Ankai Lou, Eric Olson, James Pan, Michael Chen
Professor J. Jones
Information Security
12 April 2016

Cookie Monster

Abstract

The advent of the Internet has created an exponentially growing market economy for sensitive user data. The web browser one such gateway for advertisers, businesses, and hackers to amass this 'virtual capital'. As such it has become imperative to patch the security vulnerabilities that exist in modern web browsers.

HTTP cookies are injected into browser storage to log information on browser sessions, save preferences, allow authentication, and various other functionality. Cookies are essential because they allow user personalization without authentication or a deeply embedded backend. Nevertheless, cookies pose several security risks, e.g. theft of private browsing data or providing an entry-point for code injection attacks.

The objective of Cookie Monster is to create a simple, intuitive browser extension that provides users with granular control over their cookies and actionable information on their browsing history. This allows the average user to be conscious of websites discreetly observing their personal information as well as the risks of negligent browsing habits. Awareness and initiative are an absolute necessity — not a commodity.

Design

Chrome Extensions are built on a front-end stack of HTML5, CSS3, and JavaScript. This allows extensions to be lightweight yet still have access to Chrome browser-specific APIs (e.g. tabs, storage, cookies, etc). The implementation of Chrome extensions requires an identical skillset to normal web development.

The first focus of the Cookie Monster Chrome extension was HTTP cookies. Cookies are an oft-overlooked method which website scrape user data, monitor browsing behavior, and even inject code into the browser. The power and ease-of-use afforded by HTTP cookies do not match up against the lack of effective tools for monitoring their usage. The initial objective of Cookie Monster was to provide functionality for accessing, modifying and monitoring the influx of cookies generated during daily web browsing. The initial design of Cookie Monster was based on uBlock Origin and intended to be as simple and unintrusive as possible.

The second focus was the users' web browsing habits. Browsing history is typically ignored until inevitably erased; yet there are clear, oft-unnoticed consequences associated with frequenting certain dangerous or suspicious sites. The feature-set of Cookie Monster was expanded to include a programmatically generated a report card of user's browsing activity. The information used to generated this report card was sourced from the *Web of Trust* — a site that uses community-generated reviews to rate sites on certain criterion (e.g. trustworthiness, child safety), tag websites by qualities (e.g. online tracking, misleading information), and maintain detailed blacklists for malware and other dangerous content. The *Web of Trust* provides an API for retrieving information from their databases as JSON using REST calls.

The top-down design of the Cookie Monster browser extension consists of a popup view, a dedicated whitelist page, and a dedicated report card page. The popup provides a single-click snapshot of the user's cookies and the quality of the user's browsing history. The dedicated options pages provide the user with the ability to manipulate their cookie whitelist by domain and view a comprehensive summary of their browsing behavior. This distinction provides a simple UI for casual users and more control for power users.

Implementation

Cookie Monster was implemented using HTML5/CSS5 to render the UI and JavaScript to process back-end logic. The extension requires the following Chrome permissions: tabs, cookies, history, both local and sync storage. These permissions are necessary for the extension to carry out its intended functionality.

A significant portion of the backend required asynchronous JavaScript calls to retrieve data, i.e. accessing synchronized storage, retrieving and deleting browser cookies, retrieving browser history, and generating *Web of Trust* API calls. Asynchronous function calls break the JavaScript event loop — often leading to undefined model variables and runtime errors. To mitigate this problem, the backend for both the Cookie Monster and Report Card sections of the extension were built using AngularJS.

AngularJS allows for real-time HTML-rendering and DOM-manipulation by maintaining a specific scope and model for each HTML view. The controller for a view only renders or re-renders elements when a change in the model is observed. The extension relies heavily on API calls to remote servers, e.g. the *Web of Trust* API and Google Chrome servers. This means the backend could take several seconds to generate the desired state depending on Internet connectivity. AngularJS resolves this issue by postponing content rendering until *after* the aforementioned asynchronous calls have finished.

The extension UI/UX integrated a combination of the Angular Material library, Bootstrap stylesheets, and the FontAwesome icon index. These components were all compatible with AngularJS and added polish, intuitiveness, and elegant built-in styling to the extension's HTML views.

The cookie deletion and whitelisting portion of the extension was implemented in two distinct event loops: a persistent background script for continuously tracking, deleting and modifying HTTP cookies; and a non-persistent controller script that renders snapshot and options views on-demand. The non-persistent controller updates both views by retrieving the current state of the background model. The encrypted whitelist is cached in chrome.storage.sync, accessible from both event loops, and modifiable from the non-persistent controller view. This implementation resolves the need for the extension to both be always-on in the background for cookie deletion and on-demand for the UI/UX and whitelist management.

The browsing history report card uses a single non-persistent event loop to generate the report card view. The event loop generates a REST call and summary view whenever the extension is opened by the user. The report card controller is non-persistent because running a persistent process in the background will overload *Web of Trust* servers with a continuous loop of HTTP GET requests.

The computation of the overall trustworthiness, child-safety, and average scores rendered to the snapshot used a special weighting scheme designed to provide more accurate summarization than a simple average. 'Poor' and 'Very Poor' ratings were weighted stronger than 'Good' and 'Excellent' ratings to prevent the occurrence of false negatives in the final Report Card; low confidence ratings were weighted proportionally less than high confidence ratings; and sites were weighted proportional to their visit count. This custom computation algorithm produced higher quality results that were resilient to bias and skew.

While non-persistent model variables are destroyed at the end of each event loop, the persistent background script for cookie deletion maintains its model during the entirety of the browser session. Furthermore, the Chrome API Documentation explicitly stated that chrome.storage.sync contents are not encrypted. The Stanford JavaScript Cryptography Library (SJCL) and a private 128-bit key was used to encrypt persistent and storage variables. This safeguards whitelists and cookie data from JavaScript injection attacks.

This completes the implementation of the initial release of the Cookie Monster Chrome browser extension. The 0.0.0 release was packed, deployed and published to the Chrome Web Store on April 10th, 2016. At this junction in the lifespan of the Cookie Monster extension, only basic unit-testing had been conducted to ensure no immediate app-breaking bugs were present.

Testing & Patches

Post-release edge-case testing revealed that the storage limit for chrome.storage.sync exhausted quickly, particularly if Cookie Monster is installed in conjunction with other extensions. This prevented users from synchronizing their whitelists using their Google Account. To mitigate this issue, additional functionality to import and export encrypted whitelists was implemented into the extension. Furthermore, the cookie deletion scripts would default to chrome.storage.local once the chrome.storage.sync failed to upload.

Another issue that surfaced during post-release testing was overly-aggressive cookie deletion would affect the css and functionality of certain sites (e.g. Outlook, Messenger, etc). The Chrome API Documentation provided a simple solution: disabling the injection of content-scripts into the current tab. The patch for this error consisted of zero source code modifications and an alteration of the manifest file.

Post-post-release testing revealed that chrome.storage.local was not a robust backup for chrome.storage.sync. Other Chrome extensions such as Imagus and Google Drive exhausting local browser storage rapidly as well. A second safeguard using a persistent background variable to store whitelists was implemented into the extension. This solution passed edge-case testing and was deemed a sufficiently robust solution.

These initial patches were pushed to the Chrome Web Store as early as April 11th, 2016.

Observations

The Chrome Extension API is poorly maintained and documented. Several event-listeners for chrome.tabs (e.g. onChanged, onVisited) are marked deprecated but were removed from the browser as early as 2012.

Additionally, existing chrome.tabs event-listeners such as onUpdated or onCreated were over-aggressively triggered by the background script. This leads to instances where the Cookie Monster extension will return the result "0 cookies deleted" due to the background process running multiple times on a single reload.

Both the chrome.storage.sync and chrome.storage.local APIs have no function for verifying a key exists in storage. This is a fundamental function for any key-value module derived from the JavaScript Object prototype. The solution to verifying a whitelist exists in storage before setting an initial empty whitelist is to call chrome.storage.sync.get() and to catch the chrome.runtime.lastError error. This is an unintuitive and unnecessary solution to a problem that should not be a problem.

AngularJS does not provide an intuitive way to nest ng-repeat directives for rendering nested variables, e.g. an Object of string-Object pairs. This was resolved by flattening nested data structures into several arrays. AngularJS also cannot access several important directives (e.g. ng-show, ng-hide) in a Chrome extension unless Angular-csp.js is also sourced by each HTML page that uses AngularJS. This makes very little sense as AngularJS is developed and maintained by Google.

AngularJS does not provide a way to set an ng-model value for HTML <input type="file"> tags. This can be mitigated using a custom Angular directive — there is still no intuitive native solution.

The *Web of Trust* API stores trustworthiness scores under the attribute '0' and child-safety scores under the attribute '4'. Furthermore, the '40*' classification group (e.g. 401, 402, etc) is the only set of classifications in the API that does not have a uniform category (e.g. Positive, Negative, Neutral, etc). This ambiguity in the documentation in conjunction with the tendency to frequently deprecate and alter their JSON formatting implies that the Cookie Monster dependency on the *Web of Trust* API will require constant supervision and maintenance. This affects the longevity of the extension once it stops receiving support.

Each of these roadblocks were mitigated before the initial release of Cookie Monster.

Results

Cookie Monster v0.0.1 is a feature-rich Chrome extension with both a simple and intuitive UI and an expansive set of tools for users to audit both their browsing behavior and the behavior of their browser.

The Cookie Monster section of the extension has the following functionality:

- Real-time notifications for cookies deleted
- Real-time tracking for all cookies deleted
- On-Off switch for disabling cookie deletion
- Functionality to add/remove from whitelist from popup
- Additional options page for viewing entire whitelist
- Add to whitelist by domain name or IP address
- Import/Export encrypted whitelist from file
- Synchronization of whitelists across machines with Google Account
- Exported whitelists, local storage & backend are encrypted with SJCL

The Report Card section of the extension has the following functionality:

- Snapshot view of final overall scores by category in popup view
- Comprehensive summary of browsing history accessible from snapshot
- Constantly updated site information via the Web of Trust API
- Color-coded report card for easily differentiating score brackets
- Identification of site qualities, e.g. malware-hosts, online-tracking, etc
- Identification of sites that appear on malware/phishing blacklists
- Robust algorithm to compute accurate and informative scores

The extension is currently published and available for download on the Chrome Web Store. The Chrome Web Store provides a feedback and bug-report option, which allows issues in the source code to be identified and fixed in a timely manner. Feedback also provides an outlet for crowdsourcing new feature ideas.

Given more time, the following functionality would be interesting:

- Functionality to to rate cookies — similar to the *Web of Trust* community
- Functionality to support multiple whitelists — and toggling each on/off
- Functionality to blacklist sites that do not meet a standard of quality and safety
- Functionality for users to send feedback directly through the extension
- User Login can whitelist synchronization that does not depend on Chrome Storage
- Custom pre-built, pre-loaded whitelists generated using the *Web of Trust* API
- Support for additional browser, e.g. Firefox, Opera, Safari, etc.