Identifying Cross-origin Resource Status Using Application Cache

2015 Network and Distributed System Security Symposium

Sangho Lee, Hyungsub Kim, and Jong Kim
POSTECH, Korea

February 9, 2015
Web, HTML5, and Threats

• Web and HTML5
  • The most popular distributed application platform
  • Rich functionality introduced by HTML5

• Security and privacy threats
  • Popularity attracts a lot of adversaries.
  • Rich functionality opens security and privacy holes.

• Discovering unrevealed threats of the Web and HTML5 is important.
HTML5 Application Cache (AppCache)

• Enabling technology to offline web application
  • Specify resources to be cached in a web browser
  • Allow fast and offline access to the cached resources

• Potential threat of AppCache
  • Arbitrary cross-origin resources are cacheable.
    • Neither server- nor client-side control
  • Error handing can breach user privacy.
    • Recognize whether a user can cache specific resources
Motivation and Goal

• Motivation
  • In-depth security analysis of new web functionalities is necessary.
  • Security analysis of AppCache is insufficient despite its wide deployment.

• Research goal
  • Analyze and solve security problems of AppCache
    • Discover security problems of AppCache
    • Suggest an effective countermeasure against the security problems
Contents

• Introduction

• AppCache Details
  • Declaration
  • Procedure and Failure
  • Non-cacheable URLs

• URL Status Identification Attack

• Discussion

• Conclusion
AppCache Declaration

```html
<html
manifest="example.appcache">
...
</html>
```

HTML document declaring AppCache

CACHE MANIFEST

CACHE:
/logo.png
https://example.cdn.com/
external.jpg
NETWORK:
*
FALLBACK:
/ /offline.html

AppCache manifest
AppCache Procedure

Visit a web page declaring AppCache

Fetch and decode the manifest

Download the resources listed in the manifest

Re-fetch the manifest to check changes
When Does AppCache Fail?

- Visit a web page declaring AppCache
- Fetch and decode the manifest
- Download the resources listed in the manifest
- Re-fetch the manifest to check changes

Any failure rolls back AppCache to maintain content consistency.

Invalid or erroneous manifest

Non-cacheable resources

Changed manifest
Non-cacheable URLs

• Invalid URL
  • No content to be cached

• Dynamic URL
  • Caching dynamic content is less meaningful.
    • Cache-Control: no-store or no Content-Length

• URL with redirections
  • Final URL can be dynamically changed.
  • Violation of the same-origin policy is possible.
    • Refer a cached resource with the URL specified in a manifest
Contents

• Introduction
• AppCache Details
• URL Status Identification
  • Basics and Advantages
  • Attack Procedure
  • Concurrent Attack
  • Application: Determining Login Status
• Discussion
• Conclusion
URL Status Identification

• Basics
  • Specify a target URL in an AppCache manifest
  • Check whether AppCache succeeds or fails

• Advantages
  • Deterministic identification: Don’t measure timing
  • Identification of URL redirections
  • Scriptless attack
Attack Procedure: Cacheable URL

- Visit a web page declaring AppCache
- Fetch and decode the manifest
- Download the target resource
- Re-fetch the manifest to check changes
- Re-fetch the manifest to check changes
- Page refreshing lets AppCache check the manifest’s changes.

**Succeed**

**Refresh (optional)**

**Record browser info.**

**Identify success**
Attack Procedure: Non-cacheable URL

Visit a web page declaring AppCache

A browser don’t re-fetch the manifest when the target URL is non-cacheable.

Re-fetch the manifest to check changes

Visit a web page declaring AppCache

Page refreshing initiates an AppCache procedure from the beginning.

Record browser info.

Identify failure

Refresh (optional)

Fail
Concurrent Attack

Concurrently inspecting multiple target URLs with multiple iframe tags, web pages, & manifests

```html
<html>
<iframe src="attack_each.php?target=http://target1.com"/>
</iframe>
<iframe src="attack_each.php?target=http://target2.com"/>
...
</html>

attach_all.php

<html>
manifest="manifest.php?
target=http://target1.com">
</html>

CACHE MANIFEST
CACHE:
http://target1.com
NETWORK:
*

manifest.php

<html>
manifest="manifest.php?
target=http://target2.com">
</html>

CACHE MANIFEST
CACHE:
http://target2.com
NETWORK:
*

attach_each.php

...
Application: Determining Login Status

Determine login status by inspecting URLs with conditional redirections or errors

amazon.com/gp/yourstore/home → amazon.com/ap/signin?
tumblr.com/dashboard → tumblr.com/login?redirect_to=/dashboard
youtube.com/feed/subscriptions → accounts.google.com/ServiceLogin?

URLs redirecting non-logged-in browsers to login pages

bitbucket.org/account/user/<user-id>
github.com/<user-id>/<repository-name>/settings
<blog-id>.wordpress.com/wp-admin

Private URLs returning errors to unauthorized browsers
Contents

• Introduction
• AppCache Details
• URL Status Identification Attack

• Discussion
  • Problematic Countermeasures
  • Countermeasure: Cache-Origin
  • Service Worker

• Conclusion
Problematic Countermeasures

• Ask user permission for AppCache
  • Vulnerable to careless users

• Always/never check changes in manifests
  • Vulnerable to page refreshing attacks
  • Content inconsistency problem

• Eliminate web pages having conditional behaviors
  • Detection and modification of all vulnerable web pages are challenging.
Countermeasure: Cache-Origin

• Attach a Cache-Origin header when requesting resources during AppCache
  • Contain the manifest’s origin
  • Notify a web application of who initiate an AppCache procedure
  • Resemble the Origin header of CORS

• Abort suspicious AppCache procedures by returning no-store or error code
  • Cache sensitive resources
  • Be initiated by doubtful servers
Service Worker

• Provide scriptable caches as an alternative to AppCache
  • Intercept and respond to network requests from certain web pages

• Have the same policy to handle URL redirections and errors with AppCache
  • Also vulnerable to our attacks
Conclusion

• We introduced a new web privacy attack using HTML5 AppCache.
  • Identify the status of cross-origin resources
  • Do not rely on client-side scripts
  • Can attack major web browsers

• We suggested a Cache-Origin request-header field to mitigate our attacks.
  • Minor variation of the Origin header
  • Easy deployment
Backup Slides
Script-based Identification

```javascript
var appCache = window.applicationCache;

function handleError(e) {
  // fail to download a given URL
  var img = new Image();
  img.src = "results.png?failure";
}

function handleCached(e) {
  // succeed to download a given URL
  var img = new Image();
  img.src = "results.png?success";
}

appCache.addEventListener('error', handleError, false);
appCache.addEventListener('cached', handleCached, false);
appCache.addEventListener('updateready', handleCached, false);
```
Execution Time of Concurrent Attack

- Firefox: 0.95 s
- IE: 0.27 s
- Chrome: 0.11 s for each URL

# of target URLs vs. Execution time (s)
Scriptless URL Timing

Visit a web page declaring AppCache

Fetch and decode the manifest

Download the target resource

Re-fetch the manifest to check changes

Measure elapsed time

Record browser info.

web browser

attack.com  target.com

NDSS 2015