Building Private Applications on the Personal Web

Jean-Sébastien Légaré, Long Zhang, Alexandru Totolici, Mark Spear
Kalan MacRow, William Aiello, Andrew Warfield
University of British Columbia
Vancouver, Canada
{jslegare,zhlong73,totolici,mspear,kalanm,aiello,andy}@cs.ubc.ca

ABSTRACT

Users of the web trust third-party application providers with the safe keeping of their data. As the wealth of quality application services grows, so too does the volume of data that users disclose and entrust to others. This trust is precarious: services may go out of business or fail, while application provides may not take sufficient care of sensitive personal data, leading to data loss or unauthorized access.

We argue for a web application model that treats personal data differently. We first observe that the best practical backup of a user's data is to collect and store a complete log of their browser-website interactions, allowing them to search their browsing history for content and application state that they have viewed in the past. After showing that the collection and storage of this history is practical, we argue that many services that access personal data would be better implemented with access to the log, and in isolation from the Internet at large. This approach allows applications to be trusted with access to personal data without concerns over the leakage or unauthorized disclosure of that information.

We describe the design and implementation of our system, and then evaluate the model's potential with three useful applications: a search application working over all websites visited, a credit card fraud detection application, and a user self-profiling application.

1. INTRODUCTION

Will your online documents still be online in ten years? Would you give your credit card company access to your email account if it meant lower annual fees? Would you trust a new web startup with copies of your medical records or banking history?

The modern Internet, unlike any information system before it, demands unprecedented sacrifice from its users. While we, as individuals, all benefit from rich opportunities for communication and commerce, we must also increasingly *trust* the safety and stability of hundreds of individual online services to protect our data, indefinitely into the future. Whether it is account histories from a financial institution, documents in an online office tool, or play lists in a music sharing service, users have a broad range of valuable, personal, and often private data that they have entrusted to a large number of third-party services.

In this paper, we argue that the current state of Internet applications is incredibly precarious for its users in two regards: first, ***there are limited guarantees that data will remain available and accessible into the future***; catastrophic events such as hardware failure or corporate bankruptcy may result in services failing forever, and API changes in services that remain available may obscure or eliminate access to data that was previously accessible. Second, ***large-scale application service providers are actively campaigning to reduce users expectation for individual privacy***. In arguing for the value of personalization and using idioms such as "frictionless sharing", Internet services aggressively encourage users to expose more and more personal information. One major aspect of this exposure is that users receive value in exchange for their privacy: an online financial site such as Mint.com, helps a user manage their finances – in exchange for being entrusted with access to that user's financial institutions.

1.1 The Personal Web

This paper argues for the importance of a "personal" web. We believe that a great deal of the information accessed by individuals through a web browser is of considerable personal value and needs to be protected. This protection, however, should not limit the user's ability to take advantage of online services and especially to allow third parties to develop valuable tools that extend private user data.

We present the design and implementation of *Pando*, a continuous and persistent secure audit log that records all activities undertaken by a user through web browsers on any device that they use. Pando collects a complete and encrypted history of all of a user's web sessions, and allows the user to interact with this history at a later date in order to benefit from the data that they have seen in the past.

By preserving data in a secure log, Pando completely decouples the analysis of personal data from the active use of third-party web services. We propose that, in addition to providing an excellent long-term archive of personal state, Pando also allows useful applications to be developed by third parties and used on user data in a completely safe and isolated manner. Using Pando, a financial analysis application such as that offered by Mint.com may be downloaded and executed in a user's browser and granted complete access to that user's recent and historical interactions with their financial institutions. However, the application will run in an isolated environment, unable to expose the data that it is analyzing back to the service provider. Table 1 lists a set of additional examples demonstrating the sort of queries that a user might usefully ask of their browsing history; the details of this table, including the events that must be captured to answer these queries, are discussed in Section 3.

WWW2012 2012 Lyon, France

Description	Event Types: Nav	Form	Vis	Init	Stat	DOM	XHR	Example Query
Compare all of my friend's status updates that I've read last month. Was content deleted?			•		•			`at '*.facebook.com/genevere', select "table.uiInfoTable tr" parent_of ("th.label font") with_content "*status*" during 'October 2011'`
My password may have been compromised. On which sites do I use username 'U' as login, or password 'P' or MD5('P').	•		•				•	`at '*', select document.location when (form'*' match(...) or xhr(requestText) match(...))`
Of all the sites I've visited, did any use a vulnerable version of library X?			•	•	•			`at '*' select document.location when (stat.hash == 'md5' or stat.url match('*x2.3.js*'))`
Find all pages that I've read containing "Steve Jobs" subsequently to reading mail from Ed.	•				•	•		`DOM contains(``Steve Jobs'') within(5 min) of (at "*://mail.google.com/**" click on tbody tr with (td.innerText match (\bEd\b)))`

Table 1: Example Queries

In the remainder of this paper, we describe the design and implementation of Pando and explain solutions to a number of challenging problems that arose in trying to record and analyze the personal web. We believe that the work presented here represents an incrementally deployable, and technically viable solution to protecting personal user data, and that it invites individuals – both users and developers – to reconsider how web-based applications can be structured in order to protect sensitive user data.

1.2 Contributions

First, we present a client-side browser recording tool that builds a secure and durable log of a user's browsing history. The approach presented is practical and incrementally deployable in today's web environments.

Second, we introduce a novel application of existing security primitives to create web-application sandboxes that can access the user's audit log. After being downloaded and initialized, sandboxed applications perform an irreversible run-time transition between two discrete privilege modes: they gain access to the log, but relinquish their ability to interact with the rest of the network. After this privilege transition has taken place, it remains in effect until the browser tab containing the application is closed, destroying the application's state.

Third, we motivate the benefit of our system by describing three example applications, demonstrating the potential in securely interacting with a user's personal history:

1. ***Personal web search.*** A search engine over a user's browsing history offering free-text search as well as temporal and location-aware anchoring.
2. ***Personal anomaly and fraud detection.*** A tool that combines various aspects of a user's personal history to identify possible credit card fraud.
3. ***User activity profiling.*** A retroactive calendar application that allows a user to evaluate where they are spending time, and to understand how the applications they use are behaving.

2. GOALS AND NON-GOALS

The central idea in this work is to decouple the data that users *have accessed* from the services that they use *to access* it. We believe that for both the purposes of backing up data over time and for building interesting new services that use personal data, that the only viable approach to preserving and accessing data is to store a recording of the browser sessions in which they interacted with it in a secure, durable log.

The core motivation for our approach is that the applications that manage user data today are outside the control of the users that use them. For any number of reasons–technical, organizational, or otherwise–data that is stored on these services may not remain accessible in the future. As providers cannot be expected to maintain APIs that allow users to continuously take copies of their data, because tracking changes to these API when they do exist is onerous, *and* because the data of web-based applications depends critically on browser-side code for presentation and interaction, we believe that data rendered inside the browser is the best representation for preservation.

Further, we take the position that the secure log represents a considerably safer approach to accommodate the growing collection of applications that require users to grant permission for applications to access sensitive personal information, such as email accounts and financial institutions. By restructuring these applications to use the log, we can safely present them with both current and historical versions of sensitive data, while preventing them from leaking or otherwise disclosing this data to other parties. The remainder of this section discusses the specific goals and non-goals of our system.

2.1 Applications Over a Secure Log

Pando aims to support a rich set of third-party analysis tools that

can run on top of recorded personal user data, without divulging this information to these third-parties. We set out the following goals for our system, and for the applications that run above it:

1. *We should collect the most complete, high-fidelity recording of browser activity that is feasable.*

   While we believe that a recording of all activity within the browser is the ideal way to preserve user data, we also recognize that capturing this activity represents a major technical challenge. Our design acknowledges that while fidelity is important, incremental deployment over all the browsers that an individual uses is critical. Our prototype implementation, discussed in Section 3, explores one point in this design space: an extension for Google's Chrome browser. We believe that this approach demonstrates a useful mid-point, between pervasive browser modifications on one extreme, and proxy-based solutions on the other.
2. *They should be forbidden from leaking information to third parties.*

   Applications must be sandboxed within the browser, and only be provided with access to the Pando log. Note that where analysis applications need access to useful, general-purpose data (such as GPS coordinates) the user can arrange to visit relevant sites (e.g., Google Maps) on a periodic basis in order to have that data recorded in the log. This approach is bolstered by proposals for "background web pages" where client-side services may remain running and scheduled despite not having a visible tab.
3. *They should be prevented from modifying the log.*

   Malicious applications should not be able to modify or delete contents from the Pando log. The log must be presented to the analysis environment as a read-only API, avoiding tampering or rewriting of historical data.
4. *Log contents must remain encrypted and protected to the greatest degree possible.*

   Log entries should only be decrypted within an execution domain that the client trusts, and should be carefully protected. Analysis tool state should be carefully flushed and expunged upon termination.

In light of these goals, our approach is to:

- Sandbox applications. This means limit external communication and access to the DOM.
- Provide local, read-only access to the database. We run the database decryption in a local, protected runtime environment. This is currently a background process on the host, and we plan to move it to run within a NaCl environment in another tab of the browser.

This implementation provides a safe environment in which users may develop and share analysis tools with others, and they may safely experiment with third-party analysis tools without fear of leaking or corrupting data. At worst, third-party tools can present incorrect results from the log analysis, and so the veracity of analysis results should be treated with the same degree of suspicion as users treat any other data available over the web.

The following items are non-goals:

1. *Accessing third-party data as part of analysis.*

   For instance, an application that examines cell phone bills and advises a user on the lowest-cost cell phone provider will not be able to refer to an external database of cell phone rates. This database can, however, be included in the analysis tool as a static block of data. This rules out analysis that needs to perform correlation between the Pando log and large external datasets, but protects against information leakage (e.g., XSS-style embedding of content within URLs). We will explore techniques to relax this constraint as future work.
2. *Allowing third-parties to run analysis on the Pando log.*

   While the Pando log may be stored as encrypted data on a remote data store, our current, conservative model is that all analysis will be performed on a device that is owned and trusted by the user. Establishing trustworthy third-party compute resources remains an open problem. While the notion of third-party "services" might be attractive (for instance, trusting Google to access a user's Pando log to better personalize the web experience), we feel that the nature of Pando's data is sufficiently sensitive that care should be taken before allowing arbitrary third-party plugins and remote analysis.
3. *Auditing of websites for security exploits.*

   Some of the JavaScript instrumentations employed in our current implementation can be detected, which makes it unsuitable for auditing websites or detecting certain exploits.

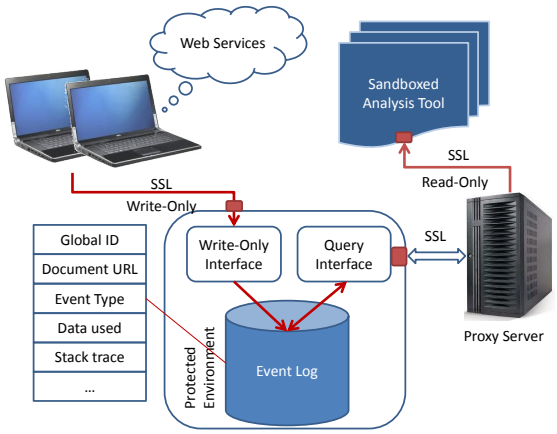

Figure 1: Pando organization sketch

3. IMPLEMENTATION

The overall architecture of Pando is illustrated in Figure 1. On the left side, we have the user's browsers running our extension. Recorders running in the browsers each authenticate to a web server that will receive their events. The sum of all events accumulating in the event log constitutes the personal web. The web server in charge of receiving these events is written in a safe language and is kept very simple.

Completely separate from browsing activities is the analysis environment, where tools can access the data contained in the event log, but are not allowed to expose results outside their environment. All external communications are funnelled through a secure web proxy.

By default, the proxy is configured to drop all connections, except for connections to the query-interface. Furthermore, the query interface is configured to only accept connections originating from that proxy. The analysis tools are thus confined to their environment, and the information contained in the user's personal web.

The following sections will cover each architectural component in depth.

3.1 Recording Engine

The recording engine must use recording techniques applicable to already deployed web browsers. In order to reach a wider audience, it must also be easy to install and configure. Pando's recorder ships as a Chrome Extension, and is implemented completely in JavaScript. It can be easily installed on any up-to-date Chrome browser in a few clicks. The only configuration settings needed are the address and credentials to the server exposing the log interface (covered later in Subsection 3.2). We expect users to install the extension on all browsers from which log capture is desired.

Even though our current extension depends on some features specific to Google Chrome's extension model, we are confident that they can be be ported to other JavaScript engines (such as Gecko for instance) with the help of the extension APIs they provide.

Capturing a browser log from within the browser is surprisingly hard. In the rest of this section, we present the the collection of events that are recorded in browsers running our extension, and describe the challenges faced when recording their different aspects.

For many of the different events, the core idea behind the recording technique relies on JavaScript program injection immediately before the contents of the page are loaded. Chrome provides the necessary hooks to extensions to instrument the content of every frame visited, before contents are loaded. This technique has been applied to other browsers in previous work [6, 9], and is also employed in some web frameworks such as Greasemonkey[1].

3.1.1 Static Resource Load (Stat)

To record a complete view of a webpage it is important to not only record the initial contents of a page, but also its auxiliary content attached to the page. Static resources consist of images, stylesheets, scripts, and embedded media objects depended on by the web application.

Recording GET for static content embedded in a page is quite challenging without the assistance of a forward proxy. We avoided employing a proxy for the recorder to preserve HTTPS and support scenarios where users already require a proxy to access the internet. The JavaScript APIs provided by Chrome Extension offer a very limited view on network requests made by the browser. Extensions also have no way to access the browser cache, or retrieve the payload of HTTP responses after-the-fact. Parameters users can see in DOMInspector (the built-in developer console), for instance, are unavailable to extension APIs at the moment.

However, Chrome extensions have the possibility to listen for a cancellable event, `beforeload`, dispatched before new resources (e.g. sripts, images, iframes, or a stylesheet) are loaded from an external URL. This gives an opportunity for the Pando extension to replace the traditional browser behavior with equivalent operations that allow recording of the content to be downloaded.

For every resource type fetch that we wish to record, our extension cancels `beforeload` events of those element types on receipt. This prevents the normal network requests from occurring, and we create an `XmlHttpRequest` object to perform a GET request at the URL provided as part of the event sent by the browser. The `XmlHttpRequest` object lets us read useful headers such as `Content-Length`, `Cache-Control` headers, custom `x-*` headers, as well as the original response payload.

One implication of downloading content this way is that we must respect the guarantees made by web-browser engines towards applications, in terms of both download and execution guarantees. The most complicated and most important element type to simulate correctly is the `script` element. WebKit has different ordering rules for scripts, depending on whether they are inlined or external (i.e. have a `src` attribute), or whether they possess `async` or `defer` attributes. Inline scripts are always executed as they are added to the DOM, regardless of their attributes, and behave similarly to external scripts with no attributes for which content must be downloaded and executed synchronously. Inline scripts are part of the containing page, and do not trigger `beforeload`. For any synchronous operation, we use `XmlHttpRequest`'s synchronous interface inside the `beforeload`, whose dispatch is also synchronous with parsing and execution of scripts living in the page.

Scripts with the `defer` or `async` attribute can be downloaded in parallel while the page is parsing, and so can be downloaded asynchronously with `XmlHttpRequest`'s asynchronous interface. Scripts marked with `defer` differ from script marked `async` by their ordering guarantees: deferred scripts execute in the order they are added to the DOM, whereas scripts marked with `async` have no ordering guarantees between them and other deferred scripts. In the particular case of WebKit, deferred scripts are also guaranteed to execute before the `DOMContentLoaded` event fires, and `async` scripts before the `load` event which always occurs after the former event. This varies across browser vendors.

Pando initiates the `XmlHttpRequest` download as soon as `beforeload` fires (as dictated by the engine's download rules), and enforces execution ordering by queuing scripts internally. To respect the execution ordering rules with respect to relative timing of load events, event handling mechanisms in the page are instrumented to wait for an additional signal from the extension before propagating both `onload` and `DOMContentLoaded` to the rest of the application's event handlers. To execute a script payload, a new script element must be created. The cancellation of the original script element unfortunately makes it non-reusable for executing code. The new script element's text content is set to the `XmlHttpRequest`'s response payload, prefixed with a statement that "self-destructs" the script in such a way that the JavaScript code is unable to get a handle on the element itself. Allowing the script to see the contents of the "ephemeral" script element would violate same-origin constraints. After the script completes executing, an `load` event is dispatched on the original script to preserve `load` semantics of scripts.

Other resources such as images and embedded objects can be downloaded in similar ways, and can use HTML5 `File` APIs, and `Blob` objects. We are investigating ways to record image content part of CSS not encoded as data URIs.

To download objects with `XmlHttpRequest` rather than using proxies, the set of permissions for our extension had to be adjusted to allow contacting all origins, rather than just the event log web server origin. The same-origin policy is still enforced in the page, as the extension and page reside in two isolated JavaScript environments, each with different restrictions.

The request headers sent by our extension `XmlHttpRequest` operations match the headers that would be present in regular static content fetches, so existing security mechanisms relying on Referer [sic] headers, for instance, still work.

The main limitation of this approach is that the fetching of static content for the document itself (e.g HTML or XML document) and frames cannot be overridden in a way that allows us to first record and then overwrite contents of the container. We are investigating approaches using HTML5's `window.open`. However, as we will describe in the next section, we can work around this limitation by capturing a snapshot of the frame content after its parsing is complete.

[1] http://www.greasespot.net/

3.1.2 Initial Load (Init)

The initial contents of the page are important to reconstruct an accurate view of documents when they are recovered from the log. The initial contents of a document also contain text that can be later indexed and queried.

WebKit defines two events that indicate different stages of a page being "loaded". First, a `DOMContentLoaded` event is fired when the browser completes the parsing of all of the document's text into DOM elements. Subsequently, when all of the scripts and embedded objects have been downloaded and made visible in the page, a second event "window.load" is fired.

Pando generates load status events at two critical moments. The first event is generated immediately before a frame starts loading a document at the start of Pando's boot sequence. The second fires as soon as the document finishes parsing, and before the application becomes notified that the loading is complete. Once the DOM is constructed, at `DOMContentLoaded`, Pando serializes the contents of the DOM. The two points in time for each event represent good "anchors" for building application queries[2].

3.1.3 Navigation (Nav)

Load events alone can provide us a list of websites that have been visited in the past, but make it difficult to determine the cause of the page transitions. Pando can fill in navigation semantics by capturing the methods used to navigate in an out of pages. Our extension can track the following navigation mechanisms:

- Direct interaction with address bar
- Interaction with the browser's chrome (e.g., favorites)
- Back and Forward history buttons
- Click on an anchor element
- Form submission (see Subsection 3.1.6)
- Opening a new tab or window (homepage), or closing it (navigating away)
- Programmatically (location . replace , location . reload , location . href , window.open, window.close, window.navigate, etc.)

This generally suffices to extract semantics for traditional website navigation. However, newer Web 2.0 applications tend to present a single page to users, from which new content is fetched via AJAX and the application view is modified with dynamic HTML. In addition to the mechanisms above, Pando also captures events susceptible of causing a significant alteration of the application's view:

- Changes to the URL fragment (location hash). Fragments are often used internally by applications to perform "routing" from a given URL into a corresponding application state.
- Clicks to DOM elements with registered click event handlers. Applications commonly rely on click events to trigger expanding menus, perform "button" functions, or change portions of a page into and out of edit-mode.

3.1.4 Dynamic Resource Load (XHR)

In order to allow for detailed analysis of applications, Pando records all of the issued AJAX requests and received responses, in addition to fetching static content Subsection 3.1.1. Our extension interposes on `XmlHttpRequest` communications to record these payloads and the associated headers.

[2]We considered using MutationEvents, but the interface is deprecated.

In our prototype, `XmlHttpRequest` operations are interposed with prototype overriding in the page, which means that the information contained in our records is limited to the information available to JavaScript. For security reasons, JavaScript cannot access some of the headers (such as cookies) and redirects are also hidden. Adopting a different recording mechanism built into the browser, adding native plugins, or employing a forward proxy would enable us to obtain more information, but would severely affect the application's deployability and security.

3.1.5 DOM Modification (DOM)

Web 2.0 applications modify their DOM repeatedly by adding, removing, or replacing existing content. Modifications to the DOM are necessary for updating the application's user interface, but are also commonly employed for loading additional library code in the page, or even as a form of communication, such as with JSON padding.

With Pando, these modifications can be searched and queried just as easily as static content or the initial snapshot of the page at load time (Subsection 3.1.2).

For methods of the DOM API that mutate the document structure (e.g. appendChild, removeChild, insertBefore , etc.), Pando relies on overriding the corresponding prototypes.

Browsers also define special properties with accessors that modify or inspect the document. For example, `Element`'s `innerHTML` property allows setting or retrieving the contents of a given element as an HTML string. Intercepting object properties is more difficult than object methods. In our implementation, we had to resort to defining JavaScript getters and setters[3]. There are at least two problems with this approach. First, unlike the methods on prototypes, the getters and setters have to be defined for each Node created. Pando must then in turn intercept every Node creation to modify those properties. Second, by defining getters and setters with the same name as an existing property, the original behavior is lost (at least in Google Chrome), so Pando simulates it by using equivalent, but slower, sequences of operations.

3.1.6 Form Submission (Form)

HTML Forms are generally used for communicating information to web servers: implementing login/logout mechanisms, changing application settings, or submitting comments and edits. Forms are of interest for Pando because they can be used to send user-specified content to a server.

Browsers generate events to notify the page scripts that a form is being submitted. In practice, these events are used to wrap form submissions with validation functions. When validation passes, the event is allowed to propagate, otherwise the application stops its propagation. If the propagation is uninterrupted, the browser initiates the default action for form submission events, that is the actual submission of the fields.

Form submission will replace the current application, and load a new page, leaving no opportunity to record the form event after the form is submitted. Because the default action cannot be overridden, the event must be recorded by Pando at some time *before* the default action happens. The recorder must find a suitable time to record the form contents, within the following constraints:

1. The recorded data must reflect the final state of the form fields.
2. Only submissions that have not been cancelled should be recorded.
3. No changes to the application should be required.

[3]defined in ECMAScript5

```
function form_bubble_handler(evt) {
 if (CANCELLED(evt))
   /* The form will not be sent: Do nothing. */
   return;

 /* Propagation stopped or top reached.
    Last handler to receive event. */
 if (STOPPED(evt) || evt.currentTarget === document)
   formrec.capture(evt.target); #synchronous

 /* let the event propagate */
}
```

Figure 2: Pseudocode for the end-of-line handler

For #1, our algorithm ensures that the last event listener to propagate a form submit event is one that has been registered by Pando, and not the application. Then, if we suppose that our algorithm always strategically places its handlers to be invoked last, #2 amounts to determining whether the event has been prevented, prior to invocation of the last handler.

Event propagation has been described in a lot of details in previous work [9]. In this section we shall focus on those details that pertain to our algorithm. *Preventing* an event affects only the course of the default action (whether form contents are sent over the network), but not the event's propagation. Default actions can be *prevented* either by calling preventDefault () on an event object, or by returning a false value from a DOM Level 1 intrinsic event handler (i.e. onsubmit=). The propagation can be *stopped* with stopPropagation (), which still allows the other handlers on the same DOM node to be invoked but not on following nodes, or stopImmediatePropagation () which causes the current event handler the be the last to receive the event.

Our algorithm first places a capturing handler on the root of the document. This capturing handler is always the first handler to receive submit events. When invoked, the handler looks at the target <form> element's hierarchy, and for each level that registered either a DOM level 1 hook, or one or more DOM level 2 submit event listeners, it installs an "end-of-line" bubbling handler. The end-of-line handler is placed such that it is always the last invoked handler for a given level in the DOM. We make this guarantee by instrumenting `addEventListener`. We also install the end-of-line bubbling handler on the document node itself, for cases where there are no other listeners.

We change `stopImmediatePropagation`'s prototype to raise a flag instead of stopping the event. To make sure we catch the flag before another handler runs, our modified `addEventListener` wraps all of the application's handlers to check for that flag after completion. Pando keeps a handle on the original application handler that was passed, because it is the one applications must provide to `removeEventListener` the same listener that was originally added. Our instrumented `removeEventListener` just maps the application's handler to the wrapper before doing the underlying call.

Pseudocode for our end-of-line bubbling handler is shown in Figure 2.

All of these modifications are transparent to the application. Event listeners cannot be introspected, so the application is unaware of the extra handlers we add. Additionally, any newly added properties are configured to be invisible to iteration, which addresses #3.

3.1.7 Visibility events (Vis)

As new long-running web applications emerge, users are necessarily browsing more sites *simultaneously*. A web user may leave one window open monitoring an email inbox or chatting with friends, and use another for active browsing. Looking only at browsing history and the times when sites were initially accessed, a user may be left wondering where time was really spent.

Browsers can keep an accurate record of which windows are active, which tabs are open, and where focus is. Fortunately, this information can be exposed to browser extensions. We observe that by recording the right amount of information, Pando can keep *accurate* accounting of time spent per-domain, per-site, and even per sections within a site.

Our recorder inserts entries into the log whenever tab focus changes. Identifiers for the source and destination window/tab are generated based on the tab and window IDs (which are exposed by the browser via the extension API) and included in these event records in order to permit accurate tracking of user activity in the browser. With this data collected, we build accurate time-tracking applications that give users better insight into their browsing habits.

3.1.8 Frame Hierarchies

Overriding prototypes requires that our recording instrumentations be injected in every document frame (<frame> or <iframe>). Each instrumented document works in isolation, and so the records they generate are relative to their containing document.

Without the frame's position in the document hierarchy, we cannot determine if events in a domain of interest are for a top-level document, or contained in another frame document. Annotating records with the hierarchical position lets us specify in queries the particular frames that are of interest, and allows for arbitrary nesting. For instance, this enables queries for all content loaded by applications on a Facebook user's profile page.

Fortunately, a document's hierarchical frame position can be obtained regardless of same-origin policies. Accesses to attributes of other frame objects (such as origin or document URL) or frame contents are subject to the same-origin policies, but the frame objects themselves can be compared by physical equality (i.e. ===), and the list of their child frames or parent frame can always be inspected. Pando obtains a frame's absolute position in the page by walking up until the top-level frame is obtained, and at each step remembering the frame's index in its parent's children list.

The mapping from frame indices to URLs is done later in the extension itself, where logs from all the documents are regrouped. Complications arise if the frames are reparented, but in those cases the parent relationships can be readjusted by making the recorders identify new and old ancestors.

3.2 Log

The log stores all of the events recorded by the user's devices, and therefore must be secure, durable, and location transparent. It must also be simple to deploy and maintain, and so should have few moving parts.

We defined two separate event log interfaces, one for recording and one for querying. We present the storage interface in this section, and the query interface in Subsection 3.2.4.

Records are transmitted from the browser extension, over a secure network connection to the event log web server. The server exposes an extremely simple REST interface to store new event records, which consists of a single `/insert/` collection handling POSTs. Request payloads consist simply of chunks of JSON-encoded records.

Client authentication to the storage interface can employ any scheme supported by browsers, such as username/password combinations, NTLM, or client SSL certificates. For server authenti-

cation, we are opting for a self-signed SSL certificate. The task of authorization, also performed by the web server, solely consists in determining if the client connecting should be allowed to write or not. No reads are ever allowed on the recorder interface.

3.2.1 Storage Format

Given that writes to the event log are append-only, with the occasional need for rotation, flat files are used as the default on-disk format. However, we expect Pando applications to build custom indices on top of these logs for speeding up their queries. For common access patterns, like searching within HTML for natural language text, Pando already offers indices. We have also built an additional storage backend on MongoDB, an open-source scalable high-performance document-oriented database, to temporarily manipulate subsets of interests with map/reduce programs. We believe these two formats alone can handle a large set of interesting queries.

3.2.2 Protecting the Data

Because the data recorded is private and potentially sensitive, it is of utmost importance to ensure that it remains protected. The records should be at the least encrypted on disk, using a key protected by a master password and loaded when the event log web server is started. Alternatively, logs can be placed on an encrypted partition, with keys managed externally.

As for the choice of the host running the server, we believe that users will have different preferences based on a combination of their levels of comfort and the degree of sensitivity of the data.

The simplest solution is to store the data on the same physical host as the browser. However, unless the user's browsing devices are connected, this solution implies that the logs gathered from multiple browsers are stored on different hosts. Users may also trust a host enough to browse from it, but not necessarily store logs, so this option may not suit all users.

Another choice is to place the logs on a home server, accessible from all of the user's browsers. This allows logs to be placed in a central location with plenty of storage space and processing power.

If users feel more comfortable securing their data with a third party, or do not have sufficient hardware to host the server themselves, it should also be possible to store event logs on cloud services such as Amazon's Simple Storage Service (S3), Google AppEngine's Datastore, or Microsoft's Azure.

3.2.3 Retention Policy

The event log web server can be configured with a retention policy to control the quantity of the logs that are kept. Logs can be rotated at the end of a configurable period. Depending on the Pando applications the user wishes to install, it may not be desirable to keep browsing data older than some given threshold. We are also planning on adding decay options based on the types of events recorded, so that for instance the list of clicks made have longer log-lifespans than the DOM modification events.

Analyses and indices constructed by applications are not automatically rotated, as they may contain statistics that roll-over, but users should always have the option to "reset" or uninstall applications.

3.2.4 Query Engine

In addition to the storage interface presented earlier, Pando also exposes a read-only query interface to the event log for the usage of the analysis tools. Access to the query interface should be controlled based on the the analysis tools that wish to access it such that only recognized tools, should be granted permission to connect to the query interface.

The query-engine is exposed as a web service running on a web server process separate from the storage web server. The query-engine web service has read-only access to the raw logs.

We plan on making the query-engine offer multiple modes of search: a structural search mode in which HTML/XML markup is discarded, a structural search mode in which it is possible to look for records affecting only a particular portion of the DOM or a a particular subtree pattern, similarly to XPath. It should also be possible to constrain queries on a particular context, such as the protocol used, domain names, or on sites that used a particular cookie.

These queries can also offer additional assistance when combined with any custom index that the analysis tool may have built. For instance it should be possible to form queries with temporal relations, or queries that depend on particular resource conditions such as "current number of pending `XmlHttpRequest`".

3.3 Application Proxy

On one hand, the applications must be easy to install and setup. Installing applications requires first downloading the application code itself, followed by possibly extra content (modules, data files, etc.) from third-parties. On the other hand, these applications require access to the logs to perform their analysis.

We protect the query API using a simple webproxy operating in two exclusive discrete modes: *locked* and *unlocked*. The proxy starts in unlocked mode. When unlocked, the proxy drops any incoming connection destined for the query API webserver, but all other connections are allowed. When locked, the proxy does the opposite, the only requests that are allowed to go through are those destined for the query API webserver. The proxy recognizes a special lock message , to transition permanently from unlocked to locked.

To install a new application, a new browser process is instantiated on the same host as the proxy server, configured with the proxy for sole internet access. The browser is pre-configured with a "bootstrap" extension, whose only task is to initiate the proxy's lock sequence. Because the proxy is initially unlocked, any website can be accessed, except Pando's query API.

Websites providing apps, can communicate a small message to the bootstrap extension to indicate that the application wishes to be locked to the user's logs. After confirming with the user, the bootstrap extension sends the lock message to the proxy, so that the application browser is permanently locked to the query API.

As long as browser proxy settings are unreachable from the JavaScript environments of the browser, and that extension runtime environments are protected from the page, it should be unfeasible for a web application running in the sandbox to either guess the lock message, or escape the proxy.

4. APPLICATIONS

Using the event log created by our recorder, we have developed three analysis tools that we believe are of high-utility for users. In all cases, the tools combine data coming from different online services, perform computations, and output an analysis report that can be easily displayed in a web page. It is excessively difficult to offer similar analyses as traditional online services without relying on users explicitly granting permissions to access the data required for analyses.

4.1 Personal Web Search

The first analysis consists in searching by search terms *all* of the content that has at one point or another been visible in the user's browser. Whereas online search engines help users find new pages

with high popularity, this tool helps with recalling content (not just pages) that contains the terms of interest. It works similarly to the built-in browser history, but can be configured to search over additional information, like form submissions, and XHR payloads.

History tools provided by browsers only give you a limited view of your past activities. They record an initial text snapshot of "nodes" in your history (i.e. accessible with back/forward buttons), but do not provide versioned histories of long-running application visual states. Nor is dynamic content searchable, even though it had been previously displayed by the browser.

Depending on the information available in the event log at the time of the search , snapshots of the DOM at the time of a match can be visualized in a suitable sub-frame, after scripts have been removed. This contrasts existing browser history tools by providing more visual information than simply a surrounding sentence. What we propose is of greater utility as it allows the users to search for parts of the page that surround the information they are really looking for.

4.2 Monetary Transaction Verifier

By correlating information from various sources, we can attempt to spot fraudulent transactions on credit card or bank statements. The task consists of finding events in the log that offer an explanation for entries in the bank statements, and flagging those entries that are not supported by the user's browsing history.

We assume that the financial institution provides a way to export a user's account statements in a way that can be parsed in order to extract tuples consisting of the name of the business and a transaction timestamp and amount.

The task of this tool is to find any context that could lead to explaining a transaction. Online transactions can often be tracked by either looking at mail or looking at visits to the relevant domains. Our intuition is that even purchases made offline can be explained through online behavior, such as chat messages, queries to map services, or geolocation items.

We built a prototype of this tool that scans the event log for both textual matches for the name of the business and the amount of the transaction, as well as broader cues (such as terms related to the business and URLs) that fall within a configurable temporal range during which the transaction occurred.

4.3 User Interests

By looking at events generated by Pando for webpage activity it is possible to analyze where a user spends her time online. We show in Figure 3 a breakdown of which sites get most of the attention of an author of this paper over a 12 hour work day. Activity times are summed across tabs, leading to very large durations for sites opened in multiple tabs. Additionally, the author spent a large portion of his time on non-browser activites which results in limited "focused" time in the browser.

5. EVALUATION

Pando's design includes several mechanisms that may impose a considerable overhead on the browsing experience. We evaluate the runtime overhead incurred by Pando, as well as the storage requirements for the event log. Our evaluation is performed on an Intel Core i5 2.80 GHz Quad-Core machine with 4 GB of RAM and an Intel 82578DC Gigabit Ethernet network interface. We used Google Chrome version 15.0.874.117 on Ubuntu 10.04.

5.1 Page Load Evaluation

Figure 4 shows the load time latencies for the homepage of 7 popular websites. We define the page load latency as the period

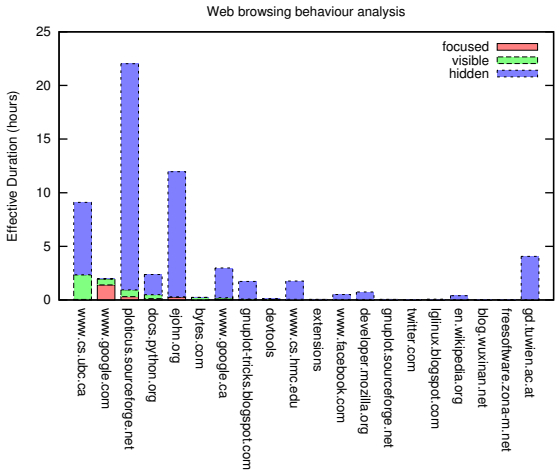

Figure 3: Breakdown of user activity on top 20 sites

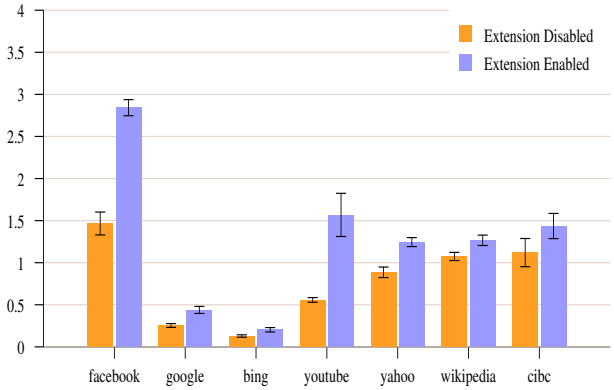

Figure 4: Pando Page load latencies comparation

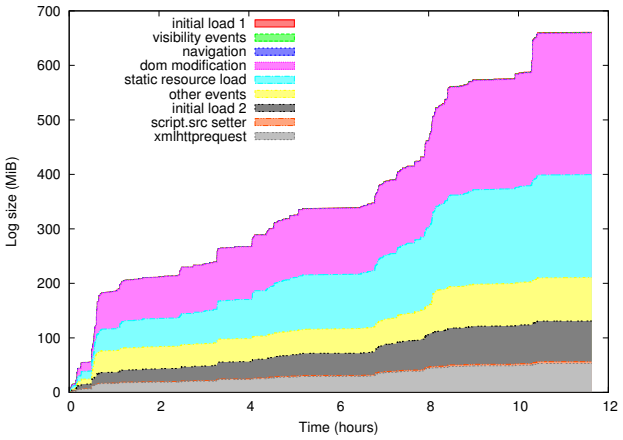

Figure 5: Pando log size vs. time

of time from then initial URL request issued from the browser, to the complete loading of all DOM events. We disabled all of the browser-side data cache in order to maintain consistency across these tests. We run each test 10 times, with and without Pando enabled, and average the results. We found that the overhead introduced by Pando is site-specific: the more DOM operations performed, HTTP requests issued and static content loaded by a site, the higher the overhead cost as Pando spends more time intercepting `XmlHttpRequest` and recording the DOM tree. While initial load times are an important metric to consider, a more comprehensive study is needed in order to gain further insight into how Pando's overhead affects the browsing experience.

5.2 Storage Overhead

We ran Pando continuously for 12 hours on the lab machine of a typical computer science graduate student. Figure 5 shows the breakdown of log events by type for this period, and identifies static content loading and DOM manipulations as the main sources of events. The non-deterministic events, including navigation and visibility changes, take a relatively small portions of the log. We compressed our raw log with *gzip* and achieved a 0.14 compression ratio (132 MB compressed from 932 MB of data).

6. RELATED WORK

Surprisingly little work has been done on recording user-centric information generated through browser-webpage interactions and using this information to protect users' privacy, improve web experience, and support various rich applications.

6.1 Record and Replay for Web Apps

Mugshot [9] is the first JavaScript record and replay tool that captures every nondeterministic event in an executing program, allowing developers to deterministically replay past executions of web applications. Because of its proxy-based nature, external content that bypasses the caching proxy will not be recorded at all. Moreover, Mugshot generates an extremely detailed log in a low-level language for debugging use, which makes it difficult to be interpreted by applications. Pando concentrates more on user-triggered browser-webpage interactions and logs events in a more semantically meaningful way. Other proxy based tools[2][3] indiscriminately track all events occurring inside the browser, and generate coarse-grained logs that requires extra effort to interpret. Even worse, the proxy based solutions are incompatible with the HTTPS protocol, and are therefore prevented from being widely deployed. WaRR [1] targets the interactions between users and web applications, but since its implementation requires modifications to the WebKit browser engine, it will likely be difficult to port to other browsers. In general, we don't believe that requiring browser modifications is appropriate for projects such as Pando as it raises additional issues of portability, security, and stability.

6.2 Content Script Injection Frameworks

Greasemonkey[4] is a Firefox extension that allows users to inject scripts into their browsers and manipulate the DOM "on the fly". Chickenfoot[5] puts an IDE in the browser's sidebar so that clients can write scripts for manipulating web pages and automate web browsing. DOM Snitch[6] uses similar functionality targeting web developers using the Chrome browser. C3 [8] abstracts a more general model from web browsers and provides a platform for web developers to write browser independent apps. Pando uses similar techniques to achieve a feature-rich, fine-grained event logging mechanism.

6.3 Applications supported by Pando

A diverse set of applications can benefit from the platform provided by Pando. Previous works [4][5][10][11] explore users' interests based on past queries, web history, and online social network information. They can therefore improve on the quality of search results generated by commodity search engines. All of them could provide local, personalized search by being ported on top of Pando's personal information repository. Pando could change the current data-collection model used by advertisers; It would be possible for Doubleclick, Smart AdServer, or Fastclick to delegate their user data collection undertakings to clients and thus mitigate the risk of Personally Identifiable Information (PII) getting leaked to third parties. A similar proof-of-concept application is Privad[7], a privacy perserving advertising tool which runs totally on the client's machine without leaking PII.

6.3.1 Time-tracking Applications

Time-tracking applications allow users to "take notes" of what happened in the past, and analyze them in the future. Browser based history and bookmarks are the simplest examples, but they only give users insight into the titles and urls of the previously visited webpages. EverNote[7] allows users to record their web documents on demand, but requiring users to actively "take notes" is ineffecient and error-prone. Time Doctor[8] and Rescure Time [9] can run silently in the background and keep track of usage among webpages. However, there are no security guarantees regarding these tools. Equivalents of these tools can be built on top of Pando, letting users "remember all the important things" while preserving privacy.

7. CONCLUSION AND FUTURE WORK

Today's internet forces users to make a compromising decision between privacy and functionality: They may elect to take advantage of useful online services but must entrust their data to others in order to do so. Pando represents an alternate perspective with regard to this tradeoff. We first observe that users can achieve long-term *durability* of their personal data, in the format that they

[4]http://www.greasespot.net/
[5]http://groups.csail.mit.edu/uid/chickenfoot/
[6]http://code.google.com/p/domsnitch/
[7]http://www.evernote.com/about/home.php
[8]http://www.timedoctor.com
[9]http://www.rescuetime.com

viewed it, by capturing a complete log of the browser's bahavior over time. Second, we observe that this secure and persistent log represents an alternate medium on which applications that have *privacy-preserving* access to personal data may be built. We present an isolated application runtime in which third party tools perform a binary transition, relinquishing external network access in exchange for access to a user's browser log. Using this environment, we implemented three useful applications that users would be unlikely to entrust to third parties.

Currently, we are in the process of doing a pando deployment over 20 users, and expect to gathering real-time data spans for a couple of months. In the future, we will actively work on enlarging the scale of our case study and doing a more profound analysis over the data set. We are also seeking to answer the questions like "Whether there are safe ways to disclose information or access remote state from Pando apps?" and "Whether we can allow Pando apps to run in a safe, cloud-based infrastructure to take advantage of scalable computing tools such as MapReduce and MPI[10]."

References

[1] ANDRICA, S., AND CANDEA, G. Warr: A tool for high-fidelity web application record and replay. In *41th IEEE/IFIP International Conference on Dependable System and Networks* (2011).

[2] ATTERER, R., AND SCHMIDT, A. Tracking the intereaction of users with ajax applications for usability testings. In *CHI* (2007).

[3] ATTERER, R., WNUK, M., AND SCHMIDT, A. Knowing the user's every move- user activity tracking for website usability evaluation and implicit interaction. In *World Wide Web* (2006).

[4] CHIRITA, P. A., FIRAN, C. S., AND NEJDL, W. Personalized query expansion for the web. In *ACM SiGIR* (2007).

[5] DOU, Z., SONG, R., AND WEN, J. R. A large-scale evaluation and anaylsis of personalized search strategies. In *World Wide Web* (2007).

[6] ERLINGSSON, U., LIVSHITS, B., AND XIE, Y. End-to-end Web Application Security. In *Proceedings of the 11th USENIX workshop on Hot topics in operating systems* (Berkeley, CA, USA, 2007), USENIX Association, pp. 18:1–18:6.

[7] GUHA, S., CHENG, B., AND FRANCIS, P. Privad: Practical pricacy in online advertising. In *USENIX Symposium on Networked Systems Design and Implementation* (2011).

[8] LERNER, B. S., BURG, B., VENTER, H., AND SCHULTE, W. C3: An experimental, extensible, reconfigurable platform for html-based applications. In *Usenix Conference on Web Application Development* (2011).

[9] MICKENS, J., ELSON, J., AND HOWELL, J. Mugshot: Deterministic Capture and Replay for JavaScript Applications. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation* (Berkeley, CA, USA, 2010), NSDI'10, USENIX Association, pp. 11–11.

[10] MISLOVE, A., P.GUMMADI, K., AND DRUSCHEL, P. Exploiting social networks for internet search. In *ACM Workshop on Hot Topics in Networks* (March 2006).

[10] Message Passing Interface, http://en.wikipedia.org/wiki/Message_Passing_Interface

[11] TEEVAN, J., DUMAIS, S. T., AND HORVITZ, E. Personalizing search via automated analysis of interests and activities. In *ACM SIGIR Conference* (2005).