Client Side Granular Consent

Micajuine Ho
Original Doc

Purpose

High-level: Expand `amp-consent` to allow developers to collect users consent decisions for individual categories and or cookies. Create a mechanism within amp-consent to make purely client side decisions to block publisher chosen components.

`amp-consent` currently collects a singular global consent to block and unblock components tagged with the `data-block-on-consent-purposes` attribute as well as pass this consent decision to vendors to make server-side decisions. This new feature will introduce a client-side mechanism to block certain AMP components based on “purposes” in order to abide by privacy regulations without vendors making server side decisions. developers should be able to define purposes and/or cookies used (i.e. marketing, analytics, and specific vendors) and tell AMP which components should be blocked if a user chooses to not give their consent for a certain purpose/cookie.

Design

AMP will provide a new tagging attribute for AMP elements called `data-block-on-consent-purposes`. Developers will be able to assign this attribute different consent purposes to be blocked for that element. For example, `data-block-on-consent-purposes=”ads,analytics”` will block the element until `amp-consent` has collected the ads and analytics purpose consents. AMP will collect these consents by letting developers display their granular consent prompt via inline prompt or iframe. Both of these will options will result in a purposes consent map being collected and used to unblock components with the `data-block-on-consent-purposes` attribute. Both `data-block-on-consent-purposes` and the existing `data-block-on-consent` attribute can be used on the same page.

Developers will be able to sync the user’s granular consent decisions to their server endpoint, as well as send AMP the most recent user’s decisions. Just like how it is with global consent, if the purpose consents are found in localStorage, they will be used. Updates to the consent decisions via `checkConsentHref` will be reflected upon next visit.

This approach is straightforward, with clear behavior that requires developers to be explicit in what purposes need to be collected and what elements need to be blocked. This approach should fulfil the immediate needs of client side granular blocking for regulation compliance. Additionally, we can expand the scope of this feature once developers ask for it.

Features

purposeConsentRequired

A new `purposeConsentRequired` API will be accepted in both the inline consent config and the `checkConsentHref` responses. It will be an array of publisher defined purposes indicating what consent purposes should be collected. There will be no changes to `consentRequired`, however, now if `purposeConsentRequired` is a non-empty array, then that will signal to AMP that we will be expecting and storing a purpose consent map (based off of the values in the array) to unblock components (in addition to collecting and storing a global consent state).

```
// From checkConsentHref endpoint or via inline <amp-consent> config
purposeConsentRequired: [‘purpose-analytics’, ‘purpose-foo’, ‘purpose-bar’]
```

data-block-on-consent

`` `data-block-on-consent-purposes` `` is a new attribute that developers can use to tag AMP elements to only unblock after **all** the consent purposes are granted. The attribute will be a string that is a comma delimited list of purposes.

```
New Granular Consent Blocking

// blocked by AMP runtime
<amp-foo
data-block-on-consent-purposes=”purpose-analytics,purpose-foo,purpose-bar”>

// blocked by AMP runtime
<amp-bar data-block-on-consent-purposes=”purpose-bar”>
```

promptUI

The `promptUI` API lets you customize your inline consent prompt, utilizing the `accept`, `dismiss`, and `reject` prompt actions to signal to AMP the global consent decision. A new `toggleConsent(purposeConsentName=...,)` API will be used to help with the granular consent UI flow.

`toggleConsent(purposeConsentName=...,)` - The arguments of `toggleConsent` will tell AMP to set the state for the `purposeConsentNames` to `true` or `false`. Since these purpose consents only have valid values of ACCEPTED and REJECTED, they will map to true or false respectively. A list of args can be passed in as well.

```
// Set purpose-foo to ACCEPT
<button on=click: MY-CONSENT-INSTANCE-ID.toggleConsent(purpose-foo=true)>
// Set purpose-foo to ACCEPT, purpose-bar to ACCEPT and purpose-xyz to REJECT
<button on=click:
MY-CONSENT-INSTANCE-ID.toggleConsent(purpose-foo=true,purpose-bar=true,purpose-xyz=
false)>
```

The `accept` and `reject` actions will still be used to pass along the global consent decisions and close the window. If a consent isn't toggled (i.e. it's neither accepted or rejected) a default value will be used and stored for this consent purpose. This default value can be customized by the argument `` `purposeConsentDefault=accepted/rejected` `` added to the `accept` or `reject` prompt actions.

For the `reprompt` action (when a user wants to change their consent decision within the same page visit), new consent decisions will override existing ones.

Full example:

```
<amp-consent>
<script>
{
  consentInstanceId: 'myConsent',
  consentRequired: 'remote',
  checkConsentHref:  'myConsentServerEndpoint.com/checkConsentHref',
  promptUI: 'ui'
}
</script>

<div id='ui'>
  <label for="consent-purpose-analytics">
    <input
      type="checkbox"
      on="change: ABC.toggleConsent(purpose-analytics=event.checked)"
      id="consent-purpose-analytics"
    >
    Accept Analytics Consent
  </label>

  <label for="consent-purpose-foo">
    <input
      type="checkbox"
      on="change: ABC.toggleConsent(purpose-foo=event.checked)"
      id="consent-purpose-foo"
    >
    Accept Foo Consent
  </label>
```

```
  <label for="consent-purpose-bar">
    <input
      type="checkbox"
      on="change: ABC.toggleConsent(purpose-bar=event.checked)"
      id="consent-purpose-bar"
    >
    Accept Bar Consent
  </label>

<!--
Even if no elements are blocked with data-block-on-consent, still need to accept or
reject.
 -->
  <button on="tap:ABC.accept(purposeConsentDefault=accepted)">
    Save
  </button>
  <button on="tap:ABC.dismiss">
    Close
  </button>
</div>
</amp-consent>


// checkConsentHref response
{
  consentRequired: true,
  purposeConsentRequired: [‘purpose-analytics’, ‘purpose-foo’, ‘purpose-bar’],
  consentState: unknown,
  purposeConsentMap: undefined,
}
```

promptUiSrc

For `promptUiSrc`, the iframe will be created and the current stored purpose consent values as well as the stored `purposeConsentRequired` values will be sent to the iframe. Then the iframe will send an accept or reject postmessage to AMP along with a new `` `purposesConsentMap` ``, which will be stored and used to unblock elements.

```
<amp-consent type=”_ping_”>
</amp-consent>

// checkConsentHref response
{
```

```
  consentRequired: true
  purposeConsentRequired: [‘purpose-analytics’, ‘purpose-foo’, ‘purpose-bar’],
  consentState: unknown,
  purposeConsentMap: undefined,
}

// promptUiSrc’s iframes postmessage response:
{
    type: 'consent-response',
    action: 'accept',
    purposeConsentMap: {
        “purpose-analytics”: accept,
        “purpose-foo”: reject,
        “purpose-bar”: accept
    }
}
```

checkConsentHref

Syncing of the users' purpose consent decisions and purposes required can be done via `checkConsentHref`.

```
// checkConsentHref request body
{
  …
  // locally stored purposeConsentRequired
  purposeConsentRequired: Array of strings | undefined,
  // locally stored purposeConsentMap
  purposeConsentMap: Object<string: boolean> | undefined,
}

// checkConsentHref response
{
  …
  purposeConsentRequired: (optional) Array of strings
  purposeConsentMap: (optional) Object<string: boolean>
}
```

What is required by developers and CMPs

If working together, developers and CMPs have to agree on what elements are going to be tagged with what consent purposes. That way the CMPs can send the correct

`purposeConsentMap` along with the `purposeConsentMap` so that AMP can sync those values and unblock the correct elements.

In the end, developers will have to retag their pages to signal to AMP which components need to be blocked by which purpose consents. They will also need to build a new `promptUI` or an iframe to handle the collection of purpose consents. `purposeConsentMap` must only have string values (aka no purpose objects).

Implementation Details

- Let developers tag the elements on their page with `data-block-on-consent-purpose="pub-purpose-1,pub-purpose-2,etc"`
- No customization of the blocking behavior (no policy, not even default)
- Only accept or reject states for purpose consents
- Developers/CMPs can define what purpose consents need to be collected `purposeConsentRequired: ["pub-purpose-1","pub-purpose-2","etc"]`
- UI will prompt when we have no stored consent values for each purpose in `purposeConsentRequired` and consent is required (hasStoredInfo())
    - `promptUI` will use new `toggleConsent`() to set purpose consents and expanded `accept()`/`reject()` functionality to allow developers to set a default for untoggled consents
    - `promptUISrc` will pass in an additional `purposeConsentMap` as an argument in the accept/reject signal post message
- Our locally stored `purposeConsentMap` and `purposeConsentRequired []` will be passed via `checkConsentHref` and ClientInfo (within iframe)
- Will store the `purposeConsentMap` and then `purposeConsentRequired` array in the local storage
- Will unblock based upon what blocking identifier is found on each element. Elements will only unblock when all the correct purpose consents are given (if the purpose consent was never collected then it will not unblock (as this is a dev error))
- Will still have off-by-one trade off

If elements are tagged with both `data-block-on-consent` and new identifier, global consent will take precedence. This also means that we can have both `data-block-on-consent` and `data-block-on-consent-purpose` on a page.