Web2Native Bridge

Delegating Hardware and Security Related Operations to the Native[1] Level

Since there are no “perfect” solutions, this proposal aims *combining* the best of the two worlds (*web* and *native*), as an *alternative* to at any cost and time *duplicating* the functionality of the native level in the Web.

A core feature of the scheme is that it enables developments by third-parties. *Currently browsers are effectively blockers for innovation.*

A *deliberate* “side-effect” of this proposal is that it makes it possible adapting the security and privacy model to the actual application[2].

Although the presentation has a certain bias toward payments, the design should be usable for many other applications as well[3].

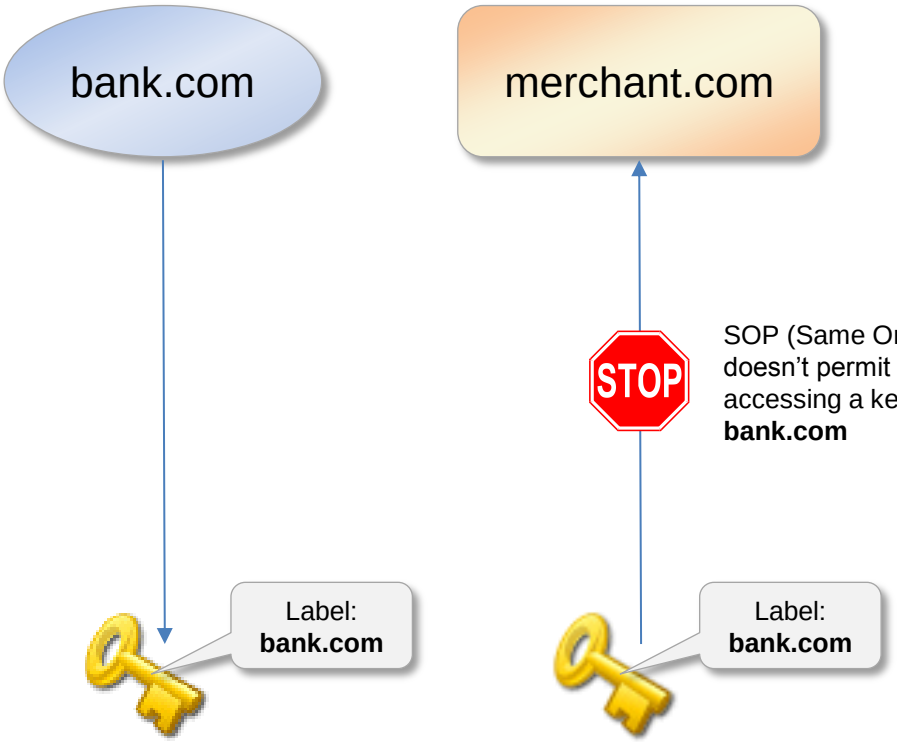
1. Native in this context means platform-local including installable HTML5/JS applications.
2. Payment- and authentication-applications typically have quite different issues and requirements.
3. Including certificate enrollment, hardware token management, federated authentication solutions and on-line signature systems. In fact, even massively popular music streaming services, cloud storage systems, on-line gaming sites and open source collaboration networks *currently rely on a hodge-podge of non-standard methods for interacting with client platforms from the browser*.

Using Web-level APIs like WebCrypto for Payments

Problem

1. Client receives a payment key through WebCrypto

2. Client wants to utilize the payment key through WebCrypto



The figure above shows the implications of keys bound to a specific origin. It is important understanding that *it would be dangerous allowing untrusted merchant code directly accessing a bank-issued key*.

"Workaround"

There are multiple non-standard and awkward workarounds which renders WebCrypto less suitable for a wide range of applications *which do not have a specific origin as a natural boundary*.

Additional issues…

- There are numerous security HW standards. That *SIM, U2F, TPM, TEE, etc. operate at entirely different levels of abstraction* also imply specific management solutions.
- There is no concept of "Trusted Code" in the open Web.

Exposing System APIs in the Open Web – “Permission Hell”

APDU Level Access

A web-application wants to connect to a secure element. Since a browser does not “understand” APDU it can only offer a primitive security prompt.

PKCS #11 Level Access

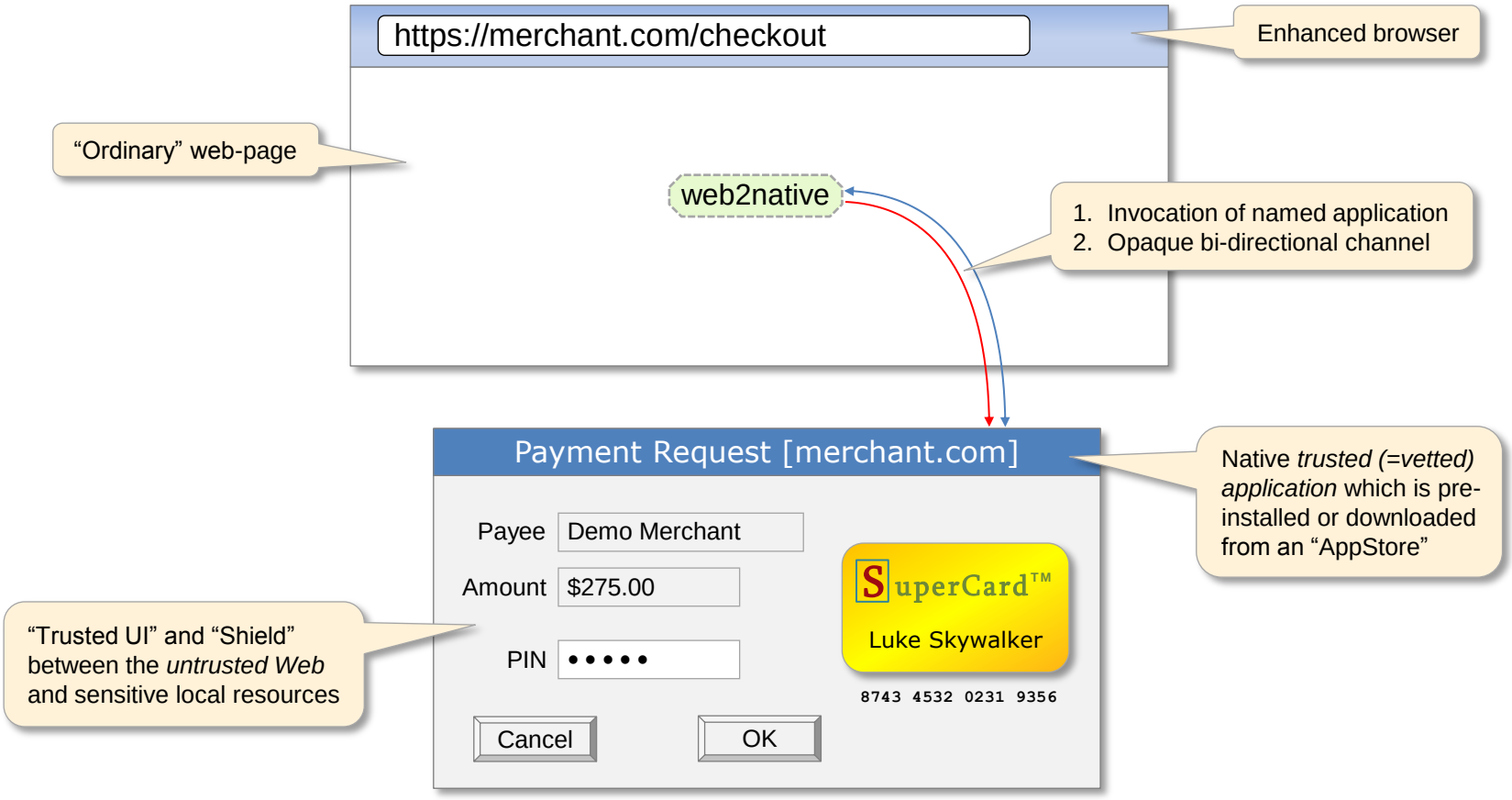
Since the browser cannot know what the application is about to do with signed data, it can only offer a primitive security prompt.

Conclusion

Permissions are fine for things that ordinary users can understand like “Your Location” but quite unsuitable for a large class of sensitive system APIs and associated user data which in the native world always are dealt with as a part of a packaged application. The next page outlines a possible way to “emulate” this functionality in the Open Web. The primary advantages would be:

- Limit direct access to sensitive APIs *by untrusted web-code*
- High-level *service-oriented* schemes make applications less dependent on variations in platform APIs and architectures
- Provide *meaningful* information to users

Suggested Deliverable – Web2Native Bridge



The Web2Native Bridge is essentially only an *application-neutral and domain-independent* invocation and message-passing mechanism. After invocation a *private*, *asynchronous* and *opaque* communication channel is created between the native application and the invoking web-page.

Web2Native Bridge applications typically offer a standardized interface towards the Web (=maintaining the web-paradigm).

Dedicated high-level service-oriented interfaces like shown above minimizes the need for annoying users with difficult security prompts.

A local wallet application could thus have identical characteristics when used in a brick-and-mortar shop as when invoked over the Web.

Web2Native Bridge – What Would be "Standard"?

Web (Browser) Standard - The Actual Work-item

- Application Invocation and Discovery[1] API
- Channel API
- Origin of caller including HTTPS info for accommodating a variety of security models *including SOP*

Application-specific Standard

- Registered name of the target application (presumably identical for all platforms)
- Message data (typically serialized JSON creating a "virtual" API)
- Privacy-preserving features including user UI alerts (a properly designed payment system does not expose identity information to the relying party in contrast to a digital signature application)

Platform-specific (Proprietary)

- Channel technology like UNIX sockets, stdin/stdout redirect, etc.
- "AppStore" and vetting processes
- Native applications including platform-specific UI
- Window handle to the invoking page enabling applications to "float" on top
- Authentication scheme between the browser and native applications
- Registry holding "granted" native applications
- Security hardware interface (TPM, TEE, SIM etc)

The Web2Native Bridge enables platform-independent interfaces to the Web while the interfaces to the platform may be entirely proprietary

1) TBD, may not be required

Web2Native Bridge – Security Considerations

Note: This section does not deal with robustness of implementations or how the system operates if the platform is compromised.

Browser Security

The Web2Native Bridge introduces a mechanism which enables standard web-applications invoking *external* (local) applications through a new interface (TBD). This does not in itself present a risk to the *browser environment*.

After successful invocation the Web2Native Bridge creates a *bi-directional trusted message channel* to the invoked application which from the browser's side has similar properties to the already established **`postMessage()`** and **`addEventListener()`** functions.

That is, the Web2Native Bridge does not rely on installing custom code directly in the browser like the deprecated NPAPI did.

Platform Security

An external application of the type used by the Web2Native Bridge would most likely have the same possibilities as any other local application running in the user's context.

In contrast to traditional local applications, *Web2Native Bridge applications can typically be invoked by any web-site*. For large-scale usage, such applications MUST therefore be vetted in a specific way to avoid potential security or privacy violations. That is, it MUST NOT be possible invoking trust-wise unknown Web2Native Bridge applications except for development purposes. Also see HTTPS CCA.

Web2Native Bridge invocation requests MUST be derivable to secure origins (authenticated by HTTPS).

Web2Native Bridge applications MUST in a clear way inform users what is requested as well as including the ability to cancel the request and possibly also offering an option to block.

Each Web2Native Bridge application exposes a specific interface based on messages passed through the Web2Native Bridge channel. Web2Native Bridge applications MUST verify the correctness of inbound messages and immediately abort execution if there is a mismatch.

A Web2Native Bridge application MAY restrict access by requiring callers performing something to prove their "membership" or similar.

Privacy Issues

There could be minor privacy-impediments since the invocation mechanism can enable additional finger-printing of the client (=finding out that a certain Web2Native Bridge application is available). However, silent enumeration of supported applications MUST NOT be permitted.

If the user accidently interacts with another web-site than he/she intended, the user could be tricked providing information which usually isn't intended for arbitrary consumption like an eID certificate containing a citizen ID. An identity-related Web2Native Bridge application SHOULD therefore inform users about previously not encountered sites.

Application Vetting

In addition to the intrinsic security features, a vendor performing vetting may further restrict usage of certain applications or impose special requirements on developers.

Web2Native Bridge – Related Information

Origin and “Inspiration” – Google Chrome Extensions:
http://www.cnet.com/news/google-paves-over-hole-left-by-chrome-plug-in-ban/
http://blog.chromium.org/2013/10/connecting-chrome-apps-and-extensions.html

Note: The Web2Native Bridge does not utilize browser extensions, it is a pure API.

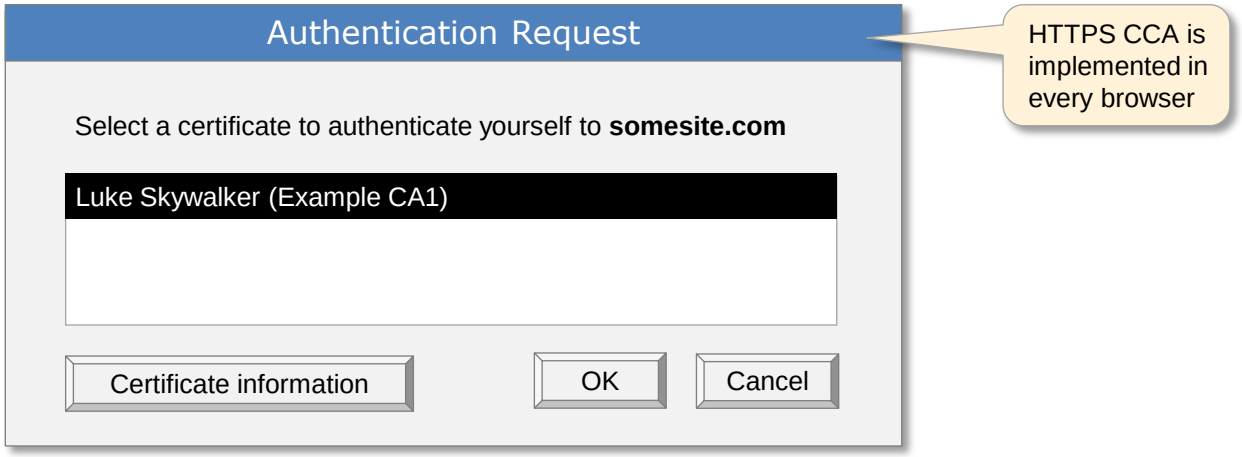
Web Intents (Shelved):
http://www.w3.org/TR/web-intents

Recognized “Pain Point”:
https://code.google.com/p/chromium/issues/detail?id=378566

HTTPS CCA Security and Privacy Model:
HTTPS CCA

When Security, Web, and PPTs Get Too Boring – Real Stuff!
https://www.youtube.com/watch?v=0O1v_7T6p8U

Reference: HTTPS CCA (Client Certificate Authentication)



HTTPS CCA enables you to handover a certificate to *any site* who accepts it!
How can this possibly be *secure*?

- The requesting site never gets direct API access to client keys, keys are only supplied as a part of a specific application (in this case the HTTPS CCA protocol)
- The code running the client-side of the HTTPS CCA protocol and UI is a part of the *trusted client platform*, not something the [potentially malicious] site has provided
- Nothing is exchanged *unless the user explicitly grants the site access*
- Supports different privacy policies without requiring modifications on the requesting side
- Supports *security hardware* without requiring modifications on the requesting side

Note that the Web2Native Bridge can also support applications using the traditional SOP security model