

Integrity for the Web

Web v/s Native Apps

Native Apps

A single file that contains all of the application code



- Code signatures to ensure that you got the exact bundle that was published
- Integrity hashes to ensure that you got the same binary as someone else

Web Apps

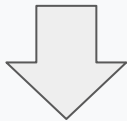
HTML referencing other JS and HTML loaded at runtime



- No ability to check integrity or authentication on the entire application
- No ability to check if you are receiving the same application code as someone else

Because of this gap...

- **Applications like Signal don't exist on the web**
 - Because there is no way to ship a web application with a threat model other than "trust that we're serving you non-malicious JS"
- **Companies like Meta have had to implement their own polyfills of this behavior**
 - E.g. CodeVerify, which allows checking an entire application bundle via a Chrome extension. But these have a weaker security model than web-platform native solutions.
- **Even many types of sub-resources can't implement integrity checking because of limitations with SRI**
 - Example: Google Analytics: A dynamic script so hash-based SRI doesn't work
 - Example: ES Modules and Service Workers: Not all APIs support SRI



Increased supply-chain security risks for the web

What if the web attempted to solve this?


- **Progressively, so that:**
 - Large applications can deploy extremely strong integrity and transparency goals
 - I.e. “Make it possible for Signal to ship a web app”
 - Small applications can deploy integrity checking for certain parts and still achieve security gains
 - I.e. “Make it so that more subresources use SRI-like mechanisms”
- **A cross-company working group has been discussing this problem**
 - We’ve been exploring possible designs privately and will soon move the discussion to W3C
 - We’re looking for initial feedback and thoughts on the importance of these problems

Approach

Full integrity and transparency solutions are possible

- **Aim for solutions that can offer strong integrity guarantees**
 - Partial mechanisms might make a comprehensive technology more difficult to deploy when it is ready
 - We don't want people to believe they have strong integrity guarantees if they're only mitigating some of the risks
 - We want to aim for comprehensive integrity properties with consistency and transparency as a goal from the start
 - While still ensuring a good separation of integrity and transparency mechanisms
- **Preliminary technical designs make us believe this is solvable**
 - Solvable for both large and small sites

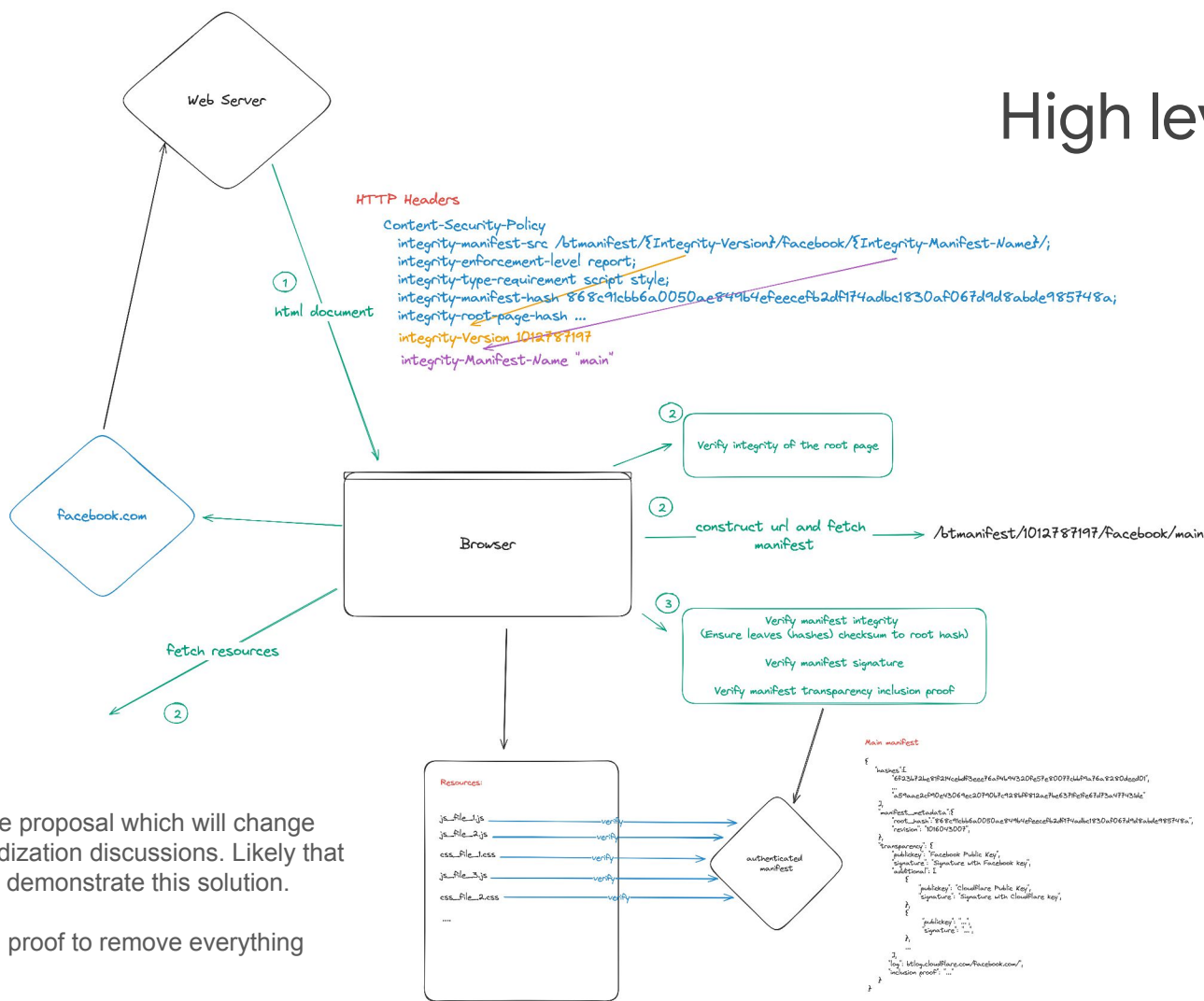
Possible intermediate integrity milestones

- 
- **SRI with signatures:** Make it possible to use SRI for dynamic resources
 - **SRI for more resource types:** Make it possible to use SRI for other types of subresources (e.g. images)
 - **document-level require-sri-for:** Make it easier to ensure SRI is enabled for all subresources (of a given type?) on a page (maybe with manifests?). Also (maybe) supporting report-only mode.
 - **origin-level require-sri-for:** Make it easier to ensure that SRI is enabled across an entire origin (but with the ability to turn this off at will)
 - ***Committed* origin-level require-sri-for:** Make it possible to commit to using SRI across an entire origin (similar to HSTS)

What if we had something like Certificate Transparency, for application code?

- Idea: Provide transparency to ensure that all users are served the same bundle of code
 - Prevents targeted attacks and ensures auditability of code
 - Important for end-to-end encrypted use cases
- This could be a stronger security model than native apps offers!
 - Mobile apps arguably offer something similar, but weaker, by relying on app stores to serve the same app to all users

High level idea



See the green arrows

This is - one - possible proposal which will change based on the standardization discussions. Likely that **Meta** and **Mozilla** will demonstrate this solution.

Remove the inclusion proof to remove everything related transparency

SRI

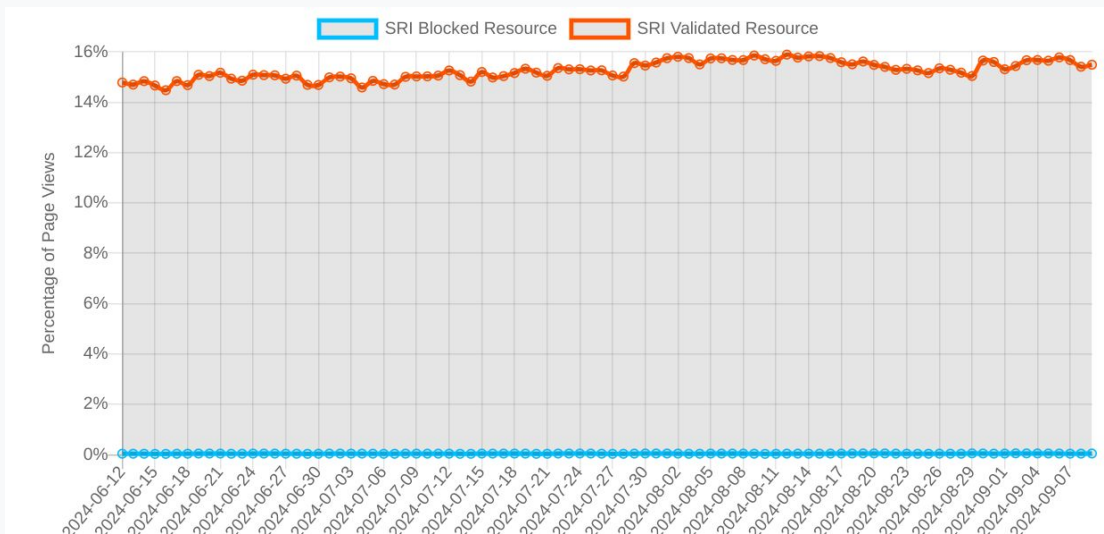
Subresource Integrity (SRI)

SRI allows verifying that a subresource matches a specified hash:

HTML

```
<script  
  src="https://example.com/example-framework.js"  
  integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9r7x7HNQlGyl1kPzQho1wx4JwY8wC"  
  crossorigin="anonymous"></script>
```

Hash-based SRI makes it possible to include a resource hosted by a third-party, while ensuring that the third-party isn't maliciously modifying it



Subresource Integrity (SRI)

Hash-based SRI doesn't work well for

- Unversioned libraries that need to be continually updated
 - Examples: Google Analytics
- Libraries that are dynamically generated based on the request
 - Examples: Polyfills or personalized scripts
- Subresources that are included without knowing the hash ahead of time
 - Examples: When applications and JS are deployed separately

Historical Proposal: Signature-based SRI




<https://github.com/mikewest/signature-based-sri>

Extend SRI to allow specifying a public key, and validating signatures

HTML

```
<script  
  src="https://example.com/example-framework.js"  
  integrity="ed25519-[base64-encoded public key]"  
  crossorigin="anonymous"></script>
```

This “solves” all three of the problems with hash-based SRI:

-  Unversioned libraries that need to be continually updated
-  Libraries that are dynamically generated based on the request
-  Subresources that are included without knowing the hash ahead of time

Maybe this is a good stepping stone?

Historical Proposal: Signature-based SRI

<https://github.com/mikewest/signature-based-sri>

But, it still leaves a number of key security questions unanswered:

- **Resource Binding:** If a publisher has one public key for all their scripts, how to ensure that the ``some-vulnerable-library.js`` isn't swapped in for ``safe-library.js``
- **Rollback Prevention:** If a vulnerability is found in a library, how to ensure that ``library-v1.js`` isn't served in response to a request for ``library-latest.js``

My personal opinion:

- **Resource Binding:** Not necessary since publishers can just have separate public keys for each resource.
- **Rollback Prevention:** Maybe necessary, but...
 - Could be an optional feature
 - This is a two-way door, we could ship signature-based SRI without rollback protections and add it in later without breaking backwards compatibility