Mozilla / Telefónica Joint Paper for W3C Payments Workshop

2014

Authors:
Kumar McMillan (Mozilla), Fernando Jiménez (Telefónica)

What Does The Web Need For Payments?

Payments work pretty well on the web. You can go to any website and buy something fairly easily and securely. You enter payment details into a form and you transmit those details securely over TLS/SSL to the website. The website processes the payment and delivers you the items.

However, there are a few problems.

Security

When paying with a credit card, giving your credit card number to every website is very risky. A credit card number provides access to a large chunk of your income when you're really only trying to provide access to the exact amount of the purchase. You have to trust that a website will charge you only the exact amount and you have to trust that the website can prevent others from intercepting the credit card number.

Slow first time payment

The first time you buy something on a website you have to enter in a lot of information, such as name, address, credit card number, expiration date, etc. This is tedious and cumbersome, especially on mobile devices.

Repeat customer advantage

Once you enter all your personal details it's tempting to re-use only a certain payment provider to avoid re-entering all your details. If you are presented with a choice of payment providers you have an incentive to choose the one that provides the quickest route to the goods.

Asset ownership

There is no standard way to express a digital receipt for an asset that you bought. If you purchase an MP3 from Apple only Apple can vouch for the fact that bought that MP3. You couldn't, say, stream the MP3 directly from the artist's website because the artist has no way to verify that you purchased their music already.

Merchant burden

As a small company it's often a burden to set up a server to process payments. In most cases one has to meet strict government compliance that costs time and money. There are only a few web solutions that make it easier for merchants to process payments.

Privacy

Merchants typically must collect a lot of personally identifiable information from a customer when ultimately they just want to charge money for an item. A lot of this personal information is not mandatory for processing a payment.

Phishing

The web suffers from many phishing attacks where the user is tricked into divulging sensitive information (such as a PayPal password) to a third party. There isn't really a good solution on the web for this; there are URL bars but those only help people who already understand how the web works. On mobile, there are barely even URL bars. On mobile, a good payments solution would mitigate phishing attacks.

These are some problems that Mozilla and Telefonica set out to solve when implementing payments for the open web app ecosystem. We focused primarily on security and ease of use but also attempted to make an open system where many payment providers and many merchants could participate in. The rest of this paper details what we built, some lessons we have learned, and what concepts we think could be useful to the web as potential standards.

An End To End Payment Protocol

Mozilla and Telefonica built an end to end payment protocol that is controlled on a user agent with the experimental navigator.mozPay() API. It's an end to end protocol because you can't just

adhere to one part of it. If you want to be compatible with it you have to adhere to all parts.

Here is a brief overview of how a merchant interacts with the protocol:

- The merchant signs up for an access key on the Firefox Marketplace developer hub.
- They securely store their access key and secret on a server
- They sign a JWT (JSON Web Token) at the time of purchase using the secret key. This JWT specifies an intent to purchase an item at a specific price.
- The JWT is passed to navigator.mozPay() where the user agent starts a payment flow.
- If the user completes the payment successfully, the merchant's website receives a callback with another JWT that can be verified using their stored access key and secret.
- In the case of purchasing a downloadable app, the merchant additionally will receive a receipt with which they can verify app ownership.

Here's how a user interacts with the payment protocol:

- A user taps a buy button on a merchant's website for something they want to buy.
- The user is asked to sign in with Mozilla Persona account. The sign-in process involves entering an email address or a password. On Firefox OS devices you only have to do this once.
- The user is asked to create a PIN code. For each purchase, the user must re-enter the PIN code.
- The user sees a purchase screen that summarizes the item and price and offers a choice of credit card payment or direct billing, depending on the region and operator.
- The user chooses an option and completes the payment.
- The user is redirected to the merchant's website where they immediately receive the goods.

Here are some highlights of what the mozPay() protocol achieved:

- **Security**
    - Merchants only initiate an intent to purchase and verify the result. They never handle credit card numbers or any kind of data that can be used to deduct money from a customer's account.
    - Users always must enter a PIN before making a purchase. If their phone is stolen, it would be somewhat difficult for a thief to make purchases on it.
- **Privacy**
    - Merchants never handle personally identifiable information such as billing addresses.
- **Decentralized asset ownership**
    - The receipt protocol allows many payment providers to offer payments for many

merchants. The merchant's server can verify each receipt independently. There is no central authority dictating who owns what. The most concrete example is purchasing an application. If some app named Adventure Game wants to charge money for installs, it can do so through the Firefox Marketplace or Some Other Marketplace. If a user purchases an app through either marketplace, the Adventure Game doesn't really care which one it is. The app will receive a receipt which can be verified according to protocol.

- **Ease of use**
  - It's very easy for users to make repeat payments. They tap a buy button, enter a PIN, and tap OK to process the payment. The payment setup is relatively easy too. They must sign in with an email address and password, create a PIN, and then they are ready to make their first purchase.
- **Direct billing**
  - Users can make very quick payments on mobile devices by putting the charges on their operator bill. In the best scenario this is as easy as tapping a button. In some regions the user must first authenticate their phone via SMS challenge / response. Once their device is configured for direct billing, repeat payments are very quick.

Problems With An End To End Protocol

Ultimately, we don't recommend standardizing the exact end to end protocol we built as is. It's too restrictive and not flexible enough for the needs of today's web. Here are some problems with the protocol:

Backwards incompatibility

There is no known existing payment provider who could easily support the mozPay() protocol without considerably re-building their system. For anyone to support the protocol they have to implement many pieces. Because of this barrier to entry, it would be hard for the web to adopt mozPay as a standard.

New features

What does mozPay offer the web that it doesn't already have? There are only a few features such as direct billing (more details on this). Most of the mozPay flow is powered by existing web technology and does not offer anything new.

Difficult to become a payment provider

Arbitrary websites cannot process payments that are initiated through mozPay. Any merchant website can *accept payments* but it is too risky to allow any website to *process* payments directly using the internal mozPay API. The result is that Mozilla is the only payment processor that ships on Firefox OS devices. This access is controlled by a whitelist within the device preferences. This does not sound like a recipe to a successful web standard.

After taking a step back and looking at these achievements and problems it seems like an end to end approach wouldn't be beneficial to the web. Instead, there are some lower level components, or building blocks, that would be more beneficial to the web for payments.

What Are The Low Level Components?

- **Payment Tokens**
  - These encapsulate an intent to purchase an item at a specific price
- **Asset Receipts**
  - These are decentralized receipts that can be used to verify ownership
- **Direct Billing functions**
  - The web is missing some low level APIs to make operator billing easier
- **Request Autocomplete**
  - Instead of repeatedly filling in common form parameters, autocomplete them

Payment Tokens

One of the biggest security problems with payments on the web is passing around credit card numbers. If credit card companies and banks were to settle on a standard way to accept a payment *token* then payments would become much more secure. By token I mean a representation of the intent to buy an item at a specific price. MasterCard and others have announced plans to work on this but there aren't many public details yet.

Mozilla / Telefonica have worked on a token protocol via mozPay() where a merchant encapsulates the intent to purchase an item at a specific price through a specified JWT. This concept could be expanded and generalized into something more flexible.

Asset Receipts

Mozilla's receipt protocol is a powerful way to express ownership of digital goods without a central authority. The receipt format could be standardized as is and could also be expanded to support different types of licensing such as ones requiring recurring payments. Every merchant needs a way to verify a purchase. If there was a standard way to do verification (such as a receipt protocol) then it would make implementation easier. Decentralized receipts also help liberate users from ecosystem silos. Receipts make it possible for many payment processors to interact with many merchants all across the web, not just in proprietary ecosystems.

Direct Billing API

Mozilla and Telefonica's work on adding direct billing features to the mobile web would be beneficial to other payment processors who intend to interact with mobile devices in the same way. The state of direct billing on the web is not great. All present day solutions either require insecure HTTP header injection or hacks with SMS messages. However, there are a few features that Mozilla / Telefonica has added to the Firefox OS platform which would be helpful to others who want to do direct billing on the web.

Silent SMS

Instead of asking a user to check their SMS inbox, memorize a code, and enter it on a small mobile screen, this handshake process is done silently in the background.

SIM changes

Payment processors need to know when a user removes or inserts a SIM card so that the processor puts charges on the right account, even in a multi-SIM scenario.

Region/network detection

Payment processors sometimes need to know what region or network the user is on in order to offer the correct payment options. GeoIP detection is not accurate enough because of how operators use proxies.

There is a draft of a proposal for a Direct Billing API[1] that would benefit existing payment providers who do not wish to implement an end to end payment protocol such as mozPay.

Request Autocomplete

Request autocomplete is not currently used in the Mozilla / Telefonica payment flow but it solves a major problem with web payments today: entering data. If you enter your billing address on one payment processor you have to enter it over and over again for each new payment processor. With request autocomplete you would only have to enter it once. For mobile devices this would be a huge improvement as it is very cumbersome to enter a 12 digit credit card number as well as a billing address.

Summary

In working so closely on payments, we at Mozilla / Telefonica have learned that an end to end protocol is too restrictive for the web. The web needs lower level building blocks to make payments more secure, easier to implement, easier to use, and open to all parties.

[1] https://wiki.mozilla.org/WebAPI/Direct_Billing