Doppelganger: Better Browser Privacy Without the Bother

Umesh Shankar
ushankar@cs.berkeley.edu
UC Berkeley

Chris Karlof
ckarlof@cs.berkeley.edu
UC Berkeley

ABSTRACT

We introduce Doppelganger, a novel system for creating and enforcing fine-grained, privacy preserving browser cookie policies with low manual effort. Browser cookies pose privacy risks, since they can be used to track users' actions in detail, but some cookies also enable useful functionality, like personalization features. Web browsers currently lack an effective cookie management mechanism. Users must choose between two unpalatable options: a permissive, privacy-compromising policy for every site they visit, or a seemingly endless series of questions to which they must supply underinformed opinions. Doppelganger takes a big step forward: it makes automated determinations of cookies' value to enable a cost-benefit analysis, and offers an automated recovery system when that mechanism—or the user—makes an incorrect judgment. Doppelganger leverages client-side parallelism to automatically and simultaneously explore multiple cookie policies, enabling each user to create her ideal cookie policy. We tackle important and difficult subproblems along the way: mechanisms for recording and replaying web sessions; improved handling of third-party cookies; and enforcing fine-grained, per-site cookie mediation. We implemented Doppelganger as a Firefox extension; we discuss experimental results comparing it to various browser settings, as well as lessons learned from the real-world engineering challenges we faced in our implementation.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communication Applications – Information Browsers

General Terms

Security, Human Factors

Keywords

Cookies, Web Privacy, Usable Security

1. INTRODUCTION

1.1 Background

An *HTTP cookie* (from here on, simply "cookie") is a small data item sent by a web site to a web browser, then sent back to the originating site on subsequent requests. While the original intent of cookies was to provide a session state mechanism for the stateless HTTP protocol, cookies have since been used not just for things like shopping carts and authentication, but also for tracking users' web surfing habits and building targeted advertising profiles. The result is that site operators or third parties can gain undesirable insight into users' habits and browsing history. Cookies can identify a user at sites where she believes herself to be anonymous and track her actions across sites and browsing sessions. The problem is exacerbated if web sites can correlate the collected data to users' real-world identities.

The difficulty, therefore, is in deciding which cookies are worth accepting and which are not. Ideally, a user should be able to compare the privacy cost of a cookie with the functionality benefit the cookie enables. Most users are not equipped to make these decisions manually and accept the global defaults in their browsers, which often apply a single policy to all sites. These defaults tend to err on the side of functionality rather than privacy. Since web site features such as shopping carts and logins often require cookies and users may become confused or annoyed if these features don't work, the default policies liberally accept cookies. Although this approach will minimize users' frustration, it will also accept many useless tracking cookies which unnecessarily violate users' privacy. Our goal is to get the best of all worlds: a cookie policy that protects users' privacy while simultaneously retaining the desired functionality and, perhaps most importantly, not pestering users so much that they disable the system.

1.2 A solution: Doppelganger

We introduce Doppelganger, a web browser privacy tool to help each user formulate her ideal cookie policy. Doppelganger is a system that, in effect, simulates a world in which the user has accepted cookies and compares it to the (default) world in which the user has not. If there is no change in the user's experience between the two worlds, then we can fairly say that the cookies are not useful. Thus, Doppelganger essentially creates a hidden twin of the user who is constantly exploring the value of cookies on the sites the user browses and who informs the user when accepting cookies may be a good tradeoff; useless cookies are rejected by default, to preserve privacy. Another key component of Doppelganger is an automated error recovery module, which users may invoke with a single click. Error recovery attempts not only to correct the cookie

Goal	Mechanism
Automatically determine useful cookies	Mirror user session in hidden browser window (the *fork* window) that accepts additional cookies; look for differences in output (see Section 3.2)
Detect differences in pages	Compare page titles; look for user's name/ID in mirrored page; see if a click cannot be mirrored
Determine privacy implications of cookies	Parse and interpret site's P3P policy
Recover from errors	Enable additional cookies and replay user session, using information from the log (see Section 3.3)
Record user session, to enable error recovery	Central log of user's mouse clicks, form field values, and browser state changes (START and STOP events for each page load)

Figure 1: Summary of Doppelganger's cookie management mechanisms.

policy, but also takes action to restore the user's session to a good state, as though cookies had been accepted from the start.

We followed two main principles in designing Doppelganger. The first principle is that users don't like to be constantly interrupted with questions or alerts, and when this happens, they will tend to disable or ignore the offending mechanism [14, 26]. In particular, users should not be asked to do anything manually that can be done automatically. Doppelganger uses *client-side parallelism* to explore alternate policies in the background, trying to find those which result in a positive cost-benefit analysis between privacy loss and functionality gain. In a perfect world, Doppelganger would be able to automatically deduce the ideal cookie policy with no user interaction. In reality, some interaction is needed because users have different privacy preferences and valuations of web site features. Therefore, we assume it is reasonable to expect users to make a small number of high-level privacy decisions.

The second principle we followed in designing Doppelganger is that users don't care about cookies so much as *privacy* and *functionality*. Previous work suggests that: (1) the lack of privacy information in an easily-digestible form may be a significant obstacle to achieving good outcomes for users [1]; (2) users are sensitive to privacy protections, and are more willing to accept a privacy risk if data about them is protected [11]; and (3) users are willing to compromise some amount of privacy if they are offered meaningful incentives to do so [15]. Doppelganger seeks to address all these issues. Instead of requiring users to make uninformed low-level decisions about cookies directly, Doppelganger reformulates cookie decisions as cost-benefit analyses between privacy loss and functionality gains, which are presented to the user. This enables users to make informed decisions regarding their privacy and accept privacy loss when there is commensurate compensation.

1.3 Contributions:

- We introduce Doppelganger, a system for creating and enforcing fine-grained, privacy-preserving cookie policies with low manual effort.

- We show that Doppelganger improves the handling of third-party cookies in Firefox, especially with respect to redirection and inline frames.

- We show how to use client-side parallelism to explore multiple cookie policies simultaneously and find the right balance of privacy and functionality for each user.

- We leverage concepts from Recovery Oriented Computing [5] to implement an automated single-click recovery mechanism.

- We present empirically tuned algorithms for recording and replicating user actions.

- We evaluate the effectiveness of Doppelganger in establishing functional and privacy-preserving cookie policies for typical web browsing habits and compare the results against those obtained with available browser settings.

We summarize Doppelganger's cookie management mechanisms in Figure 1.

2. HTTP COOKIES

HTTP cookies are a general mechanism for web servers to store and retrieve persistent state on web clients [21]. Since HTTP is a stateless protocol, cookies enable web applications to store persistent state over multiple HTTP requests. For example, web shopping applications can use cookies to track which items a user adds to her shopping cart.

When a client makes an HTTP request to a server, the server has the option of including one or more `Set-Cookie` headers in its response. Clients will return these cookies in subsequent HTTP requests using the `Cookie` header. The `Set-Cookie` header has one required field, a name/value pair of the form $NAME = VALUE$. A web server uses this field to encode the state information it wishes to store on the client. There are also four optional fields: `expires=`$DATE$, `domain=`$DOMAIN$, `path=`$PATH$ $PREFIX$, and `secure`.

The `expires` field indicates how long the cookie is valid. After that date, the client's web browser should delete the cookie. If the `expires` field is omitted, then the cookie is called a *session cookie* and should be deleted when user closes the web browser. Cookies with an `expires` field are called *persistent cookies*.

The `domain` and `path` fields indicate for which HTTP requests clients should send back cookies. To determine which cookies to include with an HTTP request, the client searches its cookie jar for cookies for domains which suffix-match the domain of the request and paths which prefix-match the path of the request. For example, if the user requests the URL `http://online.foobar.com/store/index.html`, then a cookie with `domain=.foobar.com` and `path=/store` would be included with this request, but a cookie with `domain=pics.foobar.com` would not. The same-origin policy in web browsers prohibits one domain from setting cookies for another. The final optional field, `secure`, indicates whether the cookie should be only sent over encrypted HTTPS connections.

Cookies are also characterized by the context in which they are sent or received. Suppose a user clicks on a link for a particular document, and then the web browser issues a request for that

document. After the browser receives the HTML page from the web server, it parses the page for references to elements needed to render the page, and issues additional HTTP request for these elements. Examples of additional elements include images, Javascript files, stylesheets, Flash objects, and sub-documents. Some of these requests may be to the same domain of the requested document, but some requests may be to different domains. The latter is often the case with advertisements. Content whose URL matches the domain of the main page (i.e., the one in the URL bar) is considered to be *first-party*. All other elements are in *third-party* context. For example, if a user is visiting `www.x.com`, then content served from *.x.com is first-party, whereas content on the page from `www.y.com`, such as an ad, is third-party.

2.1 Uses of cookies

Cookies have many purposes: session state, personalization, authentication, and tracking. Web sites use cookies for personalization to remember users' preferences and settings. For example, Google allows users to customize the format of their search results and uses cookies to remember these preferences. Web sites with user accounts also use cookies to authenticate users' sessions [12]: after a user logs in, a web site can set an session cookie on the user's machine to authenticate her subsequent requests. Web sites can set persistent cookies to remember users and not require a login on subsequent visits. Lastly, web sites can use cookies to track users and their actions. For example, e-commerce sites can track customers' browsing history to make purchase suggestions, and advertising sites can track users to conduct targeted advertising. However, tracking cookies have troubling privacy implications. By tracking the pages a web surfer visits, the web searches she makes, and the items she browses and purchases, web site operators and Internet advertisers can construct sophisticated profiles of users for targeted advertising, data mining, and information sharing with other companies.

Tracking cookies also make cookie management difficult. Many users might prefer not to accept tracking cookies due to the privacy risks; recent studies [19] have found that about 58% of users have deleted their cookies at some point. To prevent her web surfing habits from being tracked, a privacy-conscious user might decide not to accept or send any cookies, but blocking all cookies causes a significant loss in functionality on the web. Most web mail services, e-commerce, and banking sites require users to accept and send cookies for authentication, and blocking cookies also denies users personalization features. Blocking all cookies is consequently impractical for most users.

2.2 Web browser cookie management

Rather than blocking all cookies, the average privacy-conscious user would probably be willing to accept some cookies from the web services she derives some benefit from, but would like to block cookies that compromise her privacy "too much" or provide her no value. Sadly, web browsers provide few useful options to users who wish to customize their cookie settings to this end. Users can configure their browsers to accept only first-party cookies, accept only session cookies, prompt for a decision, and combinations of the above policies.

These options are inadequate. Accepting only first-party cookies is a good start; most web sites do not require clients to accept third-party cookies to operate correctly and advertising companies such as DoubleClick use advertisements and web bugs [3] in conjunction with third-party tracking cookies to correlate users' web browsing across multiple sites. However, current web browsers' imple-

mentations of a "first-party only" policy fall short of expectations. For example, Firefox misclassifies IFRAME content as first-party, so advertisers embed ads in IFRAMEs [23] to trick browsers into accepting and sending their otherwise third-party cookies. Also, click-tracking services and advertisers use HTTP redirection [22] to evade third-party cookie blockers. Suppose `www.xyz.com` hires a click-tracking service `www.trackyou.com` to record statistics about its site usage. As a user navigates `www.xyz.com`, say by clicking on a link that seems to point to news articles on `www.xyz.com`, the target of the link may actually be something like `www.trackyou.com/redirect?target=www.xyz.com/news.html`. The user's request first visits `www.trackyou.com`, enabling `www.trackyou.com` to record the request and then redirect the browser to the real target, `www.xyz.com/news.html`. However, since the first request is for `www.trackyou.com`, a browser with a first-party only cookie policy will allow `www.trackyou.com` to set a cookie on the user's machine. The danger here is that if a third site `www.abc.com` and `www.xyz.com` both use the same click-tracker `www.trackyou.com`, this enables `www.abc.com` and `www.xyz.com` to collude with `www.trackyou.com` to determine their common users and track their browsing habits. Furthermore, if a user has an account on either `www.xyz.com` or `www.abc.com` that reveals her real name, this enables both sites to associate her browsing history with her real identity.

Accepting only session cookies also seems like a good idea, since it limits the ability of web sites to track users across browsing sessions. However, blocking all persistent cookies denies users the option of web site personalization and authentication without logging in or another more heavyweight solution. In addition, broadband connections and more effective computer power management make it convenient for users to leave their computers on and browsers open for longer time periods. We anticipate these factors will increase the length of users' average browsing session. A session cookie used over the course of a long browsing session (say, a week) could violate a user's privacy as much as a persistent cookie.

The only existing option for users who want a fine-grained cookie policy is for the web browser to prompt the user for every decision. With this policy, when the browser receives a cookie from a web site `foo.com`, it opens a dialog notifying the user it has received a cookie from `foo.com`, and asks the user whether it should accept the cookie, accept the cookie for each session only, or block it. The dialog also offers the option to apply the decision to every cookie from the same domain. Although in theory this mechanism enables the user to tailor her cookie policy at a fine level of granularity, the usability costs are severe [18]. First, despite the option for the browser to remember her decisions for each domain, a user will often receive a barrage of these interruptive dialogs in a browsing session. Second, although the dialog informs the user that a web site is trying to set a cookie, the user is given no information on how the cookie will be used by the web site and must often make policy decisions before she has even viewed the site's home page. If a user makes a mistake in her policy (e.g., deciding to block cookies at a site she later needs authenticator cookies to login), she must navigate several confusing browser menus (up to three levels deep) to correct her decision. Also, choosing which cookies to accept is non-obvious. She may know she needs to enable cookies for a particular domain to make it "work", but should she enable session cookies or persistent cookies? A user may discover she must enable cookies after she has already taken a series of actions on the web site. In the worst case, she must repeat all these actions after she

makes the necessary changes in the browser settings to correct her cookie policy.

2.3 "This site requires cookies"

Web sites do little to help with the cookie management problem. A web site can easily detect whether a particular user's browser will accept or deny cookies by using Javascript or a series of redirects. Many sites require cookies. If such a site detects the user is blocking cookies, it will inform the user that she must enable cookies to use the site and give the user instructions on how to enable cookies. The directions given by many web sites, however, instruct the user to enable cookies for all web sites, including third-party cookies. This sort of directive is easy for sites to issue, but can have big consequences for the hapless user's privacy not only at that site but every one the user visits. Naturally none of these negative effects are suffered by the site giving the instructions. Furthermore, few web sites give users information on how the site makes use of cookies. Without this information, users cannot easily decide whether they should accept cookies from the site.

2.4 Cookie management: The state of the art

Previous work does little to help users make informed decisions about cookie policies. Several Firefox extensions try to make the user interface for managing cookies less cumbersome. Cookie Button [8], Cookie Toggle [10], and Permit Cookies [20] add toolbars and enable keyboard shortcuts to help users quickly change cookie policies for the current domain. Add'n'Edit Cookies [2], Cookie Culler [9], and View Cookies [25] add shortcuts to easily view and delete cookies stored for a particular domain. Although these tools help alleviate the difficulty and annoyance of navigating the browser menus to change cookie policies and view previously set cookies, their focus is still on the low-level mechanism of cookie management, which few users understand and fewer still know how to manipulate. They do not help users decide the correct policy for a domain, nor do they cast the problem in more intuitive terms. A much more promising system is Acumen [13], which works on social recommendations for accepting cookies; users are notified how many other users accept the cookies in question. This system does not protect users' privacy itself, though, as it does central data collection of users' choices. It also does not take into account users' inability to make good choices without information. Such a system, with appropriate anonymization, is complementary to ours and could serve as another line of defense before users are burdened.

The Platform for Privacy Preferences (P3P) Project [24] is a protocol developed by the World Wide Web Consortium to help inform users of the privacy guarantees of the web sites they visit. P3P envisions users configuring their web browsers with specifications of their privacy requirements while surfing the web. Then, when a user visits a web site, that site will send a compact P3P policy specifying how it uses personal information, and the browser will determine whether the user's and site's policies are compatible. If not, the browser would inform the user of the incompatibility. P3P seems useful for helping users make informed decisions about their cookies policies, but in practice P3P has many problems [7]. Companies have been reluctant to adopt its complicated protocol structure, policy configuration is cumbersome for users, and the barrage of privacy warnings and notifications while web browsing becomes burdensome and confusing. Recently, though, there are more tools for writing and understanding P3P policies [4, 6, 16], and we hope that either P3P or some other privacy standard emerges to help us accurately gauge privacy risks.

Felten et al. have explored techniques to increase users' peripheral awareness of cookies and improve their ability to make informed decisions about cookie policies [18]. Their Cookie Watcher tool notifies users of cookie events and gives some limited information on the risks of accepting cookies. For example, it notifies users that a third-party persistent cookie could be used to track users across sites and web browsing sessions. Although Cookie Watcher may help users understand the risks of accepting cookies from a web site, it does little to help users evaluate the benefits of accepting a cookie. Likewise, Bugnosis [3] alerts users to the presence of "web bugs"—invisible images used for tracking, sometimes via cookies—but does nothing to mitigate their effect.

3. HOW DOPPELGANGER WORKS

If a user wants to decide whether or not a particular cookie is beneficial, she must determine whether the benefit she receives from accepting the cookie outweighs the attendant privacy loss she suffers. Thus, her ideal cookie policy is one that accepts only those cookies for which the cost-benefit analysis yields a positive result. Although each user values privacy risks and functionality gains differently, we want to avoid interruption when the answer is clear.

To this end, we developed Doppelganger, a web browser privacy tool to help each user perform this cost/benefit analysis and formulate her ideal cookie policy. Doppelganger's main goal is to identify useful cookies and their privacy implications automatically. Doppelganger relies on the following principle to identify useful cookies: if a cookie from a domain confers some benefit, it should be evident in the user's experience. If no such benefit is found, then we may assume that cookies from that site may be blocked.

Doppelganger uses two main techniques to identify cookies beneficial to the browsing experience: mirroring and user initiated error recovery. Network bandwidth and CPU power have been increasing rapidly, and web browsing clients often have excess bandwidth and CPU available. We leverage that spare bandwidth and computing power to take a "partial derivative" with respect to the cookies whose benefit we are trying to measure. When Doppelganger encounters a domain in the user's browsing session for which it hasn't determined a cookie policy, it mirrors the user's web session in a hidden parallel session whose only difference is the cookies accepted and sent. We refer to this hidden parallel session as the *fork window* since it represents a forking of the browser state. Correspondingly, we refer to the cookies speculatively used by the fork window as *fork cookies*. We show an overview of Doppelganger's architecture in Figure 2.

When Doppelganger detects a difference between the main window and fork window, it reveals the fork window and asks the user to compare the two. The benefit of the fork cookies is any advantageous service present in the fork window which is not in the user's main browsing window. To evaluate the cost of these fork cookies, Doppelganger provides the user a condensation of the domain's P3P policy (if available) and a description of the kind of tracking enabled by the cookie. Doppelganger records the result and automatically uses it for future cookie policy decisions for that domain.

The second technique Doppelganger uses to identify beneficial cookies is user initiated error recovery. The user interface for this error recovery is a single button labeled Fix Me on the browser status bar. Fix Me is a rewind-and-playback mechanism. Doppelganger maintains a log of a user's actions and browser state changes, and invokes the Fix Me mechanism when the user indicates to the system that something is wrong, perhaps due to an error message or missing functionality which the mirroring system missed. The idea is that if a lack of cookies was the problem, then
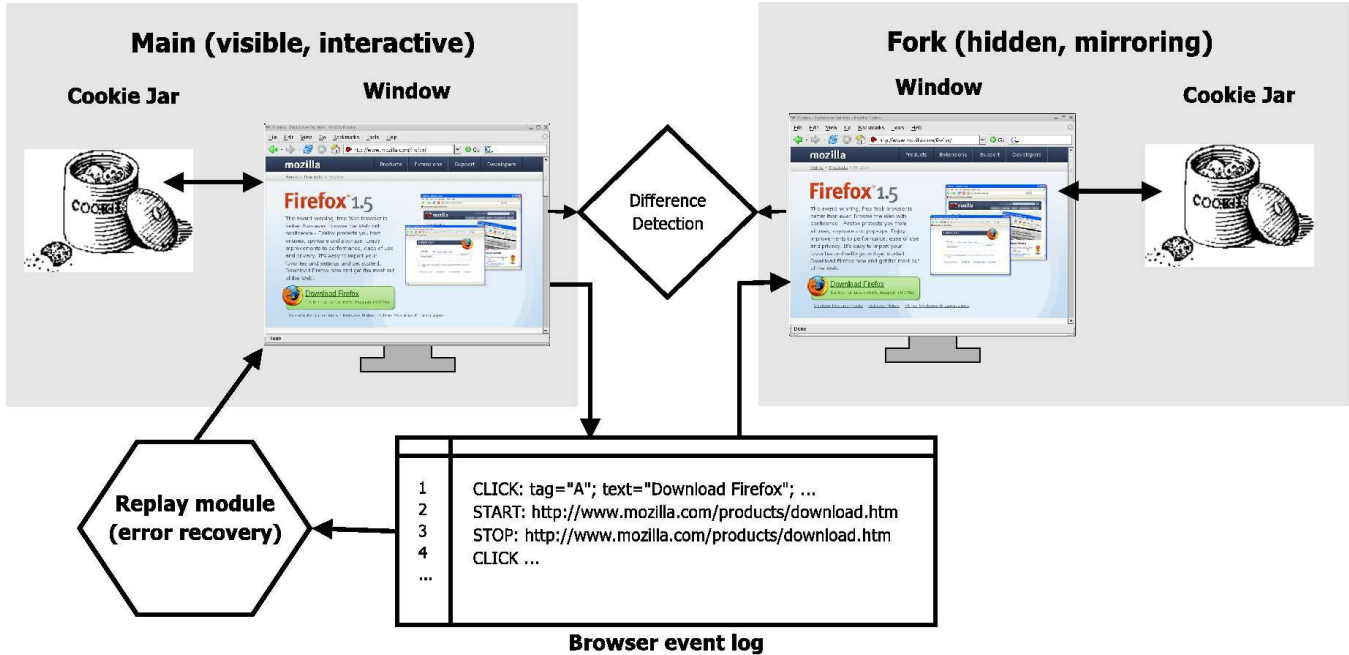
Figure 2: An overview of Doppelganger. Doppelganger mirrors the user's web session in a hidden fork session whose only configuration difference is the cookies accepted and sent. When Doppelganger detects a difference between the contents of the main window and fork window, it reveals the fork window and asks the user to compare the two (see Figure 5). Doppelganger also maintains a log of the user's actions for error recovery.

we may enable cookies and replay the user's actions, simulating what the user's session *would* have been if cookies had been enabled in the first place enabled.

Doppelganger can operate in three different configuration modes: *high paranoia*, *medium paranoia*, and *low paranoia*. These modes differ primarily in how Doppelganger handles session cookies. The privacy loss of most session cookies is relatively small, but some uses of session cookies pose higher privacy risks (Section 2.2). In low paranoia mode, Doppelganger accepts all first-party session cookies for all domains, and in medium and high paranoia modes, Doppelganger determines a per-domain policy for session cookies. In all modes, Doppelganger determines a per-domain policy for persistent cookies. We discuss these modes and their usability tradeoffs in more detail in Section 3.4.

We implemented Doppelganger primarily as a Firefox extension in about 6000 lines of Javascript code. We also made a small (30 line) change to the main C++ source code, which we have submitted for inclusion into the mainline. Since Doppelganger is implemented in Javascript, it is portable to any operating system on which Firefox runs. The primary user interface is limited to the Fix Me button on the browser taskbar. For debugging purposes, we appended a tab to the LiveHTTPHeaders extension [17], used for watching HTTP request and response traffic, that enabled us to monitor and configure our system.

3.1 An example

Before discussing Doppelganger in detail, we first present a more elaborate example of Doppelganger in operation where a user interacts with a fictitious web site www.xyz.com. To illustrate all of Doppelganger's features, we assume Doppelganger has been con-

figured in high paranoia mode, the most conservative configuration. At the end of the example, Doppelganger will have determined a complete cookie policy for www.xyz.com.

Suppose a user visits an e-commerce site, www.xyz.com, for the first time. The default policy for the main user window in Doppelganger is to block all cookies. At the same time, the hidden fork window will also visit www.xyz.com, but will accept (and send back) first-party cookies from the site, with the aim of deciding whether first-party session cookies from www.xyz.com are beneficial. For the next few page loads on www.xyz.com (details, Section 3.2), Doppelganger mirrors each user action and form field value in the fork window (Section 3.5); after each page load, Doppelganger compares the resulting main and fork windows for differences, and alerts the user if it judges them significant.

Suppose the user adds an item to her shopping cart, an action which requires cookies to be enabled. The fork window will contain the shopping cart page, but the main window will remain on the item page (a common failure mode). Doppelganger will detect the two pages as different, triggering a user-choice dialog.[1] Doppelganger displays the primary window and fork window side-by-side. A dialog box will give an estimation of the privacy risk from switching to the fork window (the cost) and the user can see what additional features are offered on the fork side (the benefit; in this case a functioning shopping cart). The user can then choose one of three options: switch to the fork side and accept the cookies, stay with the main browser and reject the cookies, or defer judgment and continue mirroring. Let us assume the user chooses to switch to the fork window, accepting the cookies. Since the user indicated that first party session cookies provided some benefit, Dop-

---

[1]Low and medium paranoia modes eliminate this dialog box.

(a) In the first session, the user must accept session cookies to add an item to her cart, but Doppelganger disables them by default. The mirroring process automatically detects this and offers the user the choice to switch to the fork browser, which shows the desired shopping cart.

(b) In the second session, Doppelganger accepts session cookies in the main window per the earlier decision. The fork window also sends back persistent cookies to test their benefit. Doppelganger finds no difference between the fork and main windows, so it assumes persistent cookies are unnecessary and appends a rule to block persistent cookies to the policy for `www.xyz.com`.

Figure 3: Graphical representation of mirroring for the example in Section 3.1.

pelganger records the decision to accept first-party session cookies at `www.xyz.com` for future sessions.

While the user has indicated that first-party session cookies from `www.xyz.com` have benefits, Doppelganger still hasn't determined whether first-party persistent cookies from `www.xyz.com` offer any benefits. However, Doppelganger will store first-party persistent cookies from `www.xyz.com` from this session in a separate fork cookie space for additional investigation during the next session. Now, suppose the user ultimately decides not to purchase the item yet, and closes her browser.

During her next session, she navigates to `www.xyz.com` again. The browser has deleted the `www.xyz.com` session cookies from the previous session, and Doppelganger is keeping the `www.xyz.com` persistent cookies aside in the fork window's cookie space. To determine if those persistent cookies have value, Doppelganger repeats the mirroring process again. This time, the main window accepts session cookies as per the earlier decision, but the fork window not only accepts new session cookies but also sends the persistent cookies it received before. For our example, let us say that the persistent cookies do not enable any additional features. After the user has visited a few pages on the site, if Doppelganger does not detect significant differences between the fork window and the main browsing window, it will decide that persistent cookies are not necessary. Doppelganger will record the decision automatically and stop the mirroring without any user interaction.

The persistent cookies in the fork window will, however, be retained for future error recovery if the user later finds that some desired feature does not work. Suppose that the user had entered some personalization features in her first browsing session at `www.xyz.com` which affect the browsing experience in relatively subtle ways that Doppelganger missed. The user may notice this problem after Doppelganger already made an automatic policy decision to reject persistent cookies for `www.xyz.com`. Doppelganger provides the Fix Me button on the status bar of the main browsing window to recover from these errors. When the user presses Fix Me while browsing `www.xyz.com`, Doppelganger rewinds the user's browsing session on `www.xyz.com`, enables the next most permissive cookie acceptance policy (in this case, accepting first-party persistent cookies), and automatically replays the user's session at
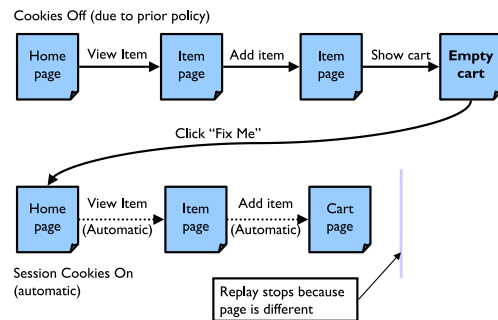

Figure 4: An example use of Doppelganger's error recovery mechanism. Suppose either Doppelganger or the user made a decision to not accept cookies at a particular site, but it turns out cookies are needed to maintain a shopping cart. If needed, the user can indicate that cookies may be needed by clicking the Fix Me button; Doppelganger rewinds to the start of the session at the site, enables cookies, and replays all the user's actions without any further user intervention.

`www.xyz.com`. The user gets the same final page as when she pressed Fix Me, but now with any additional benefits of sending the `www.xyz.com` persistent cookies received in the first browsing session. We discuss Doppelganger's error recovery mechanism further in Section 3.3.

3.2 Mirroring

In this section we discuss Doppelganger's mirroring system in more detail. During a user's browsing session, Doppelganger observes all page loads in the main window. When it encounters a page load for a domain for which it has doesn't have a complete policy, it begins to mirror the session in the fork window. Doppelganger mirrors the session by replicating the user's main window actions in the fork window and then looking for differences between the two. Mirroring user events is non-trivial; we discuss it in depth in Section 3.5. In the rest of this section, we will describe

(a) Main window   (b) Comparison dialog   (c) Fork (mirroring) window
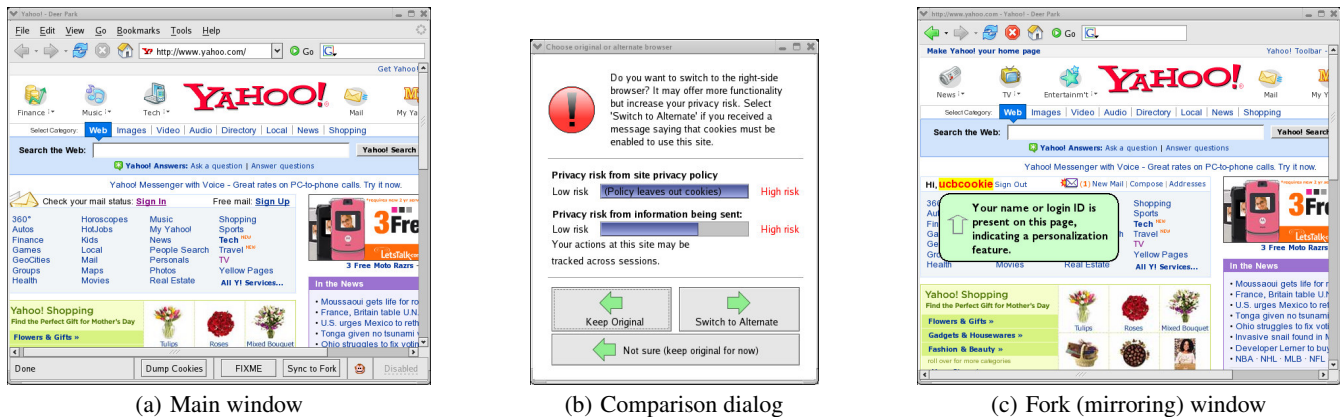
Figure 5: A screenshot of the Doppelganger's comparison dialog. When Doppelganger detects a significant difference between the main and fork windows, it prompts the user for a decision. Doppelganger provides some indication of the difference and a measure of the privacy risk from accepting cookies. In this case, Doppelganger detects the presence of a personalization feature and alerts the user to it.

how Doppelganger formulates cookies policies and how it maintains two separate cookies spaces for the fork and main windows. We then show how Doppelganger uses the fork window to make automatic decisions affecting the cookie policy and expose additional functionality enabled by cookies to the user.

3.2.1 Fork window cookie policies and cookie name spaces

Doppelganger formulates cookie policies based on tail domains. Tail domains are the last two components in the host name of URLs (e.g., yahoo.com).[2] Doppelganger applies the same cookie policy to all cookies and pages matching the tail domain.

Doppelganger enforces one of five possible first-party cookie policies for each tail domain D:

Policy	Session Cookies	Persistent Cookies
$P_{?,?}$	?	?
$P_{S,?}$	accept	?
$P_{S,P}$	accept	accept
$P_{S,X}$	accept	downgrade
$P_{X,X}$	deny	deny

In the table, "?" means that Doppelganger does not yet know the correct policy for D, and "downgrade" means that persistent cookies are converted to session cookies.

Doppelganger currently blocks all third-party cookies because we have not encountered any sites where they provide any benefit, although it is capable of enforcing third-party cookie policies on a per-site basis. If we discover some sites for which third-party cookies prove beneficial, we can easily enable that feature.[3] Note that since Doppelganger enforces per-site policies, enabling third-party cookies for one site would only allow tracking with other sites

---

[2]There is much debate over the right way to decide how many trailing domain name components to use; presently we use a simple heuristic, as do most browsers, to use an additional component for international TLDs (two letter suffixes).

[3]Since a site's context can be out of its control, such as being included in a frame by another site, an intended first-party cookie can become a third-party cookie. It is uncommon for external documents that are not advertisements to be put in frames, and many sites do not work properly when framed in any case.

that also had third-party cookies enabled. This is in sharp contrast to browsers' settings, which have only global policies for third-party cookies.

There is another problem related to third-party cookies: the mechanism Firefox uses to identify third-party cookies is vulnerable to IFRAME [23] and redirection [22] tricks. IFRAMEs are entire documents embedded in HTML pages, and for various reasons, Firefox incorrectly determines the context of HTTP requests generated by IFRAMEs. For the purposes of cookie management, Firefox classifies an IFRAME request as an independent request for a top-level HTML page rather than as a request for an element of a larger page. Therefore, Firefox classifies cookies for IFRAME requests as first-party instead of third-party with respect to the enclosing page. Doppelganger addresses the IFRAME problem by more reliably determining the context of an HTTP request by matching its tail domain against that of the topmost page's URL. We discuss the countermeasure to redirection tricks in Section 3.2.5.

In order to send different sets of cookies to the main window and the fork window, Doppelganger partitions the cookie space to allow multiple copies of a cookie with a specific NAME=VALUE pair and impose different access controls on them. Doppelganger achieves this by implementing nsICookieConsent, an optional Firefox interface designed for deferring cookie policy decisions to an external module. Its original intention was for use with P3P, but that functionality has since been disabled. nsICookieConsent was designed to be applied to classes of cookies at once, since the browser's controls do not have provisions for individual cookies. However, Doppelganger must impose differential access controls according to the particular cookie being set and which window (fork or main) is using it. Addressing this problem required a small change to the interface and a small (30 line) change to the main C++ source code to use this new interface.

3.2.2 Mirroring in the fork window

When the user visits a domain D, Doppelganger checks its cookie policy for D. If it has a complete policy (i.e., $P_{S,P}$, $P_{S,X}$, or $P_{X,X}$), Doppelganger does no mirroring and simply applies the policy to the main window. Suppose Doppelganger has no policy for D (e.g., it is the user's first visit to the domain). Then the fork window starts sending and receiving first-party cookies for
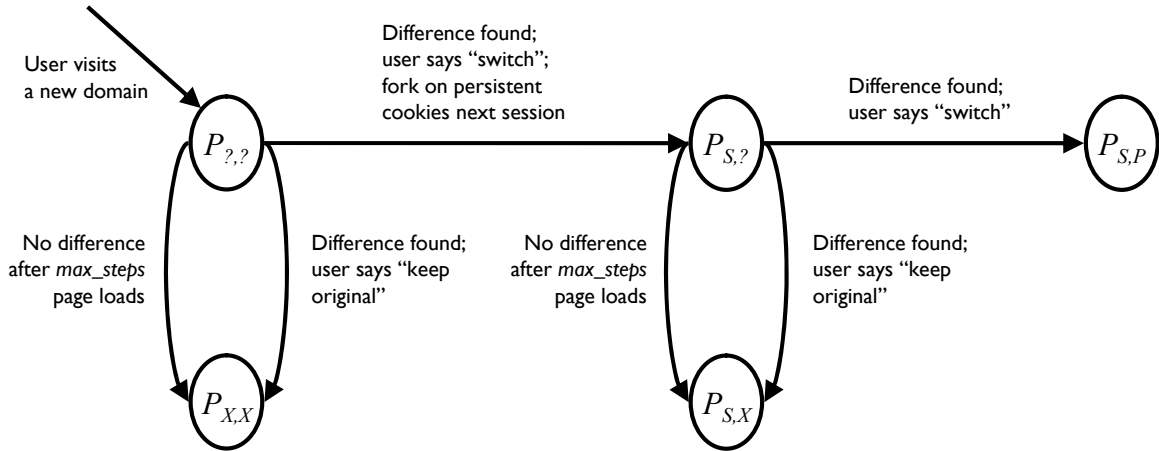
Figure 6: How Doppelganger determines a cookie policy for a domain during the mirroring process. When Doppelganger detects a difference between the main and fork windows, it prompts the user to decide whether the additional features are worth the potential privacy risk. Doppelganger makes an automatic policy decision if it does not detect any differences after *max_steps* page loads. We omit some additional transitions present in low and medium paranoia modes (see Section 3.4).

that domain. Doppelganger mirrors the user's actions for a constant number (*max_steps*) of page loads on D and monitors the fork window for differences. If it detects no difference after *max_steps* page loads, Doppelganger concludes that cookies at D provide no benefit, stops mirroring, and sets the cookie policy for D to deny all cookies ($P_{X,X}$).

Alternatively, if Doppelganger detects a difference, it prompts the user to decide whether the additional features are worth the privacy risk by attempting to highlight benefits and display privacy risks. For an example comparison screenshot, see Figure 5. If the user answers "keep original", Doppelganger stops mirroring and sets the cookie policy for D to $P_{X,X}$. If "switch to alternate", it stops mirroring and sets the cookie policy for D to $P_{S,?}$. Recall that $P_{S,?}$ accepts session cookies, but has an undetermined policy for persistent cookies. Doppelganger then transfers the state of the fork window to the main window to automatically provide the user the benefit of the cookies. For the remainder of the session, Doppelganger accepts all first-party cookies from D.

Now, suppose the user closes her browser, restarts it the next day, and revisits D. The policy for D is now $P_{S,?}$ and the browser may have persistent cookies from D from the previous session. Since Doppelganger has not yet determined whether persistent cookies from D are beneficial, it begins to "fork" on these cookies. Doppelganger loads persistent cookies for D from the previous session into the fork cookie space and clears all of D's cookies from the main cookie space. Doppelganger then proceeds as it was when forking on session cookies, except now, both windows accept session cookies instead of just the fork window.

The difference is the fork window may have persistent state from the previous session which positively affects the user's experience. Doppelganger tries to detect this. Again, Doppelganger mirrors the user's actions for a constant (*max_steps*) number of page loads on D and monitors the fork window for differences. If it detects no difference after *max_steps* page loads, Doppelganger concludes persistent cookies at D provide no benefit, stops mirroring, and sets the cookie policy for D to block persistent cookies ($P_{S,X}$). Otherwise, if it detects a difference, it prompts the user for a decision whether the difference is beneficial. If the user answers "no", Doppelganger

stops mirroring and sets the cookie policy for D to $P_{S,X}$. If "yes", it stops mirroring, sets the cookie policy for D to accept persistent cookies ($P_{S,P}$), and transfers the state of the fork window to the main window. We summarize the mirroring process in Figure 6.

Presently, *max_steps* is a constant. We want it to be small enough that we do not end up effectively accepting more cookies via the fork window, but large enough to see differences due to cookies. Large-scale trials are needed to determine a good value; in testing we set *max_steps* to 5.

3.2.3 Difference detection

Doppelganger must be able to detect when the fork and main windows significantly differ in function or personalization enough to warrant interrupting the user for a decision. Doppelganger should ignore things like advertisements, randomized placement of news items, or other sources of natural nondeterminism. In our difference detection algorithm, we must address a tradeoff: if Doppelganger reports too many page pairs are different, the user will be asked to make too many decisions, whereas if the system fails to detect meaningful differences, cookies will be rejected too aggressively and the user must detect problems manually and initiate error recovery (Section 3.3). In both cases the user is needlessly inconvenienced. At present we use a coarse mechanism: we compare page titles (to detect obvious errors) and we look for the presence of the user's name or login ID in the fork window (and its absence in the main window) to detect personalization. In addition, if a user action cannot be replicated in the fork window, we assume the pages are different. A better heuristic is the source of ongoing work. We do not consider an error in loading a page as a significant difference; instead the mirroring process re-starts at the next page after re-syncing the fork window to the main one.

3.2.4 Exposing the cost of cookies

Even beneficial cookies carry privacy risks. When Doppelganger detects a potential benefit of accepting cookies at a domain D, it tries to measure and expose the privacy risks when it prompts the user to compare the fork and main windows. One measure of the risk is the type of cookies Doppelganger must enable for the user to

benefit (i.e, session or persistent). We also assess risk by interpreting the domain's P3P policy, if one exists; we borrowed some P3P parsing code from [4] for this purpose. Doppelganger represents the privacy risk with two bars, one derived from the site's privacy policy, and one representing the risk from the type of cookie allowed. For an example of Doppelganger's risk assessment during a comparison, see Figure 5.

3.2.5 Addressing ephemeral site visits

Doppelganger uses a slightly different strategy to address domains which may be visited often but never for very long. This situation arises in several situations: click-tracking and advertisement redirect tricks [22] (discussed in Section 2.2), certain web portals, and search engines. Web portals and search engines contain links to other domains that are the user's ultimate goal; in the meantime, though, the portals use cookies to track the user's actions. Also, shopping search portals use redirects through advertising trackers (e.g., DoubleClick and Dealtime) which set persistent cookies to track the offsite links that users follow. All these cookies appear to the browser as, technically, first-party cookies, but we want to block most of them since they confer no benefit.

The risk is that Doppelganger will perpetually mirror visits to these sites. Since users will likely never have *max_steps* consecutive page loads on these domains, Doppelganger will never arrive at a policy decision for them. Doppelganger would therefore invoke the mirroring process on every visit to these domains to try to determine their cookies' value, in effect enabling cookies forever. To address this problem, we maintain a lifetime hit count for domains with an undetermined cookie policy and set the domain's cookie policy to $P_{X,X}$ when the hit count exceeds a constant, *max_visits*. We are still determining an optimal value for this constant; in testing, we set *max_visits* to 8. The end result is that a policy decision is made for every site after a finite amount of time.

3.2.6 Logins

Doppelganger optimizes cookie management for sites where a user logs in. When Doppelganger detects a user logging into a domain, it automatically enables session cookies for that domain. The rationale for this policy is that if a user has a relationship with a site which requires a login, then accepting session cookies is unlikely to cause additional privacy loss, and we want to avoid unnecessary user interruptions. Doppelganger detects logins by looking for form submissions containing username and password fields.

3.3 User initiated error recovery

The second major component of Doppelganger is user-initiated automated error recovery. The first line of defense is the mirroring mechanism described above, but Doppelganger's comparison function may be imprecise, mirroring may end prematurely, or the user may change her mind regarding the cost/benefit of cookies from a domain. Doppelganger invokes the error recovery mechanism when the user notices some feature is not working properly, or when she sees a cookie-related error message Doppelganger did not automatically detect. The user interface is simple: Doppelganger installs a single button labeled Fix Me on the browser status bar that the user can click when necessary. Our techniques for error recovery borrow ideas from Recovery-Oriented computing; in particular, we use the "Three R" model of recovery introduced by Brown et. al [5]: Rewind, Repair, Replay.

Doppelganger handles recovery differently depending whether it is mirroring a session or not. If Doppelganger is mirroring a session, it simply uses the mirroring comparison dialog to show the user what recovery would look like. If Doppelganger is not currently mirroring a session, it must achieve the same effect. To do this, Doppelganger enables the next most permissive cookie policy setting (as the fork window would have) and replays the user's session at the current site from the beginning by replaying all user-initiated UI events (e.g., clicks, form submissions). We do not replay across site boundaries.

Of course, strict replaying is not the goal: we want the result to be different (and better). Doppelganger manages the replay with a state machine which watches page loads and sends user events. If Doppelganger cannot replay a user event, an expected page does not load, or an unexpected page loads, Doppelganger stops the replay. Since one of these events is evidence of a page not present in the original sequence, Doppelganger optimistically assumes the problem is fixed; since the desired outcome is one that we have not yet seen, there is no way to know if it is the correct one automatically. If the problem has in fact not been fixed, the user may click the button again, and Doppelganger will enable the next most permissive cookie setting (if possible) and replay again. If this, too, fails, then likely a lack of cookies was not the source of the problem.

There are two problematic cases for replaying. The first is the nonlinearity of many sessions: what if the user had hit the Back or Forward buttons during the original session? Our current approach is to replay those buttons (but not Reloads) during the replay as well; this seems to work in practice. Another case is that of HTTP POST requests. According to the specification, GET requests, the most common kind, are to be used for idempotent requests, and POST requests for non-idempotent ones like transactions. Although we believe the danger is low—after all, if the transaction completed, why would the user be invoking the replay?—we do not replay through POSTs. Some sites abuse the POST request for idempotent actions, which would block the replay. This misuse is bad policy, since it makes it difficult for users to go back and forward, reduces the effectiveness of proxy servers, and reduces the effectiveness of our replay system while making their users do more work to accept cookies. Others misuse GET requests for non-idempotent actions, which is very dangerous since the back and forward buttons can easily and inadvertently trigger the action again; proxy caching could also break. In short, there are many existing reasons for sites to use POST and GET requests appropriately, and if a site does misuse POST or GET, problems will probably surface regardless of Doppelganger.

3.4 Higher-privacy modes

Always-on Internet connections and more effective power management have conspired to make very long sessions not only possible but easy. Accordingly, we have implemented multiple modes of operation for Doppelganger, characterized by "paranoia level", which have different session cookie policies. *Low paranoia* mode always accepts session cookies, and is thus the least intrusive, least private mode. *High paranoia* never accepts session cookies by default, using the same mirroring-and-recovery tandem on session cookies as on persistent cookies. It is the most privacy-preserving, but most intrusive mode; remember, though, that comparisons only must be made when the mirroring process detects a difference.

As a compromise between the two, *medium paranoia* mode uses mirroring, but when a difference is detected, automatically enables session cookies without asking the user. Since the privacy risks of session cookies are generally low, the net benefit of accepting them is likely positive at a domain where the mirroring process detects a benefit, and we can avoid interrupting the user to make a decision. In addition, medium paranoia mode enables session cookies when

Mode	Session Cookie Policy	Persistent Cookie Policy	Notes
Low paranoia	Accept all	Per-domain	Requires least user interaction
Medium paranoia	Per-domain (never ask user)	Per-domain	Automatically enables session cookies on POST or when a difference is detected during mirroring
High paranoia	Per-domain	Per-domain	Highest privacy; requires the most user interaction

Figure 7: Summary of Doppelganger's different privacy modes.

a `POST` request is seen. A main benefit of medium is that it automatically denies cookies from tracking sites which are visited using redirection, but never requires users to make left-or-right comparisons for session cookies. However, if the mirroring process fails to detect a useful difference, the user may need to use the Fix Me button. We summarize Doppelganger's different privacy modes in Figure 7.

3.5 Replicating individual user actions

In this section we show how Doppelganger replicates the two dominant types of user interactions in web browsers: mouse clicks and form submissions. Doppelganger replicates user actions both during mirroring and during error recovery. In the former case, we replicate user actions in the fork window immediately as they occur; in the latter case, we replay a series of actions from the log into the main window. In both cases, the proximate mechanism of replication is the same, and we describe the algorithms in this section. By replaying at the level of user actions rather than page loads, relevant Javascript code on the page is triggered automatically.

3.5.1 Mouse clicks

Replicating clicks turns out to be difficult in practice for two main reasons: document elements do not have unique IDs, and there is a significant amount of nondeterminism in what results are returned for a given URL. This latter problem can arise from naturally changing pages, e.g., news sites and search engines, but this problem also surfaces in advertisements and stochastic link rewriting for click tracking. The basic click replication mechanism involves three steps: (1) record information about the click; (2) try to find the matching target in the fork browser; and (3) send the appropriate click event to the target.

3.5.1.1 Recording the click.

Our goal in recording clicks is to capture enough information about the click that we can replicate it, and to record it in a way that tolerates small changes to the document in which it is being replayed. Our initial algorithm constructed a path in the DOM tree from the document root to the clicked element and tried to reconstruct this path in the DOM tree of fork window document. This approach failed because it was too precise; it could not adapt to small perturbations in the document. In the end, we used a heuristic refined through experimentation.

First, when the user clicks an element, we record some identifying information about the event:

- The URL of the page in which the click occurred
- If the click was in a frame, the topmost document's URL
- The HTML tag name of the target of the click (e.g., "DIV")
- The mouse coordinates of the click (for imagemaps)
- The element's attributes (e.g, "href", "id", "name")
- The text content within the element

- If it is a form element, information about the enclosing form
- Which mouse button the user pressed

To be more precise, we do not always record the immediate target of the element; there may be many HTML fragments of the form

```
<B>click here</B>
```

for example, and the `<B>` tag is not interesting for a click perspective. Instead, we start backtracking to the root of the document to find the nearest ancestor of the target element which initiates some action. For example, if the HTML code reads

```
<A href='/x'>To get $$, <B>click here</B></A>
```

we would record the click on the `<A>` tag, not the `<B>` tag. In general, we stop at elements which have an `href` attribute, an `onclick` attribute, or are input elements of a form. This reduces ambiguity considerably when trying to replicate the click.

3.5.1.2 Finding a match in the target window.

When Doppelganger replicates a click on an element in the source window, a primary challenge is locating the analogous element in the target window. If the web were static, each request for a URL would yield the same response and the task would be straightforward. Instead, there is a fair bit of nondeterminism in the responses. For example, on news sites, a new article may change the locations of the previous articles. Some search engines rewrite their search results links to track which ones are clicked most often. We therefore implemented a "best-match" algorithm, which compares candidate elements against the information recorded about the original click.

First, we narrow candidates to elements with the same tag name, e.g., "A" or "INPUT". We then build a match record for each one, comparing on several characteristics:

- Exact match (index 0)
- The "id" (1), "href" (2), "name" (3), "type" (4), and "value" (5) attributes
- The text content of the element (6)
- Information about element's parent form (if any) (7)

Say the candidate element is E; we denote a characteristic of E by $E.c$ where c is, e.g., the value of the `id` attribute. Denote the log-recorded value of c by $O.c$.

Then we make a match record R as follows: for each characteristic c with index i,

$$R[i] = \begin{cases} 0 : E.c \neq O.c \\ 1 : E.c = null \wedge O.c = null \\ 2 : E.c = O.c \end{cases}$$

Then the best match record is selected by comparing the record values in order, i.e., $R_i[0]$ vs. $R'_j[0]$, then $R_i[1]$ vs. $R'_j[1]$, et seq.

Finally, it may be that the best possible match is not in fact a very good match. We empirically determined a cutoff in match scores and only consider the element a match if it passes this cutoff.

3.5.1.3 Mirroring the click.

After we have correctly identified the element in the target document, mirroring the click is relatively straightforward. We construct a click event object and deploy it to the target element. We also precede each click event with a focus event, as it would be if the user in fact clicked the mouse on it. This step is important because many web pages use "onfocus" Javascript handlers to change the page dynamically when an element is focused.

3.5.2 Forms

In addition to clicks, we must fill in web forms in the target window with the same data as in the original. Some of the challenges here are similar to the click problem above, in that forms do not always have unique identifying information such as a "name" attribute. If the "name" attribute is missing, we use the form's array index in the `document.forms` array; unlike clickable elements, it is unusual for forms to be added and removed by chance. The form elements virtually always have "name" properties because that is how their values are identified when the form is submitted, so identifying them within a form is easy.

There is a more subtle issue here, though, because forms often have hidden fields which are meant to be unique for each instance. Common fields of this sort are session IDs and nonces for password protocols. We do not modify the values of hidden form fields; in practice this has not been a problem since the user is only expected to fill in visible fields anyhow.

We fill in forms in the target document when replicating each click, not just when the form is submitted. Correspondingly, we store all form values at the time we record the click information. This is because Javascript on the page may modify the page based on the form values prior to submission, sometimes even reloading the page in response.

3.5.3 Other issues with replicating user actions

One complication we encountered in replicating user actions is the presence of frames; in the interest of space, a full discussion is omitted. The main idea for handling frames is that many of the above techniques can be implemented recursively, effectively treating all the frames as one giant page.

Another question is how precise to be in recording and replicating user actions. Omitting certain frequent events yields an efficiency benefit. So far we have not found it necessary to replicate each keystroke or mouse movement the user makes, although in principle these are events that the page can handle and respond to. Most often keyboard events are used either for scrolling or text entry, neither of which must be replicated at that granularity.[4] Mouse movement events generally result in, at most, superficial changes to a page, e.g., a dropdown menu when hovering over an element. So long as the menu links are accessible through the DOM for the page, it does not matter if they are visible or not during replay. If it becomes necessary, we can easily log and replay these events as well.

---

[4]Keyboard shortcuts are becoming more common, and we plan to record non-navigation keystrokes in the future.

Site(s)	Purpose / actions
Yahoo!	Check Yahoo! mail, news, TV listings
Netflix	Research movie reviews
GMail	Check email
CNN	Read news
Verizon Wireless	Research cell phone plans
Google, etc.	Research MP3 player purchase

Figure 8: Summary of browsing session for evaluation.

4. EVALUATION

We evaluated the effectiveness of Doppelganger versus various built-in browser settings by performing a script of common browsing tasks, summarized in Figure 8. In testing, we simulated a user who is willing to accept a certain amount of privacy loss for convenience at sites with whom he has a relationship (in this case, Yahoo!, Netflix, and GMail) but is more cautious at sites with whom he has no relationship (CNN, PC Magazine, Vanns.com, ComputerHQ.com, BeachCamera.com).

For each setting, we measured (1) the number of sites whose cookies were accepted, categorized by persistence and context, and (2) the inconveniences suffered by the user, including dialog boxes and lost functionality. An ideal scheme would incur a low number of each. The five settings we tested were four global settings (same for all sites): (1) All cookies enabled, (2) First-party cookies only, (3) First-party session cookies only; (4) Ask the user what to do for each cookie that is sent; and finally (5) using Doppelganger. We measured the number of sites rather than the number cookies because multiple cookies of the same type from the same site are equivalent from a privacy perspective. We executed each script by hand three times consecutively for each setting, retaining any state between runs. The idea was to capture the effects of session cookies, persistent cookies, and, for Doppelganger, the change in policy and behavior over time. We cleared all cookie-related state before changing settings.

The accepted-cookie results are shown in Figure 9, and the details of each setting and its corresponding user experience are described below. Two values are particularly interesting. The number of sites setting persistent cookies is significant because they allow users to be tracked over many sessions, and the number of sites setting third-party cookies is perhaps more so because they let the user be tracked across multiple sites. Third-party persistent cookies combine the worst of both.

There are three ways in which the user can be "inconvenienced" during our script: he can be asked to answer a browser's yes-or-no cookie dialog (see picture); he can be asked a left-or-right browser decision by Doppelganger (see Figure 5); or he can be forced to login upon each visit to a site where he has an account. This latter case occurs when his browser could have accepted a persistent cookie that would serve as an authenticator, but did not for some reason.

In the following discussion, we use figures from the last session of each setting, as that most closely represents a steady-state figure.

4.1 All cookies

This is the default setting in Firefox, and, perhaps predictably due to its permissiveness, led to the acceptance of the most cook-

	Number of sites setting:				
Run	**FP-S** cookies	**FP-P** cookies	**TP-S** cookies	**TP-P** cookies	**Total persistent (FP-P + TP-P)**
All cookies on (Run 1)	9	8	3	13	**21**
All cookies on (Run 2)	8	8	3	16	**24**
All cookies on (Run 3)	8	8	3	16	**24**
FP only (Run 1)	9	8	1	4	**12**
FP only (Run 2)	8	8	2	5	**13**
FP only (Run 3)	8	8	2	7	**15**
FP session only (Run 1)	9	0	9	0	**0**
FP session only (Run 2)	9	0	6	0	**0**
FP session only (Run 3)	9	0	8	0	**0**
Ask user (Run 1)	4	3	0	0	**3**
Ask user (Run 2)	3	3	0	0	**3**
Ask user (Run 3)	3	3	0	0	**4**
Doppelganger (Run 1)	4	0	0	0	**0**
Doppelganger (Run 2)	3	3	0	0	**3**
Doppelganger (Run 3)	3	3	0	0	**3**

FP = "First party" TP = "Third party" S = "Session" P = "Persistent"

Figure 9: Number of sites setting cookies while performing some common tasks. A script of common browsing tasks was run three times in succession for a variety of cookie management policies, and we measured the number of sites setting various kinds of cookies. "First party" here refers to sites that the user intended to visit, and "third-party" refers to all other sites.

ies of any setting: 24 sites set persistent cookies, including 16 in third-party context. Virtually every domain we visited set a persistent cookie (none sent only session cookies). This suggests that any future browsing sessions would be extensively tracked. The advantage of this permissive policy was that we were never asked any questions during the session.

4.2 First-party only

Since the danger of third-party cookies was recognized years ago, many users disabled third-party cookies in their browsers; the size of this group has forced sites to avoid any dependence on these cookies. Thus a first-party only setting in the browser is, in practice, a big win over the "allow all" setting. Indeed, we accepted persistent cookies from a little more than half as many sites as in the default settting. However, the browser still accepted many cookies that either do not confer any benefit or were not worthwhile. These include persistent cookies from 7 sites we did not mean to interact with; these were accepted because of redirection and framing tricks. Furthermore, there were unneeded cookies from first-party sites—8 vs. 3 for Doppelganger—since we did not need cookies from, e.g., `cnn.com`. The main advantage of first-party only vs. more restrictive settings is that we were not asked any questions.

4.3 First-party session only

This setting downgrades all persistent cookies to session cookies with the aim of eliminating long term tracking. It was still vulnerable to tricks which forced us to accept session cookies from 6 to 9 sites (it varied across runs, because advertisements change) that we did not mean to interact with. The session-only restriction came with a significant downside: we were forced to log in to each site with which we had a relationship during every session, and would have had to do so indefinitely. All personalization features that do not require a login would also be lost.

4.4 Ask the user

The final built-in browser setting we tested was one in which the browser asks the user whether to accept each cookie as it was offered. This dialog box allowed a decision to apply to all cookies from the same site, and we checked this box each time to reduce the number of dialogs.

An example of the dialog box is shown here:

This setting poses something of a dilemma: in principle, we do not know which cookies to accept and which to deny, especially since most of the dialogs appear before a site's home page even finishes loading. Our solution was to use Doppelganger to determine which cookies were useful and, thus, always answer questions correctly, including when to accept persistent cookies and when to downgrade them to session cookies. While this assumes a somewhat oracular user, it helps put a lower bound on the difference between Doppelganger and the best-case scenario for existing browser settings.

Unsurprisingly, the Ask setting resulted in a dramatic reduction in the number of accepted cookies compared with other browser settings: there were only 4 sites which set persistent cookies, and no third-party cookies were accepted.

There was no loss of functionality with this setting, as there was with the "First-party session only" setting, but there was a significant problem: we were shown 26 dialog boxes during our session. Of these, 13 were due to first-party sites, and 13 due to third-party sites using various tricks. These dialogs presented almost no infor-

mation to help the user decide whether to accept the cookie, except for the domain name.

4.5 Doppelganger

Our last test used Doppelganger for cookie management. Unsurprisingly, the cookie results were virtually identical to the Ask policy, since we used information gained by Doppelganger to answer the browser's questions during that trial. The total number of persistent cookies rose from Run 1 to Run 2 because Doppelganger reserves judgment on persistent cookies until it can test their usefulness in a subsequent session; ultimately, 3 sites' persistent cookies were accepted vs. 4 for the Ask setting.

We did not have to answer nearly as many questions using Doppelganger as we did with Ask. During the first run, there was a comparison dialog on the `netflix.com` homepage, because session cookies were required to use the site. This dialog does not appear in low or medium paranoia mode. An error message at `verizonwireless.com` prompted a click of the Fix Me button; this solved the problem without further effort; this is not necessary in low paranoia mode. During the second run, `yahoo.com`, `netflix.com`, and `mail.google.com` (Gmail) all had automatic login features if persistent cookies were enabled. This resulted in side-by-side comparisons, and in each case we chose to accept the cookie in exchange for the convenience. At `verizonwireless.com`, a persistent cookie remembered our zip code, prompting a comparison; we chose not to accept a persistent cookie, because we did not plan to visit the site often. During the third run, there were no dialogs at all. Indeed, by that time, Doppelganger had already silently decided not to accept cookies from `cnn.com`, `pcmag.com`, and `dealtime.com` (a site through which we were redirected each time we clicked on vendors from `pcmag.com`).

It is important to note that none of the dialogs we saw would ever recur; once a decision has been made, it is remembered for future sessions. While the same is true for the Ask policy, the test script is heavily weighted towards sites the simulated user has a relationship with. Thus, new sites the user encounters are likely to be ones with which there is no relationship or which are visited less frequently, and thus where cookies are much less likely to have value. This is significant because the Ask policy pops up dialogs regardless of cookie value, while Doppelganger does so only if cookies are likely to be useful, and in fact shows the user *how* useful, as well as relevant privacy information. Turning on the optimization above would make the number of Doppelganger dialogs smaller still.

5. WEB SITE COUNTERMEASURES

If Doppelganger becomes widely deployed, web sites might try to circumvent or fool its mechanisms to store persistent data on users' machines. In this section, we discuss various approaches they might take and how those approaches affect Doppelganger.

5.1 Always require cookies

A web site might require users to always accept cookies by redirecting users to an error page if it detects cookies are being blocked. Many sites currently do this. However, addressing this problem only requires Doppelganger to accept session cookies from the site, which have limited privacy risks; and if the user is using low or medium paranoia mode, Doppelganger will do so automatically. For very privacy-conscious users in high paranoia mode, Doppelganger will expose the sites' cookies requirements for inspection. In addition, web sites might try to require users to accept persistent cookies by requiring an extensive "sign-up" procedure if it does not detect a persistent cookie on the user's machine. This might encourage users to accept persistent cookies to avoid repeating this procedure on subsequent visits. However, this approach would likely alienate users; privacy conscious users may not want to accept persistent cookies or might routinely delete all their cookies, and for privacy reasons, public kiosks and library terminals must delete all cookies after each user's session.

5.2 Cause spurious differences

A web site might try to always create subtle or inconsequential differences between pages requested with cookies and those requested without cookies to frequently trigger Doppelganger's comparision dialog. A user who receives excessive comparison dialogs for a site might become annoyed and decide to accept the cookies to prevent future interruptions, or worse, disable Doppelganger entirely. Doppelganger's current difference detection algorithm is simple: we compare the page titles and look to see if the user's name or ID is only in the fork browser. Developing sophisticated and robust difference detection is important future research for Doppelganger, and we see two promising directions. One approach is compare pages structurally by examining their DOM trees. This is based on the assumption that substantive changes often result in the addition or removal of whole page elements. Another complementary approach is a visual comparison, comparing screen captures of the fork and main windows. Both of these approaches require filtering advertisements and other sources of randomness before comparison.

5.3 Other persistent objects

If a web site detects its persistent cookies are being blocked, it might resort to storing other persistent objects on the user's machine (e.g., flash objects, images, Javascript) and retrieving these objects in subsequent sessions. To fully address privacy with respect to persistent web objects, Doppelganger must apply its cookie policy for a domain to manage other cached objects from that domain as well. However, there may be well-intentioned sites that don't use cookies at all, but cache web objects on users' machines to improve performance. Fully understanding the effect of this approach on the user's web browsing experience requires further study.

6. FUTURE WORK

Currently, Doppelganger only has mechanisms to incrementally relax cookie policies, but not make them more restrictive. Users may later change their minds about the cost/benefit tradeoff for certain domains and want to make their cookie policies for those domains more restrictive. Exploring usable mechanisms for "tightening" the cookie policy is an area for future exploration.

Comprehensive user studies are needed to understand the usability of Doppelganger's mirroring and automated recovery mechanisms. We have recently conducted a controlled user study to examine the usability and performance of Doppelganger and plan to publish the results in the near future.

7. CONCLUSION

We introduced Doppelganger, a novel system for creating and enforcing fine-grained, privacy preserving cookie policies in web browsers with low manual effort. We showed how Doppelganger automatically identifies cookies which provide users additional functionality and exposes the costs and benefits of accepting those cookies. As with most markets, more complete information has the po-

tential to lead to more efficient outcomes. In this case, that means that users will be able to select those sites that offer benefits commensurate with the users' privacy loss over sites with less favorable exchanges. Indeed, since subjective feelings of trust in users often induce users to accept more privacy costs, steps by sites to increase transparency—such as publishing a useful privacy policy—-may actually increase the amount of usable personal information they obtain. In short, we believe that systems like the one we describe here can lead to better incentives for both parties. Thus, while our work here certainly does not expose all costs and benefits, and only deals with one aspect of online privacy (viz. tracking), we believe that it represents a meaningful step forward down the right path.

Acknowledgements

8. REFERENCES

[1] Alessandro Acquisti and Jens Grossklags. Privacy and rationality in individual decision making. *IEEE Security and Privacy*, 3(1):26–33, 2005.

[2] Add & Edit Cookies. http://addneditcookies.mozdev.org/.

[3] Adil Alsaid and David Martin. Detecting web bugs with bugnosis: Privacy advocacy through education. In *Proceedings of the 2002 Workshop on Privacy Enhancing Technologies*, 2002.

[4] Fahd Arshad. Privacy Fox - A JavaScript-based P3P Agent for Mozilla Firefox. http://privacyfox.mozdev.org/PaperFinal.pdf, 2004.

[5] Aaron Brown and David Patterson. Undo for operators: Building an undoable e-mail store. In *USENIX 2003 Annual Technical Conference*, June 2003.

[6] Simon Byers, Lorrie Faith Cranor, and David Kormann. Automated analysis of P3P-enabled web sites. In *ICEC '03: Proceedings of the 5th International Conference on Electronic Commerce*, pages 326–338, New York, NY, USA, 2003. ACM Press.

[7] Electronic Privacy Information Center and Junkbusters. Pretty poor privacy: An assessment of P3P and Internet privacy. http://www.epic.org/reports/prettypoorprivacy.html, June 2000.

[8] Cookie Button. http://basic.mozdev.org/cookiebutton/.

[9] Cookie Culler. http://cookieculler.mozdev.org/index.html.

[10] Cookie Toggle. http://cookietoggle.mozdev.org/.

[11] Mary J. Culnan and Pamela K. Armstrong. Information privacy concerns, procedural fairness, and impersonal trust: An empirical investigation. *Organization Science*, 10(1):104–115, 1999.

[12] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. Dos and Don'ts of client authentication on the web. In *10th USENIX Security Symposium*, pages 251–268, August 2001.

[13] Jeremy Goecks and Elizabeth D. Mynatt. Social approaches to end-user privacy management. In Lorrie Faith Cranor and Simson Garfinkel, editors, *Security and Usability: Designing Secure Systems That People Can Use*, chapter 25, pages 523–545. O'Reilly, 2005.

[14] Nathan Good, Rachna Dhamija, Jens Grossklags, David Thaw, Steven Aronowitz, Deirdre Mulligan, and Jospeh Konstan. Stopping spyware at the gate: A user study of notice, privacy and spyware. In *Symposium on Usable Privacy and Security (SOUPS) 2005*, July 2005.

[15] Il-Horn Hann, Kai-Lung Hui, Tom S. Lee, and I. P. L. Png. Online information privacy: Measuring the cost-benefit trade-off. In *Proceedings of the Twenty-Third International Conference on Information Systems*, pages 1–8, 2002.

[16] Stephen E. Levy and Carl Gutwin. Improving understanding of website privacy policies with fine-grained policy anchors. In *WWW '05: Proceedings of the 14th International Conference on World Wide Web*, pages 480–488, New York, NY, USA, 2005. ACM Press.

[17] LiveHTTPHeaders Firefox extenstion. http://livehttpheaders.mozdev.org/.

[18] Lynette Millett, Batya Friedman, and Edward Felten. Cookies and web browser design: Toward realizing informed consent online. In *Proceedings of the CHI 2001 Conference on Human Factors in Computing Systems*, pages 46–52, April 2001.

[19] Gavin O'Malley. Jupiter analyst: Nielsen research confirms users delete cookies. http://publications.mediapost.com/index.cfm?fuseaction=Articles.san&s=2%8883&Nid=12855&p=297686.

[20] Permit Cookies. http://gorgias.de/mfe/.

[21] Persistent client state: HTTP cookies, Preliminary specification. http://wp.netscape.com/newsref/std/cookie_spec.html.

[22] Kevin Poulsen. Microsoft cookies jump domains. http://www.securityfocus.com/news/83, September 2000.

[23] Sites use IFrames to bypass cookie prefs. https://bugzilla.mozilla.org/show_bug.cgi?id=158463.

[24] The Platform for Privacy Preferences Project (P3P). http://www.w3.org/TR/P3P/.

[25] View Cookies. http://www.bitstorm.org/extensions/view-cookies/.

[26] Ka-Ping Yee. Guidelines and strategies for secure interaction design. In Lorrie Faith Cranor and Simson Garfinkel, editors, *Security and Usability: Designing Secure Systems That People Can Use*, chapter 13, pages 247–273. O'Reilly, 2005.