Google

Payment Handler Proposal Review

Presentors: Danyao Wang (Google), Justin Toupin (Google)

Presentation given on March 30th, 2020 as part of the W3C Web Payments Working Group digital face-to-face

Agenda

I. Review threat assessment & proposals for addressing gaps

II. Review other proposals for PHs

I. Threat assessment

Context

- Payment Handler enables cross-origin coordination
- A threat assessment was done to proactively identify privacy and security gaps that need to be addressed
- It is critical to address these gaps before PH usage scales

Threat assessment summary

Full Analysis

Threat	Attack vectors
x-origin tracking w/o user interaction	• Colluding/compromised merchant transfers opaque data before show() to malicious PH • Malicious PH persists tracking data (via IndexedDB and network)
x-origin tracking w/o sufficiently clear user intent to interact	• Colluding/compromised merchant transfers opaque data via show() to malicious PH • Malicious PH persists tracking data (via IndexedDB, network and cookies), and possibly shares back with colluding merchant via PaymentResponse.
Phishing	• Colluding/compromised merchant invokes malicious PH who phishes the user for login credentials for legitimate PH
Fingerprinting	• Malicious merchant can bin the user based on hasEnrolledInstrument() response for different payment methods

Proposed mitigations fall into 2 categories

1. Mitigations that are **self contained**
2. Mitigations with more **complicated implications**

1) Self contained mitigations

A. Require user gestures [1.2, explainer coming soon]

B. Stronger UX indication of cross-origin switch [1.3]

C. 3P context by default [2.1]

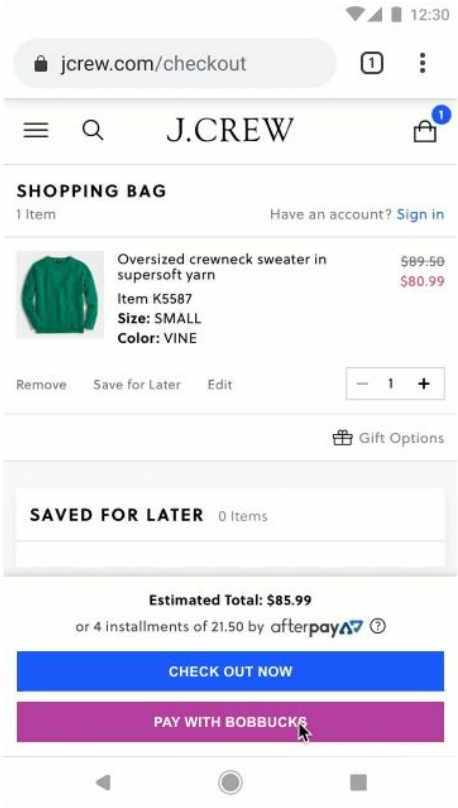
1A) Require user gestures

Summary	• Require a user gesture to trigger show(): already in spec; start to enforce in Chrome • Require a user gesture in PH window (i.e. w/ the web content) before showPromise() is allowed to resolve
Threat vector	• X-origin tracking w/o user interaction
Key open questions	• How does requiring user gesture to trigger show() affect implementations that make use of cross-origin iframes where the user gesture is not propagated along postMessage()?

1B) UX indication of cross-origin switch

Designs are directional explorations

Summary	• No associated spec changes • Alert the user during PH loading • Put more emphasis on the URL
Threat vector	• Phishing • X-origin tracking w/ user interaction
Key open questions	• Additional UXR needed to validate efficacy



1C) Move PH to 3P context

Summary	• One-time user consent required for PH (service worker & web content) to access 1P storage • Otherwise, PH only has access to 3P storage (same as cross-origin iframe)
Threat vector	• X-origin tracking • Fingerprinting
Key open questions	• What is the UX for requesting consent? This is coupled to whether we pursue explicit install (slide 12) • How do we properly sandbox the PH's service worker to 3P storage access when service workers typically have access to 1P storage?

2) Mitigations with more complicated implications

We are aware of three options to further mitigate x-origin tracking and phishing:

A. Explicit user install [1.1]

B. Payment Handler vetting [N/A]

C. Threat vector removal [N/A]

Notes:

- *Options are independent but not mutually exclusive*
- *Still early explorations*

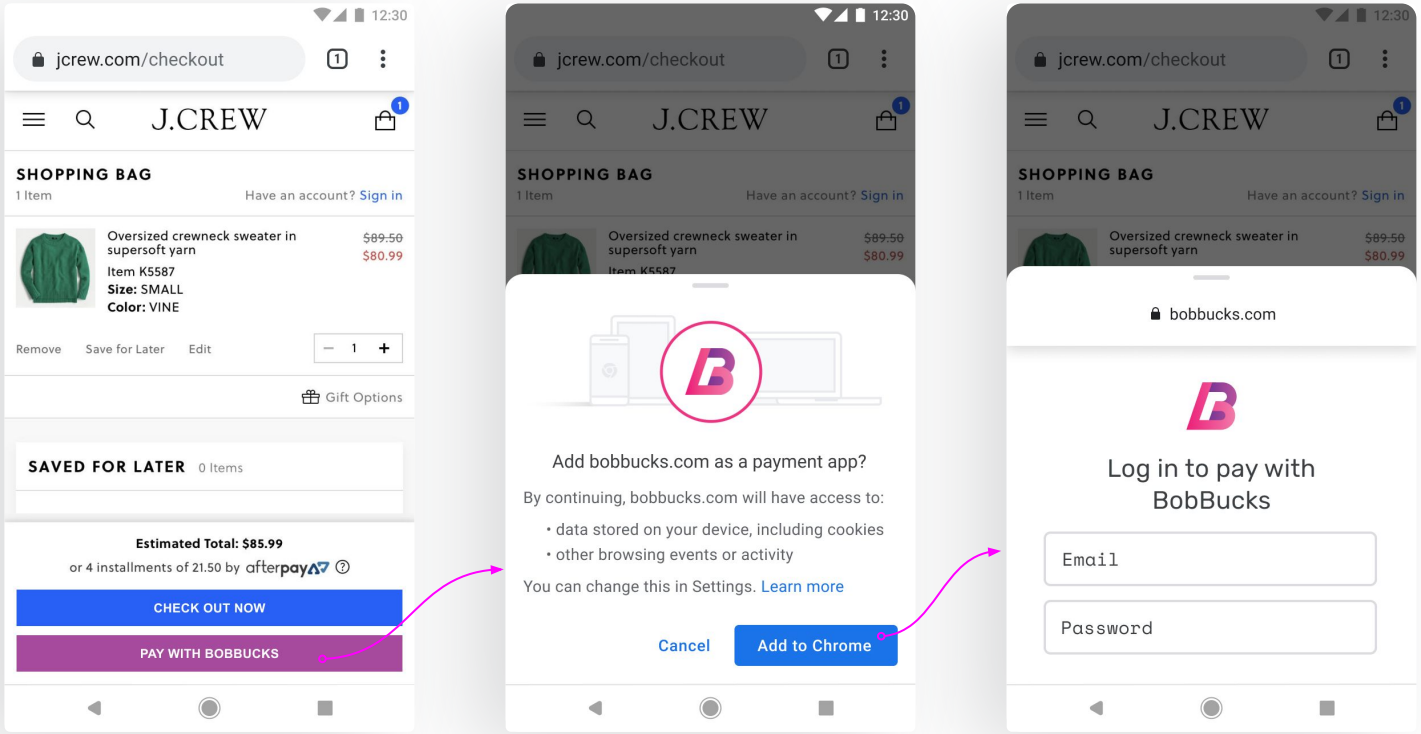
2A) Explicit user install

Summary	• PHs need explicit user consent in order to access local storage and hEI() • Consent is browser mediated and could be captured at first use OR at time of install

2A) Explicit Install

Designs are directional explorations

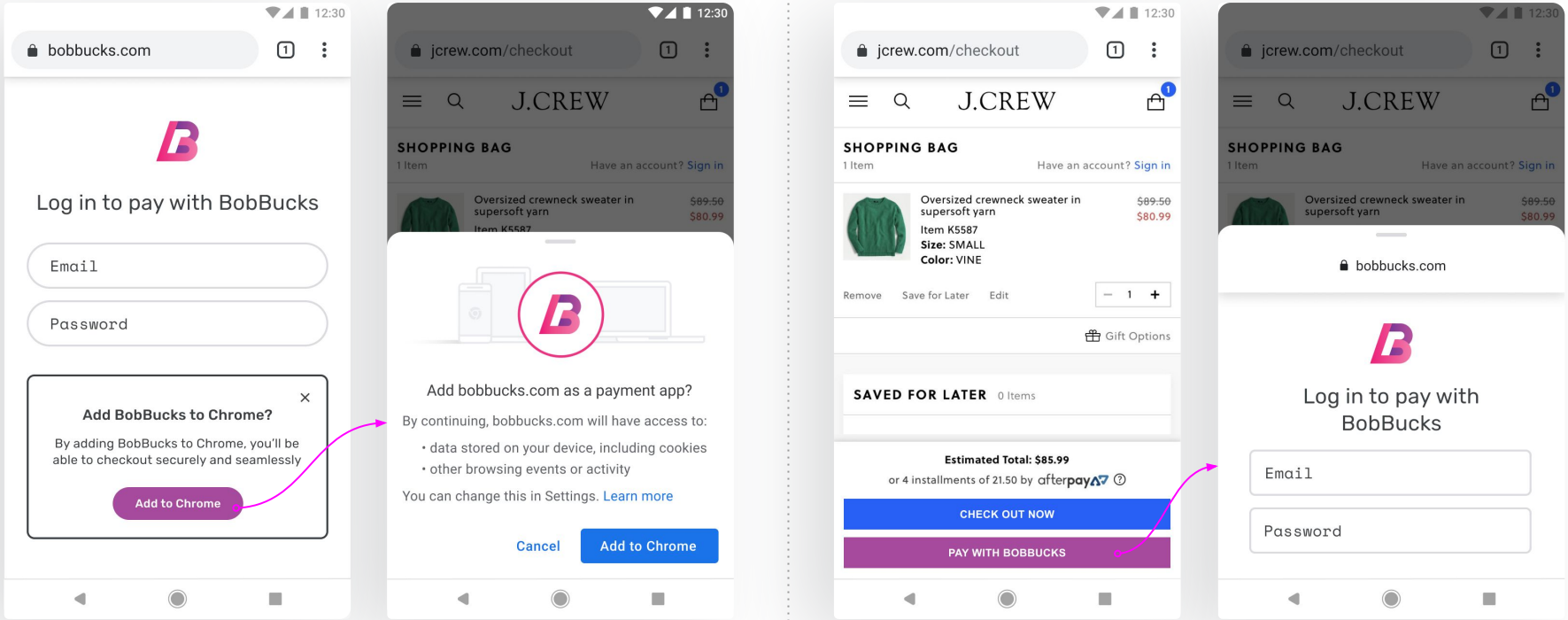
Install @ first use [part of a transaction flow]



2A) Explicit Install

Designs are directional explorations

@ time of install [outside of transaction flow]



2A) Explicit Install

Pros	Cons
• Most transparent option from a user experience perspective • Potential helps improve user awareness of PHs	• New consent step will create additional drop-off for PHs
Key open questions	• Does this make PHs unviable to Payments Apps? • What happens when consent is declined?

2B) Payment Handler vetting

Summary	• Only vetted PHs would have access to 1P local storage and hEI()
	• Vetting could be handled a number of ways:
	  ○ Per-browser whitelist
	  ○ Global registry shared by other browsers (e.g., “likely trackers” proposal)

Pros	Cons
• Low incremental user friction • Minimal change for vetted PHs	• Increases PH developer onboarding friction • Adds a gatekeeper

Key open questions	• How could vetting work at sufficient scale and consistently across browsers?

2C) Threat vector removal

Summary	• Assumes local storage limitations are added (see: 3P context proposal) • Remove or replace hasEnrolledInsturment()
Pros	• Low incremental user friction • No coordination required between browsers and developers
Cons	• Break use cases that depend on hEI() • Technically intricate to get right, also for developer ergonomics
Key open questions	• Is removing hEI() feasible? Are there viable alternatives?

II. Other proposals

Two proposals to discuss

1. Separate canMakePayment() from hasEnrolledInstrument() [1.4]
2. No partial delegation [N/A]

Separate canMakePayment() & hasEnrolledInstrument()

Summary	• Both cMP() and hEI() defaults to true, unless any PH responds false • PH receives different events in response to cMP() & hEI() • Events are not triggered in private browsing mode • Does not impact threat assessment vectors
Why	• Unbreak payment handler in private browsing mode; today hEI() always returns false.
Open questions	• How is the implementation experience of merchants for hEI()? • Should we fix hEI() or remove it? (see slide 17)

No partial delegation

Summary	• Chrome now supports full delegation for web-based Payment Handlers and native is coming soon • In the long-term, proposing that all info requested by the merchant must come from a PH (i.e., no mix of info coming from the browser and PH)
Why	• Clarifies the user experience & reduces complexity
Open questions	• We have not heard of strong use-cases for partial delegation, are there any?