eroman@chromium.org
Last updated: May 24, 2016

Extract a NetLog viewer WebApp from chrome://net-internals/ (crbug.com/472699)

Proposal

Pull out all of the code for NetLog loading from Chromium's chrome://net-internals/, and move it to a separate repository on GitHub.

This has the immediate benefit of:
- Reducing Chrome's installer size (which will no longer have the HTML/CSS/Javascript code for these bits of chrome://net-internals)
- Reducing the attack surface posed by the WebUI chrome://net-internals code

In the long term, it may also offer these advantages:
- Can make the NetLog viewer great again by developing it outside of the Chrome repository [1]
- Can use the NetLog loading capabilities independently of Chrome (i.e. load up a web page on any supported browser and view a NetLog)
- Can define a general library of code for working with NetLog files that can be reused in other applications (i.e. log file analysis independent of the provided UI).

[1] Developing code as a WebUI as part of Chromium is not pleasant. It is a much more restricted development paradigm that limits what Javascript libraries, technologies, and subresource sizes can be included, and breaks the fast edit-refresh pattern of web development (i.e. all of the downsides of Javascript/HTML with none of the benefits!). Removing these limitations may lead to a renaissance of capabilities.

Background and Motivation

Chrome currently has a WebUI internal page, *chrome://net-internals*, which provides capabilities for:

1. Loading saved files (NetLog dumps) for post-mortem analysis
2. Exporting NetLog dumps (a.k.a. captures) to a file
3. Real-time observing of network events
4. Various one-off actions that relate to networking, such as configuring HSTS, clearing DNS caches, clearing socket pools.

This is a large chunk of javascript and HTML, that runs with elevated privileges, and ships with Chrome. The vast majority of the WebUI code is for loading and displaying NetLog dumps, (and doesn't need a privileged context).

The problems with the current design are:

- **Size bloat**. *chrome://net-internals* and all of its Javascript/HTML is shipped with Chromium adding to its installation size (even though very few users will use it; also it is not internationalized…)
- **Security risk**. chrome://net-internals (and other WebUI pages for that matter) expose additional attack surface. In fact the use of chrome://net-internals combined with some other bugs was used in the past for a p0wnium exploit.
- **Architectural inflexibility**. Because *chrome://net-internals* is part of Chromium, there are a number of artificial barriers in its implementation (like being unable to depend on external Javascript libraries or toolchains).
- **Lack of portability and extensibility.** The NetLog display code is tightly coupled to the Chrome binary. It cannot be used on other browsers, nor can its code be shared for non-browser applications. It also doesn't load well or at all on mobile.

Implementation details

Let's review again the different capabilities of today's *chrome://net-internals/*:

1. Loading saved files (NetLog dumps) for post-mortem analysis
2. Exporting NetLog dumps (a.k.a. captures) to a file
3. Real-time observing of network events
4. Various one-off actions that relate to networking, such as configuring HSTS, clearing DNS caches, clearing socket pools, export ChromeOS system logs

This proposal has the following effects:

- Capability (1) is torn out and moved to a separate WebApp
- Capability (2) is completely removed -- it is superseded by chrome://net-export/
- Capability (3) is abandoned -- there are ways to bring this back that will be discussed later, but as a first iteration this feature will simply be lost.
- Capability (4) is currently a grab-bag of mismatched features. It will need to be re-thought and fixed with time, but as part of this transition all of those capabilities will be left as-is.

Concretely, all of the following views will be deleted:

- chrome://net-internals/#capture
- chrome://net-internals/#export

- chrome://net-internals/#import
- chrome://net-internals/#events
- chrome://net-internals/#waterfall
- chrome://net-internals/#timeline

And at least in the short term, these will be left behind:

- chrome://net-internals/#hsts
- chrome://net-internals/#httpCache
- chrome://net-internals/#alt-svc
- chrome://net-internals/#http2
- chrome://net-internals/#quic
- chrome://net-internals/#sdch
- chrome://net-internals/#modules
- chrome://net-internals/#bandwidth
- chrome://net-internals/#dns
- chrome://net-internals/#sockets
- chrome://net-internals/#alt-svc
- chrome://net-internals/#prerender

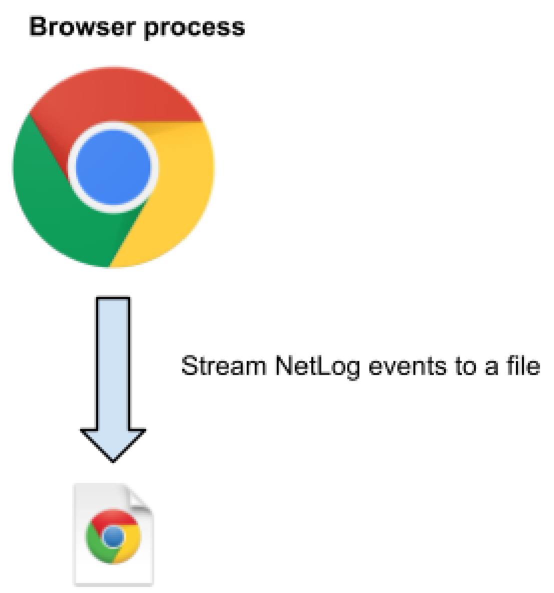
It is beyond the scope of this design document, however the views highlighted in orange are also good candidates to move out of *chrome://net-internals* in the future and can comfortably live in the NetLog viewer application. However more work will be needed to transition these. Notably (a) enable real time (or close to realtime) viewing of NetLog stream from the WebApp version (b) expose these panel's data as regular events in the NetLog stream (whereas today they work by polling the browser's current state).

#hsts offers capabilities that go beyond viewing state (i.e. can modify HSTS), and will not be extractable. Similarly #httpCache offers a browser for the current cached files which will not be directly transferrable to an external WebApp (however may be worth just killing).

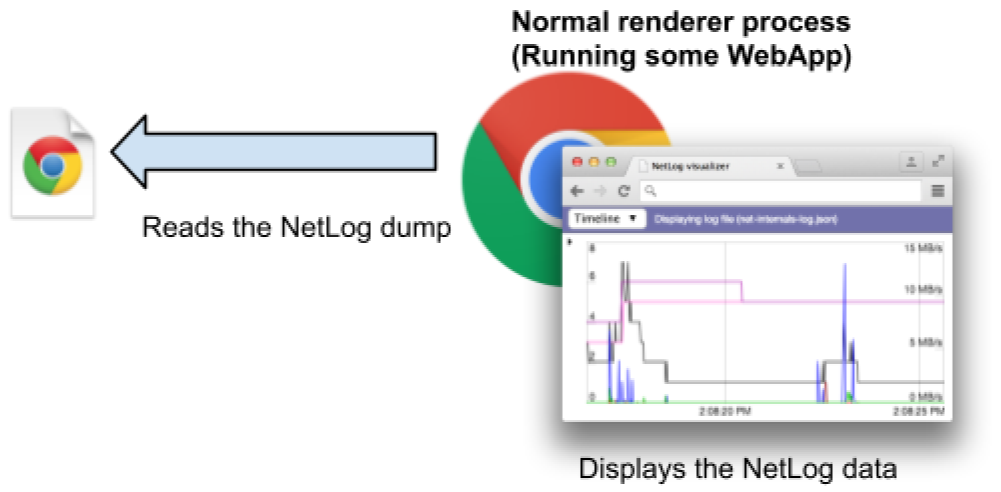
High level architecture *after* the refactor

The workflow now resembles this….

When writing NetLog dumps, the browser writes the event stream directly to a file:



Once the capture has completed (or potentially while it is still in progress), the data can be loaded into the standalone visualizer application.



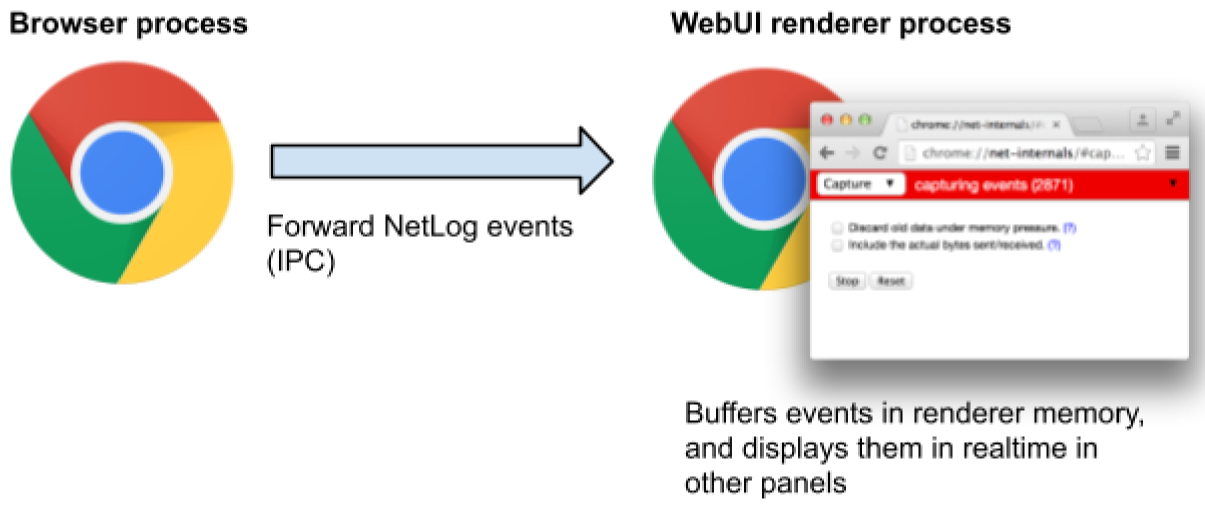
The difference is that now the visualizer application is no longer running as part of *chrome://net-internals*, but as a separate webapp (or native app, or whatever else you wanted).

The WebApp can use the FileAPI to read the NetLog dump's contents from disk.

High level architecture *before* the refactor

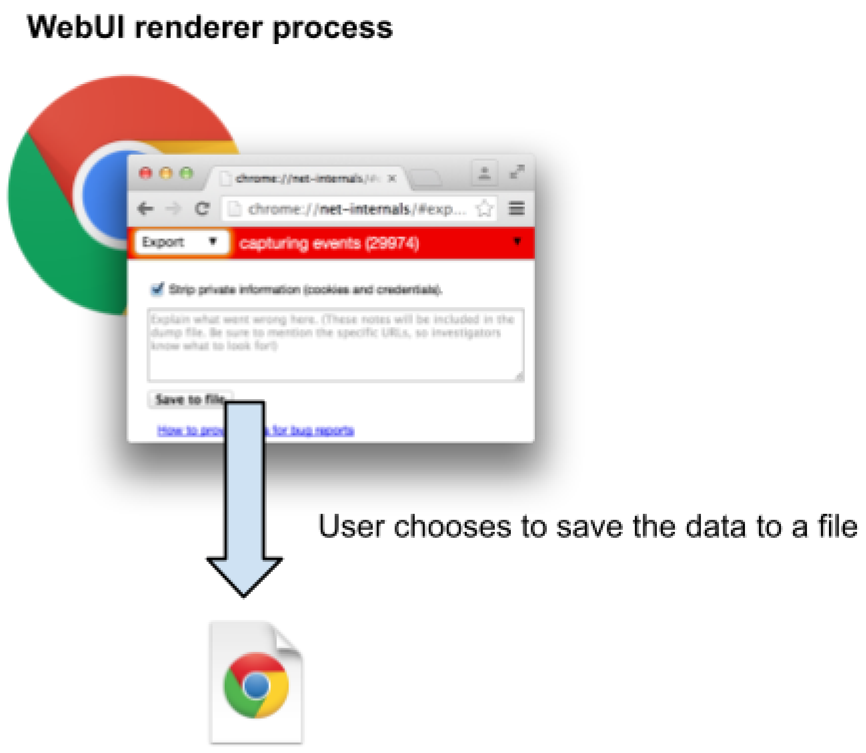
Here is how things work today (before the refactor).

The browser process runs the network stack, and generates a stream of events from C++. These events are serialized to JSON and *forwarded* to a WebUI renderer process running chrome://net-internals/ Javascript code:


Browser process
WebUI renderer process
Forward NetLog events (IPC)
chrome://net-internals/#cap...
Capture
capturing events (2871)
Discard old data under memory pressure. (?)
Include the actual bytes sent/received. (?)
Stop
Reset
Buffers events in renderer memory, and displays them in realtime in other panels

This renderer process receives the data and buffers it in memory (capture). It simultaneously updates views of the data (real-time visualization). This consumes a surprising amount of memory and CPU for all the Javascript and DOM objects. It is common for long running *chrome://net-internals* captures to crash thanks to running out of memory (and why there is a check box on #capture to limit the data kept)

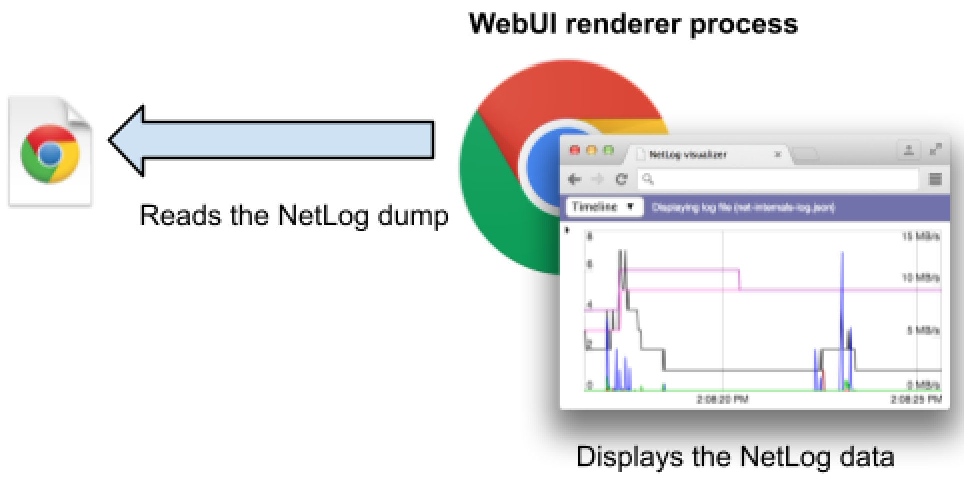
While this capture is ongoing, the user can choose to save the data to a file:



Strip private information (cookies and credentials).

Explain what went wrong here. (These notes will be included in the dump file. Be sure to mention the specific URLs, so investigators know what to look for!)

Now that a file has been saved, it can be exchanged with other developers who can subsequently load it into their chrome://net-internals/ to explore the captured data:



Where will the new code live?

As a separate project hosted on GitHub (under the GoogleChrome umbrella).

What about real-time inspection of network activity?

Proper real-time support will be no more with this model.

At least after the initial transition.

However near real-time can be approximated grafted on. It will be a separate project to figure out how best to do that. Some brainstorming to demonstrate that it is *possible* to accomplish this in the new architecture (even if not elegant):

- Hack #1: User can keep re-loading the file (which may be changing as more data is being written to it). Dumb, but requires no extra work.
- Hack #2: Can automate the above so it is transparent to the user, but achieves some granularity of real-time by polling. This assumes the File API treats file handles as a persistent entity rather than snapshots into a view of the file at a particular time (wasn't clear from spec, need to experiment)
- Hack #3: If not possible through the File API, could create a simple python server that you run locally and exposes the log file changes through an http://localhost interface. Not clean, but does show that at the extreme it is *possible*.

Why not use an extension API to preserve the real-time capabilities?

An alternate design is to make the NetLog event stream something directly observable by Chrome extensions. This means that the real-time viewing capabilities can be preserved unchanged.

I don't think this is the right approach for several reasons:

- Yet another extension API (with very narrow appeal)
- We want to move towards writing NetLog files directly to disk and have that be the canonical backing store rather than letting in-application consumers observe it directly (as that is the common interface used between desktop and mobile).
- Would be a Chrome-only solution

- Writing as an extension is less hackable (personal anecdote)