Securely-Initiated “Private Network Requests”

mkwst@, July 2019

We’d like to start imposing restrictions upon sites’ ability to poke at interesting bits of a private network that a given user has access to. The general plan sketched out in CORS-RFC1918 is to require a CORS-style preflight for any request that’s initiated from a context that wouldn’t typically have the ability to poke at the request’s target. We’ve consistently not made the time to implement this preflight, but it’s clear that this is in fact a real threat.

In order for the server’s preflight response to have any real meaning, servers need to be able to ensure that the `Origin` asserted in the request is authenticated, and isn’t under the control of a random middlebox between it and the user. A small first step, then, is to prevent non-securely delivered pages from causing requests to interesting bits of a user’s network. This is fairly tightly-scoped, meaning I might actually be able to get it done in my copious free time, but will require building out some foundational infrastructure that we can reuse for the larger project. With that scoping in mind, let’s dive into some detail.

Identifying “Private Network Requests”

The CORS-RFC1918 draft defines three address spaces: “public”, “private”, and “local” (in decreasing order of publicness). It further defines the notion of an “external request” (herein renamed “private network request” pending upstream renaming) as a request that crosses a network boundary from a more public address space into a less public address space. For example, a request initiated from a website delivered from a “public” address space would be considered a “private network request” if it targeted a server in a “private” or “local” address space.

To decide if a given request is a “private network request”, we need to know two things: the address space of the context that initiated the request, and the address space of the request’s target. In Chromium, we can calculate the former when constructing a Document or Worker by parsing `ResourceResponse::RemoteIPAddress()`, and storing the result on the resulting SecurityContext. Because of the joy that is DNS (and Happy Eyeballs, etc), we can only calculate the latter *after* a socket is connected (e.g. after HttpNetworkTransaction::DoInitStreamComplete).

It seems, then, that the following might be a reasonable approach:

1. Teach the network stack about the `IPAddressSpace` concept, and provide some mechanism for calculating it for a given IP address. It might make sense for this to live as a (static?) method on `net::IPAddress`.

2. Teach Blink to accept an `IPAddressSpace` when committing a navigation for a given response, and store this information for future use. The current `SecurityContext` implementation seems fine for this, but the calculation should move to (and be verified by) the browser process.

3. Tag outgoing requests with the initiating `IPAddressSpace` (similarly, conceptually, to what we do for the initiating origin in `network::ResourceRequest::request_initiator`).

While doing this, we should also remove the existing `is_external_request` member, as well as all the ductwork that gets it from Blink to the network stack (because the Blink-side calculation is woefully incomplete, and will be completely obviated by this new flag).

4. Add a new callback (or, possibly extend the BeforeHeadersSentCallback) to `HTTPTransaction` that will trigger after the socket connection has been established, and the IP address can be calculated. This callback should thread up through `URLRequestHttpJob` make the IP address available for comparison against the request's initiating `IPAddressSpace`. Boom, we have identiied a "private network request".

Acting on it

1. We'll want to implement the blocking check behind a new `kBlockPrivateNetworkRequestsFromNonSecureInitiators` flag, which can probably live in `network::features`.

2. In `URLRequestHttpJob`, we'll determine whether the request is a "private network request", as described above. If that's the case, and the initiating origin isn't "trustworthy", then the request should be cancelled. We could approach that by adding a return value to the callback which would signal to `HTTPTransaction` that it should error out the request.

   This will require the callback to be triggered in a place where cancelling the request is possible. It seems like `HttpNetworkTransaction::DoInitStreamComplete` might be one such place (though we don't have proxy information there, and I think we'll need it). If we reuse the BeforeHeadersSentCallback, perhaps we could alter `HttpNetworkTransaction::BuildRequestHeaders` to return something other than `OK` (perhaps a new error code, like `NON_SECURE_PRIVATE_NETWORK_REQUEST`, which we'd wire up in the usual way).

3. Blocked navigations would require an error page (and associated strings).

4. All blocked requests would require console warnings. It's not clear to me how to get these working. I thought we'd added bits to `NetworkServiceClient` for SameSite cookies, but those warnings have moved somewhere. Still, must be possible!