ Changed the way to store code cache
 - Before: Can be retrieved from resource cache
 - After: Stored in the Response object if cache hits



- ◆ Record the response time when code cache was first generated
- ◆ Check times match to ensure the code cache data is for this response

```
void ResourceLoader::CodeCacheRequest::MaybeSendCachedCode(
    mojo_base::BigBuffer data,
    ResourceLoader* resource_loader) {
    // skip...
} else {
    if (cached_code_response_time_.is_null() ||
        resource_response_time_.is_null() ||
        resource_response_time_ != cached_code_response_time_) {
        ClearCachedCodeIfPresent();
        return;
    }
}

if (data.size() > 0) {
    resource_loader->SendCachedCodeToResource(std::move(data));
}
}
```

GOAL: Different Responses with the same response time

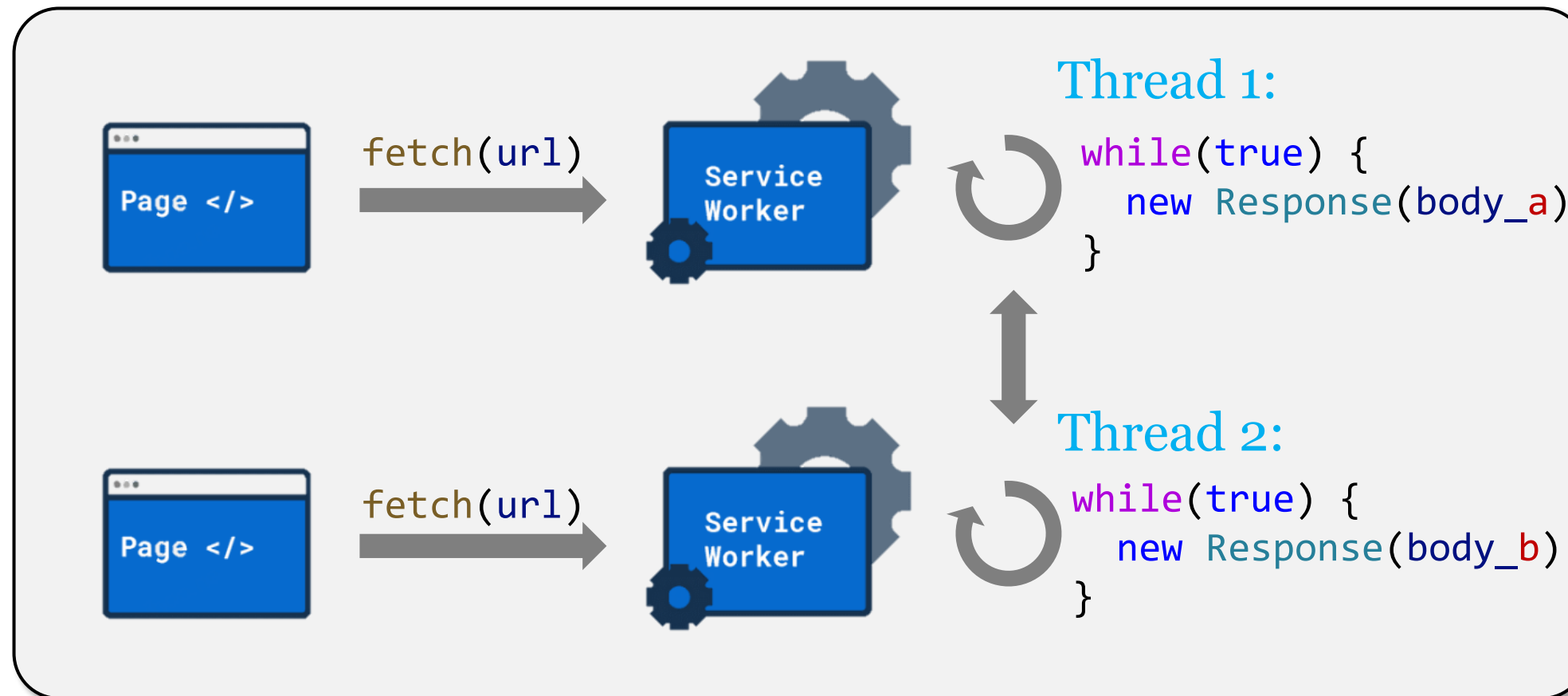
Response time (as a unique identifier)

- ◆ microseconds since the Windows epoch (January 1, 1601)
- ◆ initialized when a Response object is created
- ◆ we can create Response object using JS API

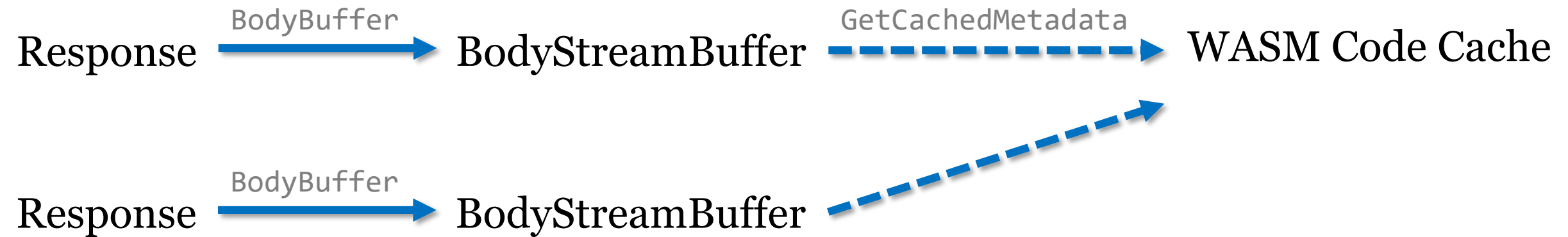
```
status_message_(status_message),  
header_list_(MakeGarbageCollected<FetchHeaderList>()),  
response_time_(base::Time::Now()),  
connection_info_(net::HttpResponseInfo::CONNECTION_INFO_UNKNOWN),
```

GOAL: Different Responses with the same response time

- ◆ two service workers continually produce Response with different data
- ◆ expect two response_time to be generated **within the same microsecond**



Code Cache Confusion again (CVE-2021-4056)



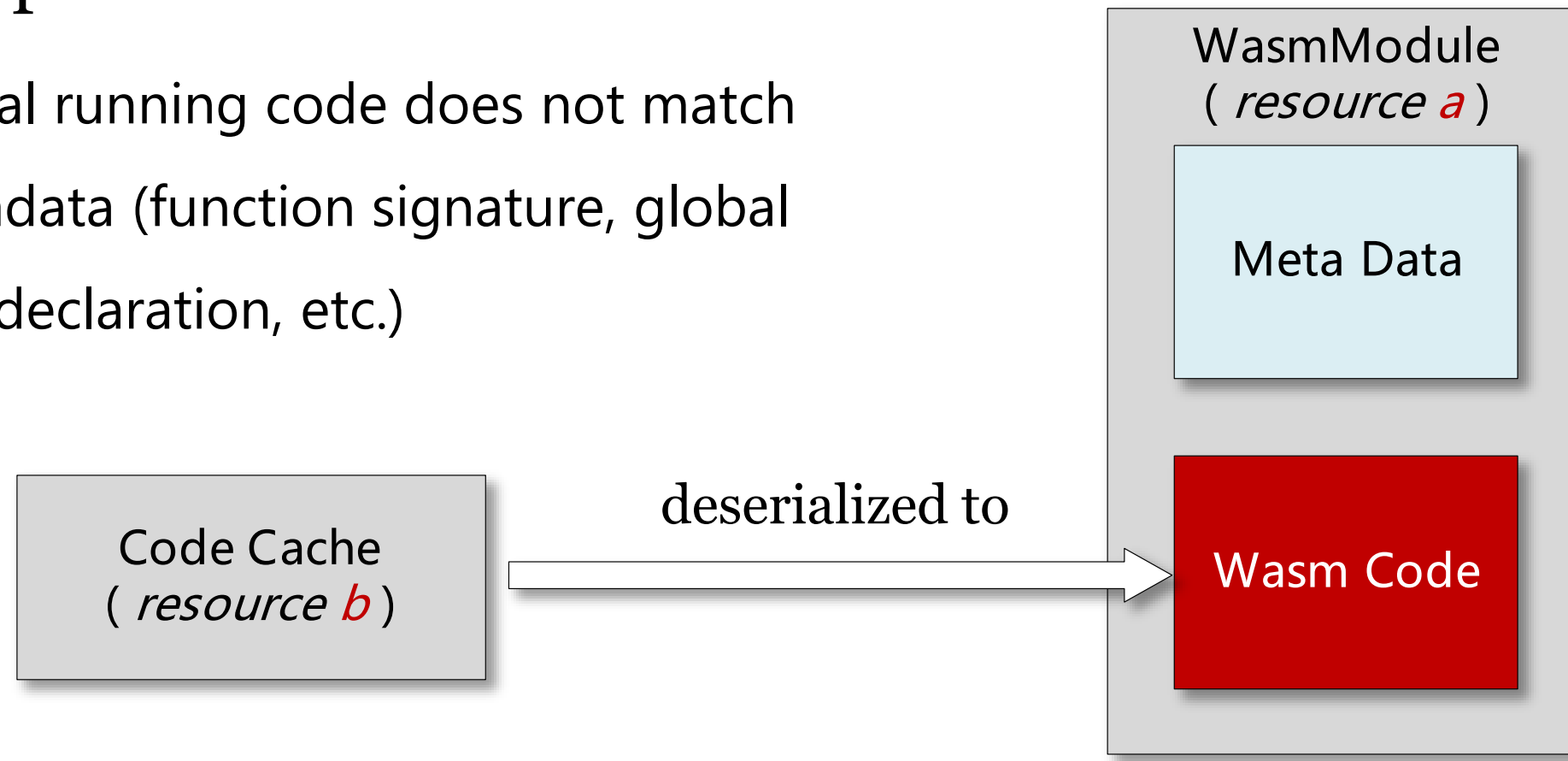
- ◆ Responses with the same response_time will have the same cache, even the response body is different
- ◆ Compile the Responses would cause cache confusion again

Fix (<https://crrev.com/c/3282643>)

Exploitation

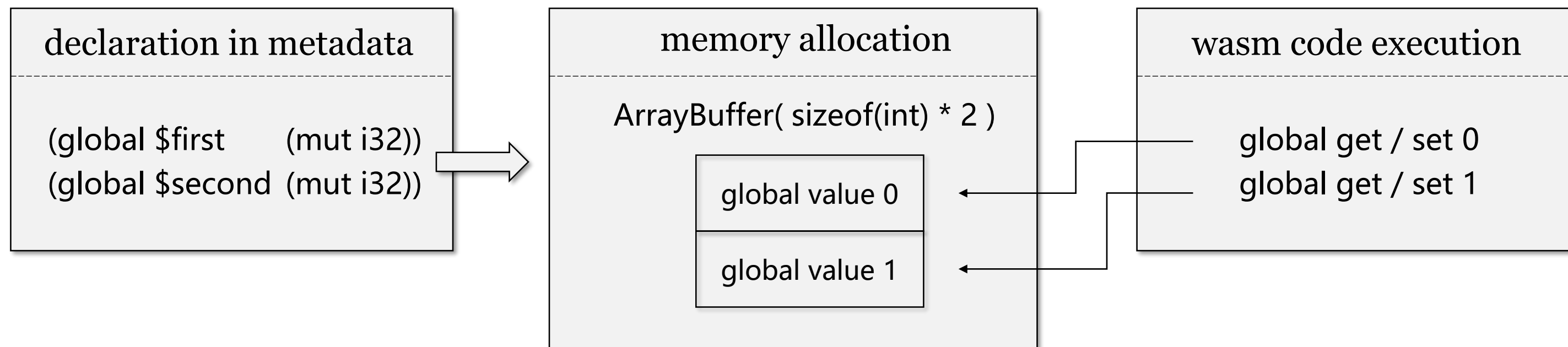
What happens next

- ◆ The actual running code does not match the metadata (function signature, global variable declaration, etc.)

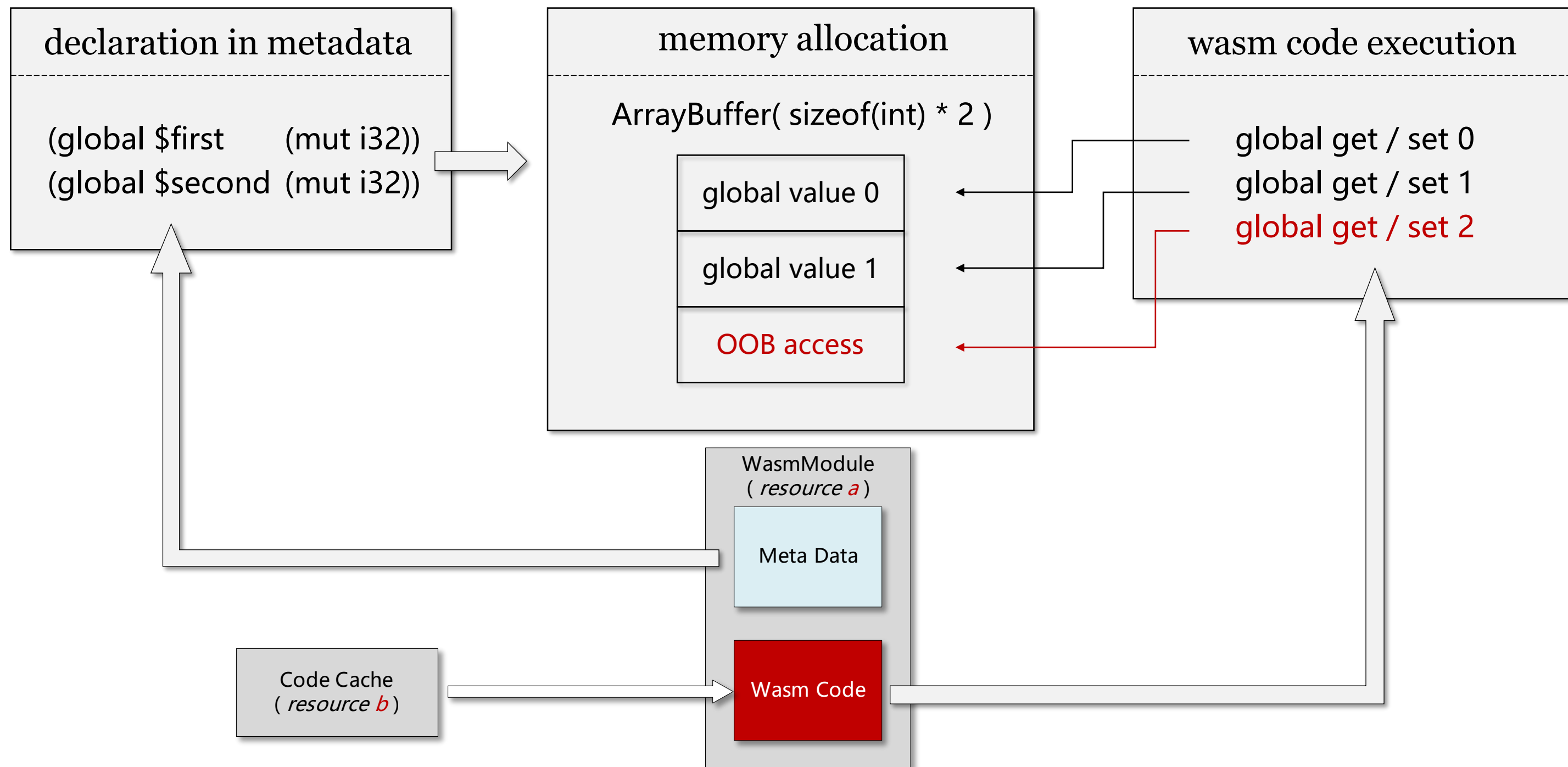


Exploitation

Global Variable in WASM



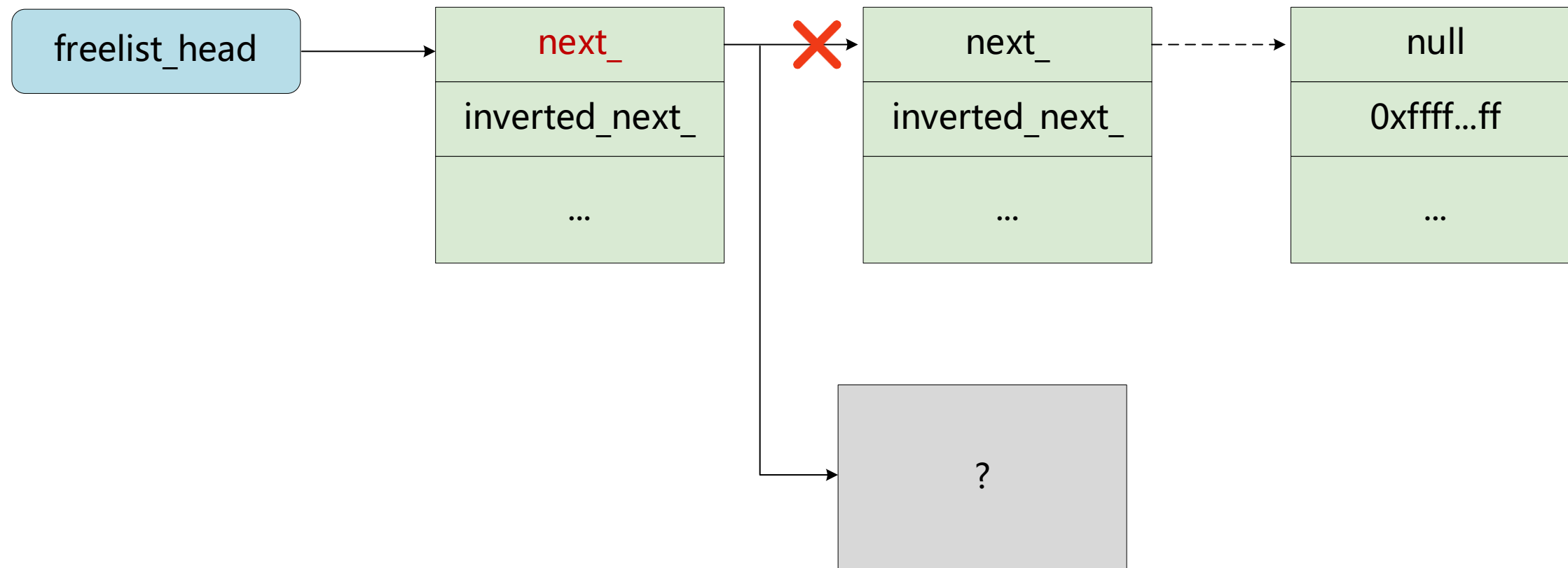
- ◆ Bounds check is performed at compile phase
- ◆ No bounds check at runtime



Classic free list corruption

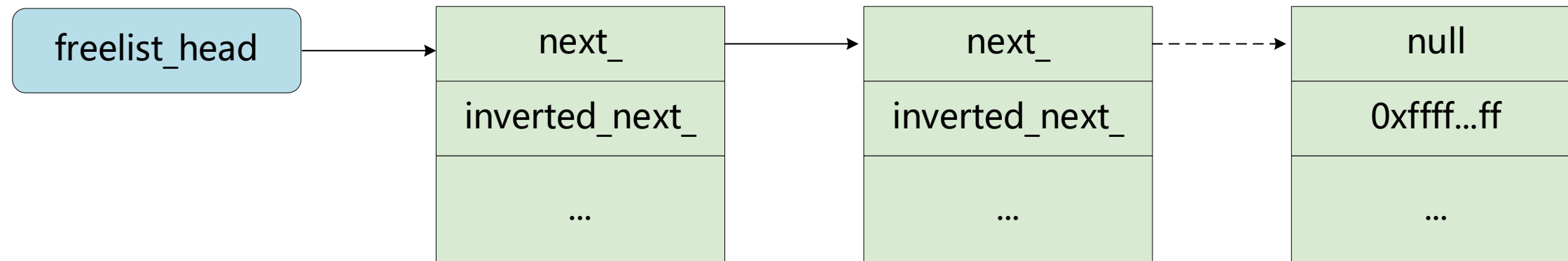
Released Arraybuffer's backing store in the free list (PartitionAlloc)

- ◆ overwrite next_ to do arbitrary allocation



Freelist corruption detection

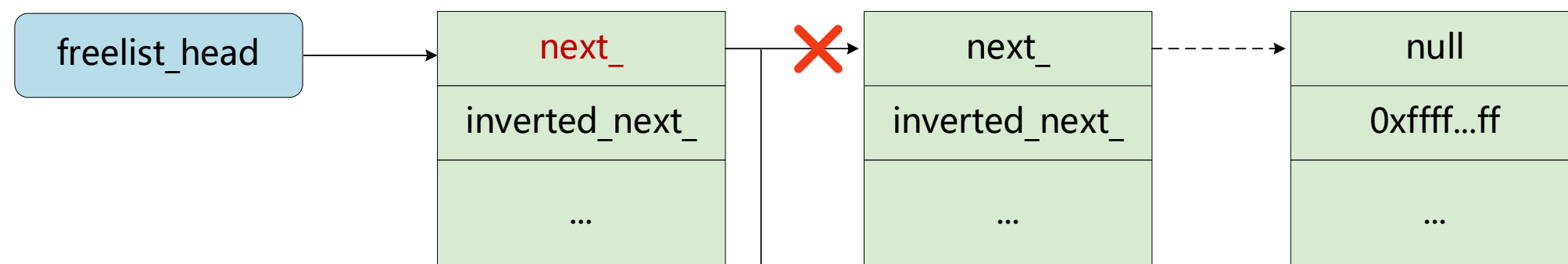
- ◆ if next_ is non-nullptr, check if $\sim \text{next_} == \text{inverted_next_}$
- ◆ if next_ is nullptr, move on



0x37c60060cfc0:	0xd0cf6000c6370000	0x2f309fff39c8ffff
0x37c60060cfd0:	0xe0cf6000c6370000	0x1f309fff39c8ffff
0x37c60060cfe0:	0xf0cf6000c6370000	0x0f309fff39c8ffff
0x37c60060cff0:	0x0000000000000000	0xffffffffffffffff

An arrow points from the 'next_' field of the first entry (0x37c60060cfc0) to the 'next_' field of the second entry (0x37c60060cfd0).

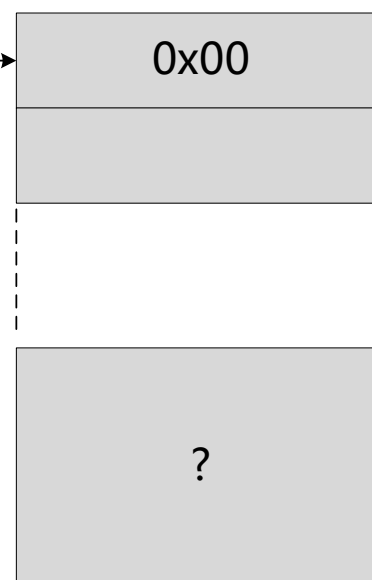
if next_ is nullptr, move on



2. find somewhere above with null data



1. memory we want to read / write



3. the actual allocated arraybuffer data
(for WASM global values)

4. do not forget we have OOB
access primitive



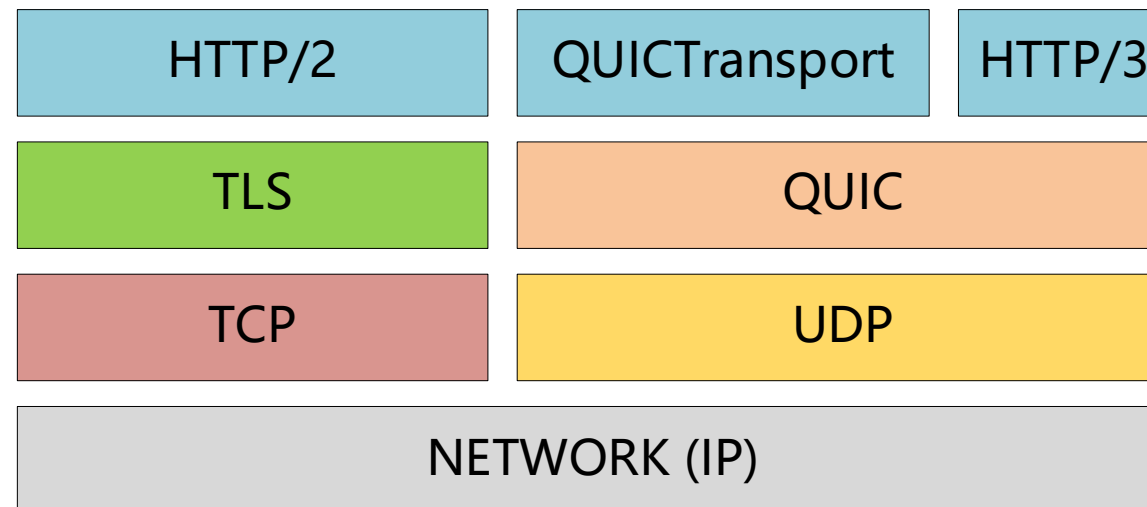
With OOB on heap and arbitrary allocation

- ◆ Leak useful data on the heap to defeat ASLR
- ◆ Enable MojoJS for further sandbox escape
- ◆ Run shellcode in render process, etc.

QUIC Transport

QUIC Protocol

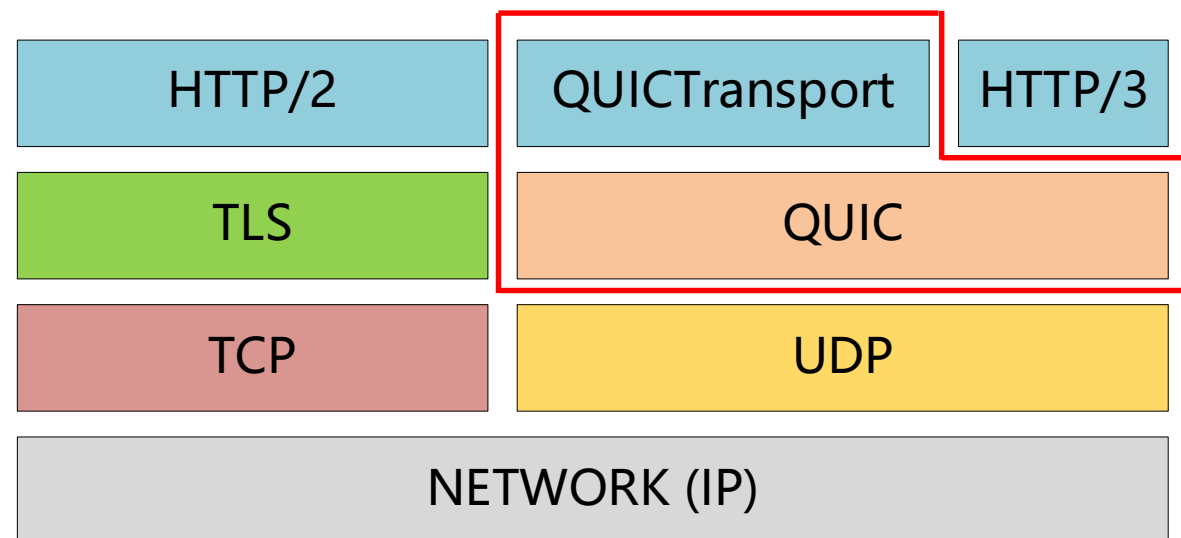
- ◆ Quick UDP Internet Connections (QUIC)
- ◆ base on top of UDP
- ◆ to improve transport performance for HTTPS traffic
- ◆ has been globally deployed at Google products



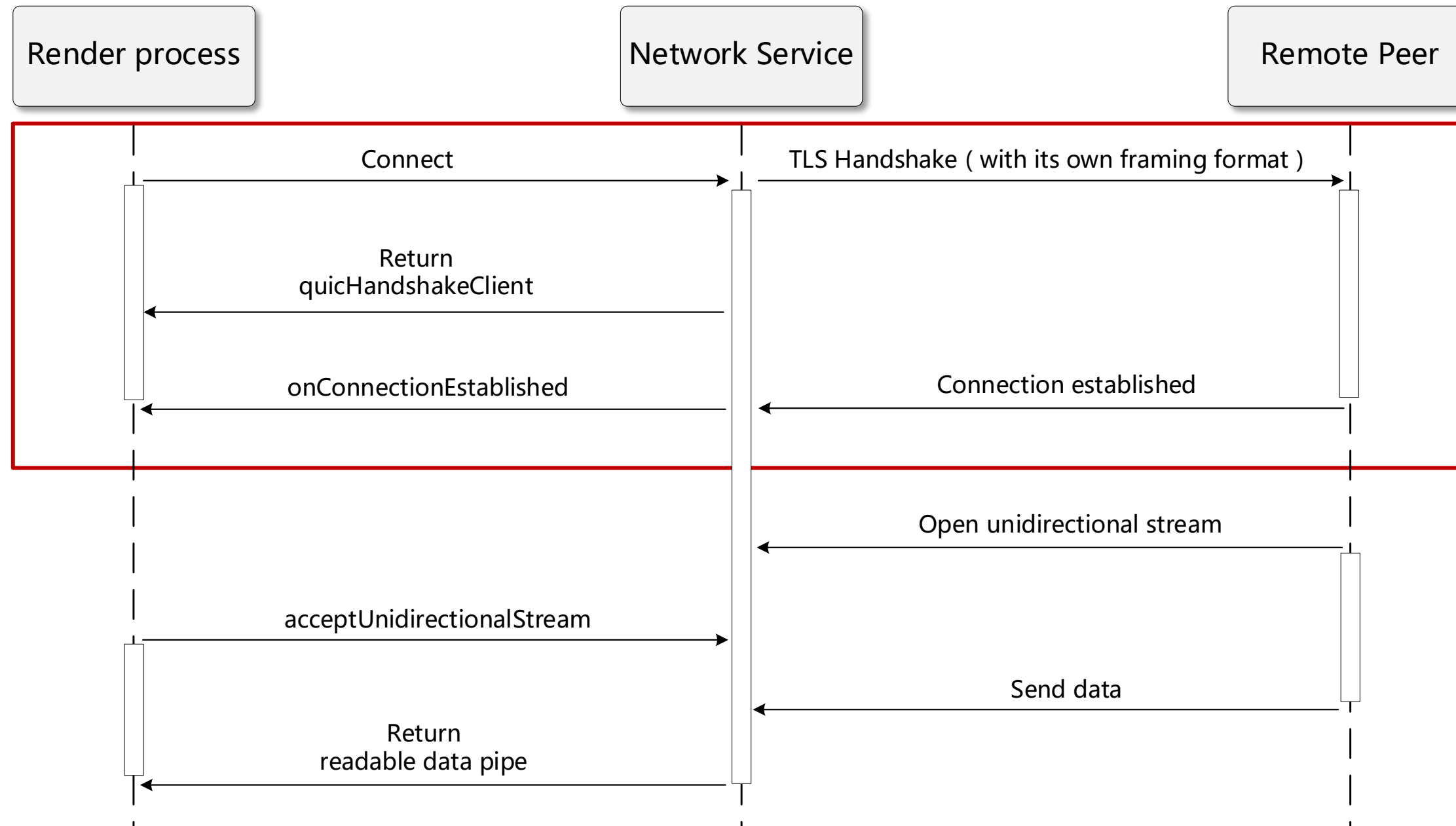
QUIC Transport

- ◆ web platform API (application level)
- ◆ allows exchanging data with remote peers using QUIC protocol

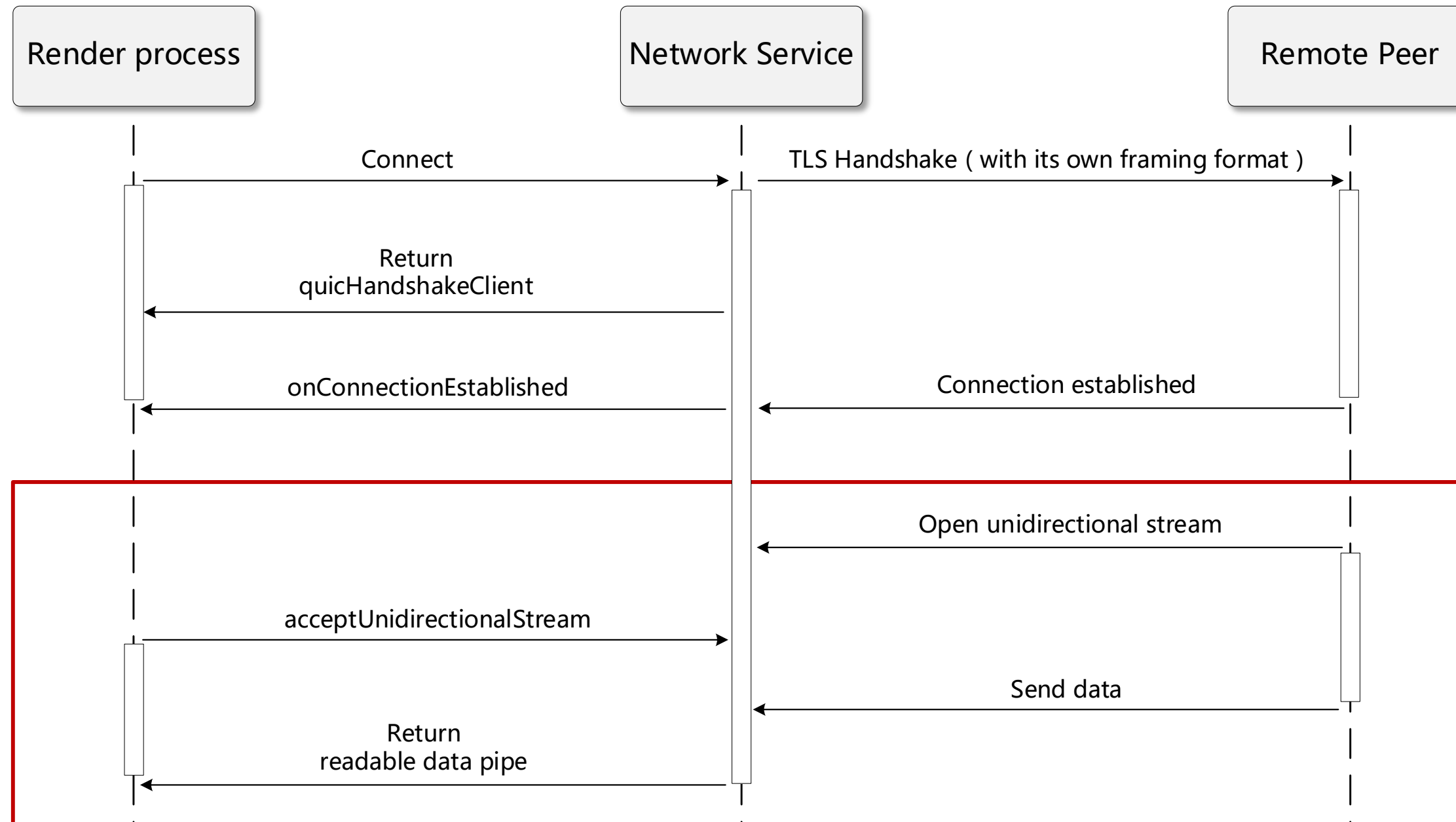
```
new WebTransport("quic-transport://example.com:4433")
```

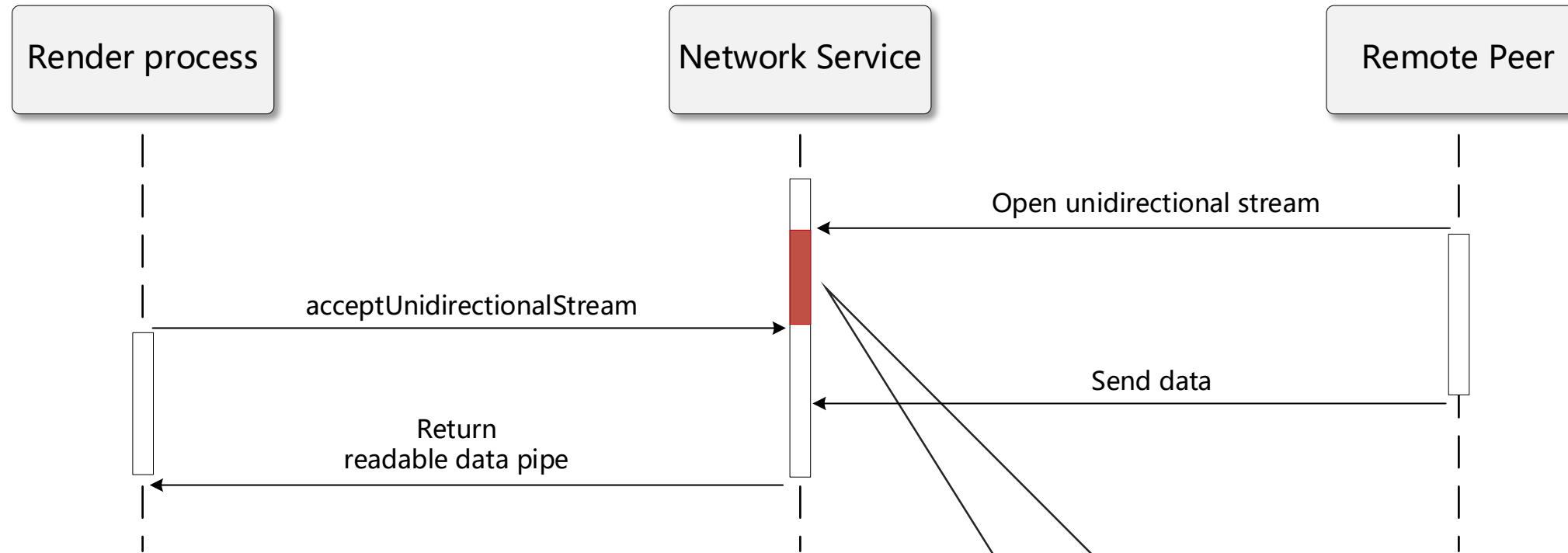


Establish the Connection



Open Stream on the Connection



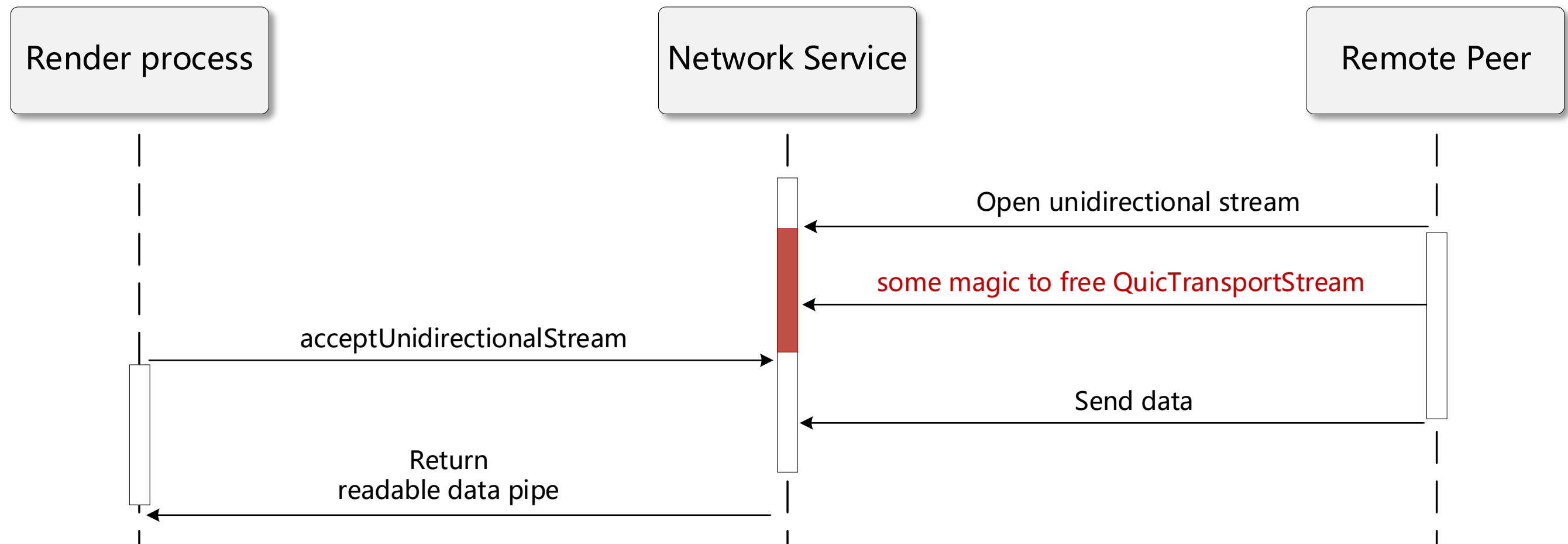


- ◆ contains streams that has been received by the session but have not been processed by the renderer
- ◆ raw pointers point to QuicTransportStream

```

// A client session for the QuicTransport protocol.
class QUIC_EXPORT_PRIVATE QuicTransportClientSession
{
    quiche::QuicheCircularDeque<QuicTransportStream*>
        incoming_unidirectional_streams_;
};
  
```

Can we free the QuicTransportStream before AcceptUnidirectionalStream ?



Free the QuicTransportStream from the server side

- ◆ send **LEFT_RST_STREAM** frame
- ◆ free the stream immediately

```
#0  quic::QuicSession::OnStreamClosed
#1  quic::QuicStream::CloseReadSide
#2  quic::QuicStream::OnStreamReset
#3  quic::QuicSession::OnRstStream
#4  quic::QuicConnection::OnRstStreamFrame
#5  quic::QuicFramer::ProcessIetfFrameData
#6  quic::QuicFramer::ProcessIetfDataPacket
#7  quic::QuicFramer::ProcessPacketInternal
#8  quic::QuicFramer::ProcessPacket
```

Use-After-Free in Network Service (CVE-2021-38002)

AcceptUnidirectionalStream >>> OnIncomingUnidirectionalStreamAvailable

```
void WebTransport::OnIncomingUnidirectionalStreamAvailable() {
    while (!unidirectional_stream_acceptances_.empty()) {
        quic::WebTransportStream* const stream =
            transport_->session()->AcceptIncomingUnidirectionalStream();
        // skip
        streams_.insert(
            std::make_pair(stream->GetStreamId(),
                           std::make_unique<Stream>(
                               this, stream, std::move(writable_for_incoming))));
        std::move(acceptance)
            .Run(stream->GetStreamId(), std::move(readable_for_incoming));
    }
}
```

← use-after-free

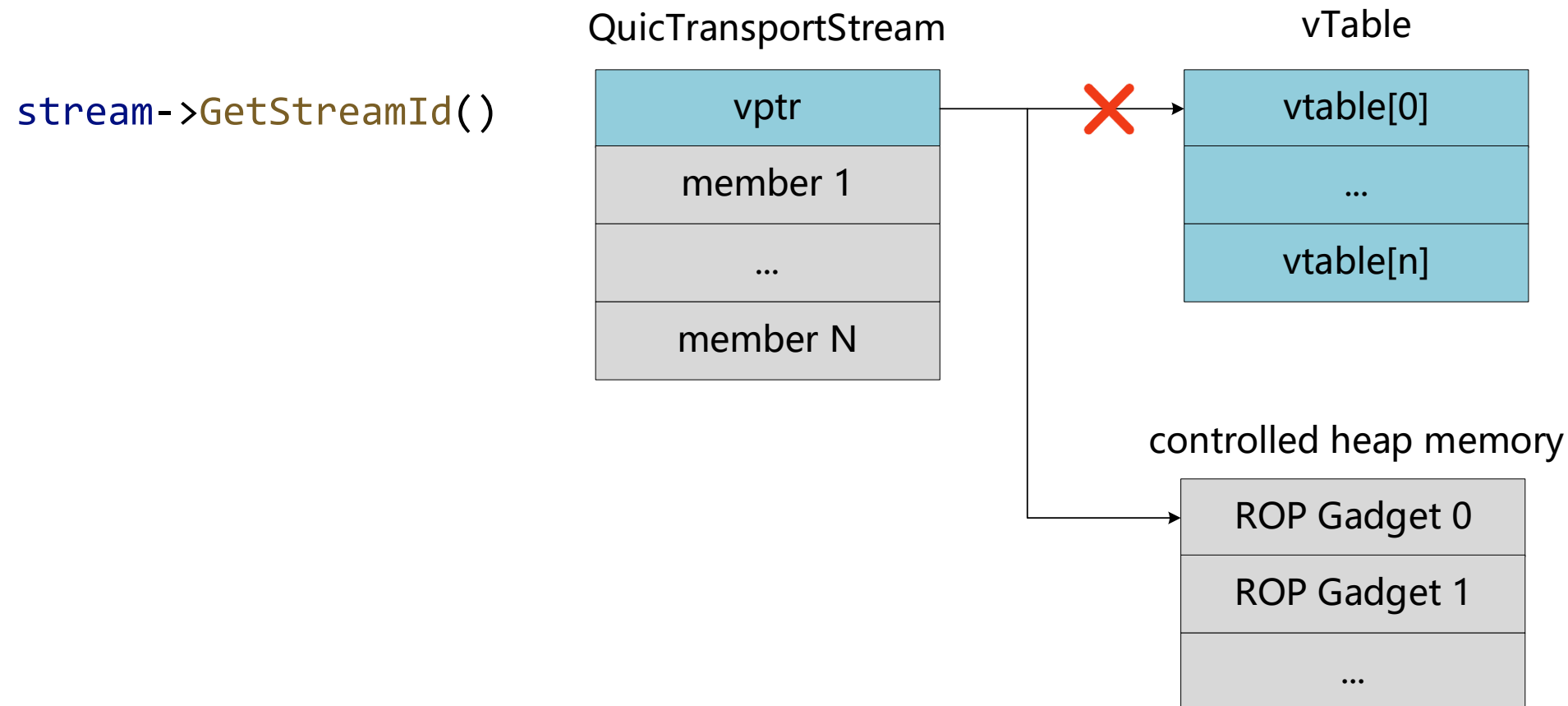
Exploitation

Environment: Chrome version 94.0.4606.81 on Windows 10

- ◆ Unlimited attempts at exploitation
 - Network Service would restart automatically after crash

Faking Virtual Table of the Freed Object

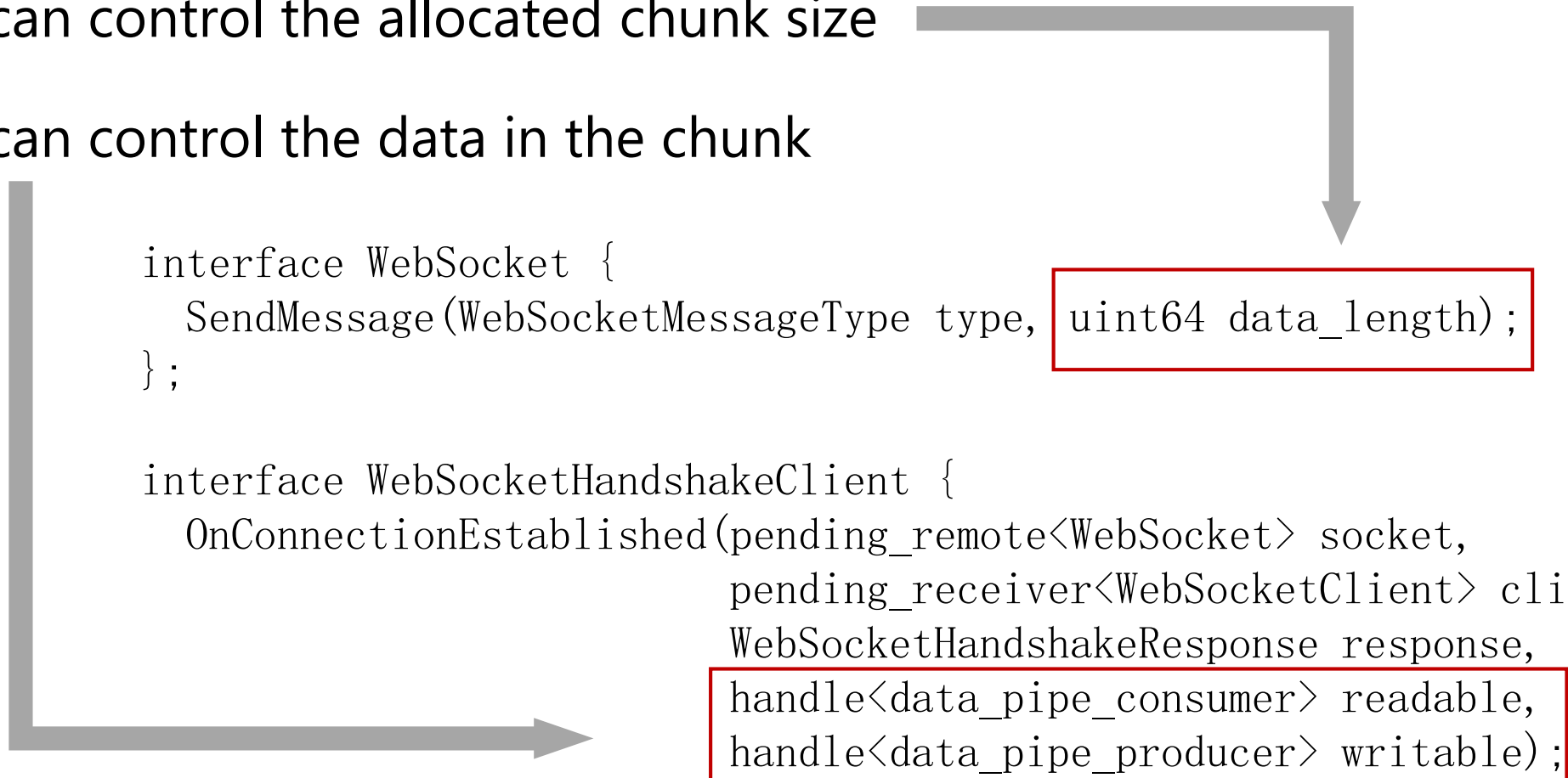
- ◆ Trigger UAF for leaking a heap address
- ◆ Build a ROP chain at a known heap address
- ◆ Trigger it again for executing the gadgets



Manipulate Heap in Network Service

WebSockets Interface

- ◆ can control the allocated chunk size
- ◆ can control the data in the chunk



```
interface WebSocket {  
    SendMessage(WebSocketMessageType type, uint64 data_length);  
};  
  
interface WebSocketHandshakeClient {  
    OnConnectionEstablished(pending_remote<WebSocket> socket,  
        pending_receiver<WebSocketClient> client_receiver,  
        WebSocketHandshakeResponse response,  
        handle<data_pipe_consumer> readable,  
        handle<data_pipe_producer> writable);  
};
```



Conclusions

- ◆ Modern browsers support many protocols and provide corresponding JavaScript APIs. One can control both ends of the protocol connection in some degree
- ◆ Network service that handles untrustworthy inputs at high privilege makes it a good target for security researchers to investigate
- ◆ Logic flaws in high-level network stack can cause memory safety bugs which are easier to be exploited

Thanks



<https://vul.360.net>