Bridging the Gap between People and Policies in Security and Privacy

by

Umesh Shankar

B.A. (Harvard University) 1999
M.S. (University of California, Berkeley) 2002

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor David Wagner, Chair
Professor J. D. Tygar
Professor Marti Hearst

Fall 2006

The dissertation of Umesh Shankar is approved:

Chair Date

Date

Date

University of California, Berkeley

Fall 2006

Bridging the Gap between People and Policies in Security and Privacy

Portions of this work drawn with permission from [86, 87]:

Umesh Shankar and Chris Karlof. "Doppelganger: Better Browser Privacy Without the Bother". In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006)*, October 2006.

Trent Jaeger, Reiner Sailer, and Umesh Shankar. "PRIMA: Policy-Reduced Integrity Measurement Architecture". In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT 2006), June 2006.

Abstract

Bridging the Gap between People and Policies in Security and Privacy

by

Umesh Shankar

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor David Wagner, Chair

The most powerful of security and privacy mechanisms may be rendered ineffective if people cannot use them. A common usability problem is that it is hard to specify the policies that the mechanisms enforce. Indeed, the more powerful the mechanism, the larger and more complex its policy can be; this makes it difficult not only to write a policy down, but also to make sure that an existing policy is a secure one.

In this dissertation, we make progress in addressing both these problems: *translating* people's high-level intentions into low-level policies and *verifying* that low-level policies meet high-level goals. To this end, we explore two application domains and their corresponding user bases.

For system administrators, we define a useful secure information-flow property, which we term *CW-Lite*. It says that untrusted processes should not be able to send unfiltered inputs to trusted processes. This is a basic security concern which can lead to system compromise, but it is unverified on most systems today because there is no effective, easy way to do the verification. A big advantage of our approach is that system administrators can perform a completely automated verification of CW-Lite using our tools, making it easier to integrate into a system.

With *Doppelganger*, an extension to the Firefox browser, we target a wider audience. Web browser *cookies* are used to manage relatively benign session state such as shopping carts, but also—almost ubiquitously—to track and record users' actions across sites and sessions, representing a significant privacy risk. Doppelganger seeks to generate a good

cookie policy for each user, one that reflects that user's privacy vs. functionality cost-benefit curve, in an automated way. It uses several techniques: it automatically determines when certain cookies yield no benefit; when necessary, it asks the user to make a few informed, high-level decisions; and lastly, it offers a one-click error-recovery mechanism. We evaluated Doppelganger for privacy and usability in two experiments, including a controlled usability study with 18 users. In both cases, we found that Doppelganger offered greater privacy than the built-in browser settings, and that the cost in usability was modest.

Professor David Wagner
Dissertation Committee Chair

Contents

List of Figures **iv**

List of Tables **vii**

1 Introduction **1**
1.1 Background and Motivation 1
1.2 Target Domains and Approach 4
1.2.1 Bridging the Gap 4
1.2.2 Automating Translation and Verification 5
1.3 Summary of implementation and results 6
1.3.1 Operating system security policy (Chapter 2) 6
1.3.2 Web browser privacy policy (Chapters 3 and 4) 7

2 Related Work **10**
2.1 Security policy configuration and verification 10
2.1.1 Integrity vs. confidentiality 11
2.1.2 A note on covert channels 12
2.1.3 Formal models of integrity 13
2.1.4 From theory to practice 14
2.1.5 Specialized models 16
2.1.6 Firewall policy management implementations 16
2.2 Browser privacy and usability 17
2.2.1 Cookies and tracking 17
2.2.2 Existing tools for cookie management 18
2.2.3 P3P 20
2.2.4 Privacy and usability 20
2.2.5 Recovery-Oriented Computing (ROC) 23

3 CW-Lite: Verifying an information-flow property of OS security policies **24**
3.1 Introduction 24
3.1.1 Motivation and Goals 24

3.1.2 Contributions . . . 27
3.1.3 Roadmap . . . 27
3.2 Overview of CW-Lite and its Verification . . . 28
3.3 CW-Lite . . . 30
3.3.1 Basic Information-Flow Integrity . . . 30
3.3.2 Clark-Wilson Integrity . . . 31
3.3.3 CW-Lite . . . 33
3.4 Developing CW-Lite-Compliant Systems . . . 34
3.4.1 SELinux . . . 37
3.4.2 Supporting filtering interfaces in the MAC Policy . . . 38
3.4.3 MAC Policy Analysis . . . 39
3.4.4 Finding filtering interfaces . . . 41
3.4.5 Handling illegal information flows . . . 42
3.5 Example: CW-Lite Integrity Verification . . . 43
3.5.1 Goal . . . 43
3.5.2 Setup . . . 43
3.5.3 Roadmap for OpenSSH . . . 45
3.5.4 Inter-process Flow Analysis . . . 46
3.5.5 OpenSSH Filtering Interfaces . . . 49
3.5.6 Verifying vsftpd . . . 50
3.6 Related Work . . . 50
3.6.1 Integrity Models . . . 50
3.6.2 Policy Analysis . . . 51
3.6.3 Whole-system Analysis . . . 52
3.7 Summary . . . 52

4 Doppelganger: better browser privacy without all the bother 54
4.1 Introduction . . . 54
4.1.1 Background . . . 54
4.1.2 A solution: Doppelganger . . . 56
4.1.3 Some implications of our approach . . . 57
4.1.4 Contributions: . . . 57
4.2 HTTP cookies . . . 58
4.2.1 Uses of cookies . . . 60
4.2.2 Web browser cookie management . . . 60
4.2.3 “This site requires cookies” . . . 63
4.2.4 Cookie management: The state of the art . . . 63
4.3 How Doppelganger works . . . 65
4.3.1 An example . . . 67
4.3.2 Mirroring . . . 69
4.3.3 User initiated error recovery . . . 77
4.3.4 Higher-privacy modes . . . 79

4.3.5 Replicating individual user actions . . . 79
4.4 Evaluation . . . 84
4.4.1 All cookies . . . 85
4.4.2 First-party only . . . 85
4.4.3 First-party session only . . . 87
4.4.4 Ask the user . . . 87
4.4.5 Doppelganger . . . 88
4.5 Web site countermeasures . . . 89
4.5.1 Always require cookies . . . 89
4.5.2 Cause spurious differences . . . 90
4.5.3 Other persistent objects . . . 90
4.6 Future Work . . . 91
4.7 Conclusion . . . 91

5 Doppelganger usability study **92**
5.1 Introduction . . . 92
5.2 Setup . . . 93
5.2.1 Profile of the subjects . . . 93
5.2.2 Method . . . 93
5.3 Results . . . 95
5.3.1 Ease of use (tabular data in Figure 5.3). . . . 95
5.3.2 Performance . . . 97
5.4 Discussion . . . 100
5.5 Conclusion . . . 103

Bibliography **105**

List of Figures

1.1 **Bridging the Gap.** The focus of our work is on bridging the gap between high-level properties and low-level policy languages. 3

3.1 **Application developer tasks required to enable CW-Lite verification.** Filtering interfaces are those that accept inputs from untrusted sources, and must sanitize, or *filter* the input. An interface is marked by a distinct call to `open()`, `accept()`, or other call that enables data input. The `DO_FILTER()` annotation on an interface tells the access-control system to grant additional permissions allowed by the filtering subject type to that interface. The default subject type, used on all other input interfaces, only allows inputs from the the system TCB. 34

3.2 **System Administrator tasks required to verify CW-Lite.** The system administrator decides on a system TCB initially. Then, when she wants to verify CW-Lite for a particular trusted application, she runs Gokyo on its security policy. If no errors are reported, CW-Lite integrity is verified. If it reports an illegal flows, the offending permissions must be removed, or the TCB expanded to include the source of the illegal flows. 35

3.3 **Supporting filtering interfaces.** Initially, the program above is not allowed to accept network input, because the network is not in the TCB. In order to accept such input, the source code must filter it and the programmer must supply an annotation indicating that the interface is filtered. Then the policy must be modified to allow the network input only for the filtering interface. The `DO_FILTER()` macro tells the MAC system to use the filtering subject type permissions for the enclosed operations. We annotate `accept()` (which implies a read/write socket), rather than subsequent socket read/write operations, because that is where the MAC system performs access checks. This is analogous to how file access checks, including read/write permission checks, are performed once on `open()`, not for every `read()` or `write()` call. 38

3.4 **Gokyo support for SELinux object relabeling** 40

3.5 **Process structure of privilege separated OpenSSH.** The *listen* process simply processes new connection requests, forking a *priv* process to handle each one. The *priv* process forks a *net* process to perform the network portion of user authentication, providing a narrow interface to privileged operations necessary to complete authentication. After *net* completes its task, *priv* spawns the authenticated user's requested process, in our example a bash shell. 44

4.1 **Summary of Doppelganger's cookie management mechanisms.** 55
4.2 **Example of a site's instructions on how to enable cookies.** The instructions for both Firefox and Internet Explorer tell the user to enable all cookies, including third-party cookies. 61
4.3 **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 4.6). Doppelganger also maintains a log of the user's actions for error recovery. . . . 64
4.4 **Graphical representation of mirroring for the example in Section 4.3.1.** 68
4.5 **An example use of Doppelganger's error recovery mechanism.** Suppose a decision not to accept cookies was made in the past, but cookies are needed for a shopping cart feature on a site. Doppelganger does not employ mirroring because it has already formed a policy. 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. . . 70
4.6 **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. 73
4.7 **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 4.3.4). 75
4.8 **Summary of Doppelganger's different privacy modes.** 77
4.9 **Summary of browsing session for evaluation.** 85

4.10 **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, such as from third-party images, IFRAMEs, and click-tracking HTTP redirections. 86

5.1 **Usability study survey results — General questions** 94
5.2 **Usability study survey results — Familiarity with cookies** 95
5.3 **Survey results — ease of completion**. Doppelganger and Ask were the second and third scenarios; for 10 of the 18 subjects, Ask came before Doppelganger. 98
5.4 **Task list used in Doppelganger usability study.** 99
5.5 **Cookies accepted for each cookie management setting.** The figures used here are the averages across all users, taken from the cookie jar as of the end of task list (Figure 5.4). Each type of cookies from each site counted as "1" in the tally; i.e., we did not distinguish between accepting two first-party session cookies from a site and accepting three first-party session cookies from the site. The end of the task list represents the end of the second browsing session, since users had to close and restart the browser after Task 3. Error bars represent the standard deviation. 101

List of Tables

3.1 Information flows to our target subjects (*priv* and *listen* for OpenSSH and *ftpd* for vsftpd) that may lead to integrity problems. 47

Acknowledgments

I owe a great debt to my advisor, David Wagner, who found time between world-beating works to guide me from being a naive and ignorant first-year student to being somewhat less of each. He encouraged me to pursue my own lines of inquiry, which were often quite different from his, and apologized profusely for those few times when the situation required that I do some particular research task.

I would also like to thank my co-authors and colleagues, from whom I learned a tremendous amount. Kunal Talwar and Jeff Foster collaborated with me on my first published paper. Vern Paxson's lightning-quick insight and no-nonsense approach made him a wonderful mentor and collaborator, and made me a better researcher. Fellow graduate students Naveen Sastry, Monica Chew, and Chris Karlof made collaboration easy with their friendship; and my officemates Rob Johnson and Ben Liblit always seemed to have patient answers to my PL and "my system is broken" questions. David Molnar was always ready to provide related-work references and read a draft of a paper on a moment's notice; AJ Shankar and Marco Barreno came through with reading and testing when I really needed it. Trent Jaeger and Reiner Sailer not only helped me learn about trusted computing as we collaborated but patiently worked with me from three time zones away. Marti Hearst and Doug Tygar changed the way I thought about my own work and computer security in general, bringing me numerous new big-picture insights.

Last, but far from least, I want to express my deep gratitude to my family and friends, whose love and support made good times the norm, rough times smoother, and whose frequent but reasonable queries of "So when are you going to finish?" propelled me to completion.

—Umesh

Chapter 1

Introduction

1.1 Background and Motivation

The impetus for this dissertation is simple. I, like many others, have suffered much irritation—and resultant insecurity—by systems whose configuration interfaces are overly complex and ineffective; that is, not very usable. There is a good reason why getting usable security right is difficult: security and privacy violations are often invisible and hard to understand, but loss of easy access to functionality is obvious.

Recently, the tradeoffs in information security and privacy in computer security and privacy have been modeled as microeconomic problems. Vila et al. [94] argue that people don't bother to read privacy policies on web sites because there is a "lemons market" for privacy. That is, many sites have poor privacy practices, and consumers who can't tell the difference simply and rationally assume the worst. Put another way, it is hard to make an individualized cost-benefit analysis when you don't know what the costs and benefits are. It is especially easy to make the wrong decision when you only know what the costs are, namely, inconvenience, but cannot see the benefit you receive in not having your personal data collected and sold to third parties. Gordon and Loeb did an economic analysis [48] of investments in information security, and found that there may be good reasons to protect only some data, and not necessarily the most sensitive data. Again, the difficulty—that is, cost—of securing the data was a critical factor in their conclusion.

Our overarching goals in this dissertation are twofold: first, make the analysis of what to

protect easier, by exposing costs and benefits; and second, to lower the costs themselves by providing more usable mechanisms. In particular, we focus on the problem of *configuring* security and privacy mechanisms. In our opinion, the capability for good security, and to a lesser extent, privacy, exists already. It is wringing that ability out in practice that is difficult, and configuration is a primary obstacle. (This is neatly summarized by Balfanz et al. [9] in a retrospective on deploying a more usable wireless network setup scheme: "Tools aren't solutions.")

In the interest of examining these issues in more detail, first let us lay down some working definitions. We will say that a system is *secure* if actions that its operator does not want to happen, cannot happen, and preserves *privacy* if it does not reveal information that a user does not want to be revealed. We also say that a system is *usable* if tasks the user wants to perform can be made to happen easily.

Right away, we can see a tension between these goals. In order to satisfy all the requirements at once, a system must be a perfect filter, reflecting its user's intentions exactly with respect to which actions may be performed and which information may be inferred during that process, disallowing all other actions and disclosures. Such systems are naturally very difficult to build. It would be much easier if everyone were the same, in which case we could hard-wire the correct set of features and policies once and for all. In the real world, of course, the same products are used by many different kinds of people, and so the products must be highly configurable. As soon as users have choices, they can make mistakes.

Unfortunately, the way many systems work today is to start with certain defaults (occasionally, there is a choice of standard profiles), then require any further customization to be done through low-level languages for which little support is given. Rarely does the user know if she has made a serious security-relevant configuration error—say, configuring a web server to allow everyone access to administrative pages—since such errors allow too much, not too little. The latter case is easy to detect because legitimate users will be denied access and complain, but the former may go undetected until an attack occurs, and perhaps not even then. This is true even for security-sensitive applications like the Apache [7] web server—the mostly widely deployed server on the Internet [73]—and the OpenSSH [77] secure shell server. The administrator enters a text configuration file, and the server uses

it. Attackers will never complain if that file allows them to access more than they should be able to. Some applications, such as the Gallery program [43] that serves web-based photo album, perform a security configuration check when they are installed. This is a nice feature, and a useful one. However, many security configuration problems arise because of interactions between programs that no one program can anticipate or verify. Furthermore, these kinds of errors are not going to be solved through software patches, nor publicized on vulnerability mailing lists: they are local, site-specific problems, but no less damaging as a result. In short, users need a way to verify that certain high-level properties (e.g., restricted access to administrative controls) are preserved by low-level configuration policies.

The converse problem exists as well, especially for users with little technical expertise. Such users still have security and privacy preferences but may not have the means to express them using a policy configuration language or intricate dialog boxes. Wherever possible, then, we should strive to make the power of fine-grained control available in a user-friendly way; that is, to be able to translate each user's requirements into a low-level policy.

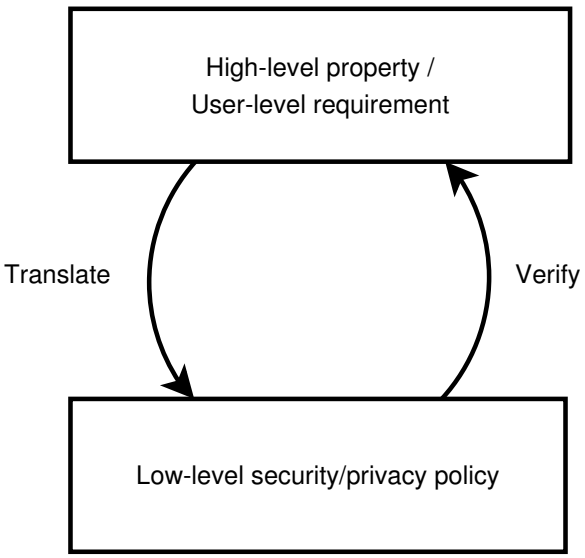


Figure 1.1: **Bridging the Gap.** The focus of our work is on bridging the gap between high-level properties and low-level policy languages.

In summary, there are two tasks we can do on users' behalf: we can translate high-level goals or intentions into low-level policy, which is particularly helpful for users with coarse preferences or preferences that are easier to express operationally; and we can verify low-level policies to ensure that they meet high-level goals, which is particularly helpful for system administrators who need to edit low-level policy but want to make sure that no broad security invariants are violated inadvertently in the process. Both these automations serve to reduce the cost of security and privacy enforcement, and so make expending security effort a rational choice for more users.

1.2 Target Domains and Approach

1.2.1 Bridging the Gap

The particular problem we are trying to address is summarized in Figure 1.1: we want to make it easier to keep low-level policies and high-level goals in sync. Keeping effective security and privacy within the reach of users is particularly important because these are adversarial situations. Unlike conventional systems, failures are not random; even small configuration mistakes can be blown up into large breaches. While we cannot very well deploy a "perfect" configuration to each user, what we *can* do depends on the type of application we are working with. One broad distinction we may make is between *interactive* and *non-interactive* applications.

Let us define interactive applications to be those primarily concerned with accepting and processing user input. Web browsers, word processors, and other office applications are good examples. These applications often have broad, non-technical user bases, so solutions aimed at them should be particularly user-friendly. On the other hand, their interactive nature means that, first, we have time to do some processing at runtime, and second, that we can get feedback from the user. This latter capability is very useful, as it allows us to iteratively refine the underlying policy.

Non-interactive applications are those meant to run largely unattended. Network intrusion detection systems, operating system access controls, and web servers are typical examples. These applications are often configured by more skilled users, like system administrators, who may write at least parts of the security policy by hand to match their operating needs. The automated nature of these systems means that they usually process a large number of transactions or authorizations in the background, so runtime interruptions are not just undesirable; oftentimes there will not even be a user present. Accordingly, we prefer *static analyses* for such systems, ones that perform verifications before the system even starts running.

1.2.2 Automating Translation and Verification

From an implementation standpoint, our focus will be on automating the translation and verification steps for particular policy languages. That is, we will build systems that take high-level input and observations and generate a policy automatically, or which verify high-level properties from a policy automatically, with little or no manual intervention. After all, the goal is to make things easier for users, with the expectation that if things are easier, people will use them more, improving practical security and privacy.

Let us next note that while the security and problems we are trying to address are widespread and real, we have defined the properties to be verified or translated in such a way as to make them amenable to automated analysis. Actually, this is an important step. It is useful to have a clear idea of the goals and effectiveness of a particular method, so it is easier to know when we have made progress. It is our view also that a new system or policy language should not be developed in isolation, but rather with a view towards user-friendly, ideally automatic, security and privacy protections.

One way to do this is simply to make the user use a restricted language whose lack of expressive power makes it harder to express unsafe actions. Type-safe programming languages are one example; the programmer cannot violate type safety (see [97] for a formalism) because it isn't possible to express an invalid type conversion. Another interesting case, closer to our work, is the Firmato tool [10], which gathers network information and combines it with a high-level language to generate low-level firewall configurations. What you lose, of course, is flexibility; unsophisticated users may not have the wherewithal to write down a policy at all, and sophisticated ones may want to use the full expressive power of the underlying system.

We want to get the best of both worlds, if we can. We want to preserve and expose the underlying fine-grained mechanism to let us approximate an ideal policy as closely as possible, and to allow more sophisticated users to edit it directly. At the same time, we want automated generation and verification of the policy. In this thesis, we'll explore one interactive application, in looking at web browser privacy mechanisms, and one non-interactive application, operating system security policy enforcement. While hardly the final word on the subject, the hope is that in addition to making these steps forward we will

learn some lessons about what works and what does not for future efforts.

The policy-generation direction lends itself more naturally to interactive applications. Few users will be writing their own browser privacy policies by hand, but many users do have different (and currently largely uncaptured) privacy preferences and browsing habits. In Chapter 4, we will attempt to translate these preferences into a browser cookie policy during the browsing process without putting too much burden on the user.

Conversely, system administrators often have the expertise to write some of their security policies by hand, and to know what high-level security properties they want to preserve, but are overwhelmed by the sheer amount of information required to do this verification by hand. The non-interactive applications that they use are therefore good candidates for automated policy verification. Although a lot of attention has been given to security monitors in the research literature (see Section 2.1), in practice on most machines some basic security properties remain unchecked. In our treatment of the subject in Chapter 3, we'll look at some of the reasons why this old problem has persisted and try to make progress by setting reasonable goals that we can achieve using automated means.

1.3 Summary of implementation and results

1.3.1 Operating system security policy (Chapter 2)

One of an operating system's primary functions is to control access to resources, including files, network data, the screen, and so forth. From a security perspective, it is best if the OS implements *complete mediation*, that is, every security-relevant operation that a process can perform is given a yes-or-no decision. The more fine-grained the mechanism, the more control the administrator has over what is allowed and what is not. That means that the *principle of least privilege*, in which each process is given the minimal required set of permissions needed for it to work at any time, can be applied more accurately. But—and here we return to what will become a tired refrain—this power comes with complexity, and people are not good at dealing with that kind of complexity on their own.

It is therefore common for system security configuration mistakes to be made. When the policy is too restrictive, usually the problem is fixed, because something breaks and

the administrator is notified. When it is too permissive, the mistake is often unnoticed, and a sufficiently wrong configuration can lead to a compromise of the system. Since security policies can run to many megabytes, automated tools are essential to make sure that policy doesn't inadvertently have particularly gaping holes that the administrator does not intend. It is quite possible to create a significant vulnerability by changing just a few innocent-seeming permissions for different applications, which happen to interact in a way that magnifies the impact of the changes. Naturally, we want a way to ensure that small changes do not compose to break some bigger system-wide security invariant. Just such a tool is what we describe in Chapter 3. Of course, the tool needs to check some specific property; in this case, it verifies an *information-flow integrity property* of the system that we term *CW-Lite*. The idea is simple enough: we want to make sure that trusted processes on a system do not depend unduly on data controlled by untrusted users on that system. But there are several practical and historical reasons why this check is not routinely performed; we aim to change that situation here.

1.3.2 Web browser privacy policy (Chapters 3 and 4)

Web browser cookies are small data items stored in your browser by web sites in order to maintain session state for things like shopping carts. However, they can, and often are, used for privacy-compromising purposes like tracking your browsing actions or building extensive profiles of the kinds of sites you visit, what you buy, and when and from where you do all these things. Both Internet Explorer and Mozilla Firefox, which account about 96% of the browser market as of July 2006 [76], ship by default with policies that allow all cookies. That is because doing so allows all sites to work properly, and so causes the fewest support requests. It does not do much to protect users' privacy. There are more privacy-preserving settings available in the browsers, but they have drawbacks. They either (1) apply to all sites, which does not account for individual preferences, (2) do not protect privacy particularly well, or (3) require extraordinary user effort to maintain. The ideal system would figure out exactly which cookies the user "wants", that is, which ones have a benefit in functionality that, in the user's estimation, outweighs the privacy loss incurred.

We designed and built a new system, called *Doppelganger*, which manages browser

cookies and attempts to figure out which cookies to accept automatically. It does this by secretly mirroring the user's session at each site in a hidden window (hence the name). The hidden browser window is managed by Doppelganger, which replays the user's actions into it. The difference between the two windows is that the hidden window accepts more cookies that the visible one. In effect, we are taking a "partial derivative" with respect to those additional cookies. By comparing the two windows' contents, therefore, Doppelganger can see if the cookies yield any tangible benefits. If so, then it displays the two windows side-by-side, with differences highlighted (to show the benefit) and can provide the user with any available information on the privacy loss incurred (to show the cost); the user only needs to make a left-or-right-is-better decision.

Recognizing that mistakes are made, Doppelganger has an automated recovery mechanisms to fix many wrong decisions after the fact. In testing, Doppelganger yielded the same cookie set (or better) as the most effort-intensive, all-manual existing browser setting (even supposing an omnipotent user which made all the right decisions), but with considerably lower user burden. This was true in both our own manual tests and in controlled usability testing with eighteen test subjects.

Doppelganger collect high-level input from the user in different forms. It collects explicit input, such as when a user makes a choice between the main and mirrored windows, or when the user invokes the error-recovery mechanism. But it also collects implicit input: the way in which users use sites provides information as well. For example, if the user logs in to a site with username and password, Doppelganger automatically enables session cookies for the site, on the theory that a log-in is already more privacy-compromising than session cookies. Users may also use only portions of the site that don't benefit from having cookies enabled. For example, a user might use a shopping site for research but not for purchases; Doppelganger could discriminate between that user and one who makes purchases on the site if, say, only the shopping cart required cookies to function. Doppelganger combines the information from these inputs and translates it into a low-level policy for that user.

Of course, all this is not very useful if the system itself is hard to use. Software that protects security and privacy is particularly vulnerable to usability failures, since the benefits of the software are hidden, but the annoyance is not. In addition, the threats themselves

may not be well understood by the users of the system. While a long-term study with a large number of users would be ideal for measuring Doppelganger's usability, due to practical concerns we conducted a controlled study on a more modest scale. Nonetheless, we learned some important lessons: users indeed were able to protect privacy very well with Doppelganger, accepting far fewer persistent and third-party cookies than the default. Most users still found completing the study browsing tasks "easy" or "very easy". Full results are discussed in Chapter 5.

In the next chapter, we will put our work in a broader context by examining related work in more detail.

Chapter 2

Related Work

2.1 Security policy configuration and verification

The security of a computer system is a function of the behavior and interaction of each of its components: typically the hardware, operating system, user-mode programs, and the users themselves. One major difficulty in analyzing these interactions is that the components' behavior generally depends on a variety of configuration settings that can vary from installation to installation. As a result, it is impossible for someone to buy a "secure operating system" or "secure hardware" in isolation, since they may interact in insecure ways, or may be configured in a way that is insecure given the operating environment. The burden of system security is therefore generally the province of the system's operator, who must manage the interactions between complex components just so in order to achieve a secure state.

In our work on system security in this dissertation, we consider a particular subset of this problem: we define an *information-flow integrity* property which considers data channels between user-level programs that interact via a trusted operating system, and implement a system that verifies that property. We say that there is an *information flow* from process A to process B if A can write to an object (e.g., file) which B can read. In other words, A can control the content of an input to B. The mechanism governing this interaction is called an *access control system*, since it decides which processes can access which objects. An *information flow integrity* property typically seeks to control information flows

to a trusted process, to prevent the process' being compromised by an untrusted user.

In order to put our work in context, we will review a few related research directions: formal models of information flow security, including some that are inspired by real-world, domain-specific requirements, and tools that check those security models and other properties. We will also look at policy generation and verification for a different domain—network firewalls—where the literature on actual implementations and practical experience is richer.

2.1.1 Integrity vs. confidentiality

There are broadly speaking two kinds of information-flow security properties: *confidentiality* and *integrity*. To understand these, let us divide the processes running on a system into two exclusive classes: the first includes trusted processes containing sensitive information, and the second includes everything else, typically processes executed by unprivileged users who should not have access to the sensitive information. The former class is often given the label "high" and the latter "low", reflecting their respective degrees of sensitivity.

The confidentiality property is the one that most people think of when security is mentioned. It requires that sensitive data cannot be read by unprivileged principals. Bell and LaPadula [11] formalized this as follows: each *subject* (i.e., user or process) and *object* (e.g., a file) in a system is given a security label from a hierarchy of labels. A typical example of a label set would be `unclassified` $<$ `classified` $<$ `top secret`. Confidentiality is preserved via enforcement of two rules. The *no read-up* rule states that no subject may read an object at a higher security level. For example, a user with a `classified` label may not read a `top secret`-labeled file. The *no write-down* rule states that no subject may write to an object at a lower security level, so for example a `top secret` user may not write to an `unclassified` file. In specifying restrictions between subjects and objects at different levels, Bell-LaPadula is an example of *multi-level security*, or MLS.

The integrity property is the dual of confidentiality; the idea is that if a privileged subject accepts input from an unprivileged one, the former may be used as a proxy to perform operations normally denied to the latter. Sensitive data could be compromised or modified without proper authorization. Biba [12] described exactly this property formally: subjects

are allowed to read up and write down, but not vice-versa. The operating system or other mediation mechanisms can, given the labels, enforce these restrictions without regard to the particulars of the processes themselves. For the remainder of our review, we will focus primarily on work that addresses integrity, rather than confidentiality (though some do both).

Sometimes data will need to cross label boundaries. An *upgrading* operation changes a label from "low" to "high", and *downgrading* does the opposite. An upgrading operation typically follows some verification that the data in question meets a certain format or standard for "high" data; a downgrading operation can be thought of as a declassification of data (or, commonly, a portion or summary of sensitive data).

2.1.2 A note on covert channels

So far, we have discussed security in terms of having explicit access to sensitive data, such as reading or writing it via ordinary system mechanisms. There are, however, other ways to transmit information implicitly or using out-of-band channels; these are typically referred to as *covert channels*. Lampson [60] observed that even when explicit channels are suitably controlled via things like file access permissions, processes with access to sensitive data may communicate information about that data in other ways. For example, the communicating process may use the system in a way that degrades system-wide performance, such as by heavy disk or memory usage; that performance change can be observed by an unprivileged process and by controlling the usage, a communication channel (albeit a low-bit-rate one) may be established.

Goguen and Meseguer [46] introduced a *noninterference* property that captures confidentiality in the presence of covert channels. In short, the property states that untrusted subjects' behavior given the presence of trusted subjects operating on sensitive data is exactly the same as it would be in the absence of those trusted subjects.

Eliminating covert channels is an extremely difficult problem to solve, and we do not attempt to solve it in this dissertation; indeed, in our work on integrity we further assume that trusted processes are written with benign intent.

2.1.3 Formal models of integrity

We have already discussed the Biba model, which provides a foundation for virtually every integrity model to follow. Changes to the Biba model are necessary to have a model that is useful on real-world systems for which Biba's strict no-write-up, no-read-down requirements are unsuitable. In practice, almost no systems can adhere to so strict a requirement.

So far we have modeled processes and objects using strict hierarchies of privilege. Denning [31] generalized this to a lattice structure, reflecting the fact that, for example, there may be parallel hierarchies of trust with a common upper bound (e.g., highest privilege) and a common lower bound (e.g., no privilege). Denning and Denning [32] also noted the need for control-flow dependence analysis in building a sound model. For example, a sensitive value S may never explicitly be leaked via assignment, but a public value P that implicitly depends on it may leak information, via code that looks like (e.g.)

```
if (S)
    P = 1;
else
    P = 0;
```

By knowing the value of P, an attacker would also know the value of S. The authors term this the *confinement problem*.

The Clark-Wilson [25] model takes a step forward from Biba by acknowledging that sometimes subjects at different privilege levels must share data; in particular, sometimes privileged subjects must process data generated by unprivileged ones. Obviously such data could not be used unchecked, and indeed Clark-Wilson requires that all inputs be verified to meet an application-specific invariant. To borrow an example from [6], the invariant for a bank might be that the assets and liabilities books balance. To accept a transaction as input, this invariant must be verified to hold, since the transaction would then enter the trusted portion of the system, where the invariant is always assumed true. Clark-Wilson therefore takes a more holistic view of integrity, imposing requirements not only on the operating system's security policy but also on applications' handling of data.

The Caernarvon model [84] starts with the Biba model, but allows processes to span multiple levels. In particular, it defines special "guard processes," which are allowed

to upgrade certain inputs while retaining the ability to manipulate sensitive data as well. Caernarvon also supports using separate read and write lattices, to allow for different secrecy and integrity policies. The target application domain is smart card operating systems and applications.

2.1.4 From theory to practice

Practical challenges

The downgrading/upgrading problem is identified by Anderson et al. [6] as being a significant obstacle to implementation of Biba and other MLS systems. In short, processes at different security levels often need to share data. Clark-Wilson takes a step in this direction, but is still sufficiently heavyweight that it has not seen widespread implementation. Part of the reason is that, as Landwehr [61] observed, a close match between a model and particular operational or business requirements is very important for successful deployment. It is impractical for users to give up useful features or spend inordinate amounts of time or money on compliance without a clear return on that investment.

Dynamic labeling

A few information-flow frameworks have sought to achieve practicality by removing the need to do static labeling of all subjects and objects. These approaches have the advantage that undesirable flows are still blocked from occurring with lower initial investment, but they are subject to less predictable runtime behavior, i.e., bad flows may not be discovered until they actually occur. Myers and Liskov [69, 70] introduce a "decentralized label model," in which data is labeled to preserve secrecy, but individual applications have absolute discretion to perform declassification (downgrading). This makes it easier to integrate external code into a system. Foley et al. [38] also take a dynamic labeling approach, but their system addresses both upgrading and downgrading, allowing for both secrecy and integrity information-flow property checks. Neither of these systems has been implemented to our knowledge. The LOMAC [39], or "Low Water-Mark Mandatory Access Control," system implements the Biba integrity model in a dynamic fashion. All untrusted data is

labeled “low”, and any process that reads that data is dynamically labeled “low”. A “low” process is not allowed to read any “high” data. LOMAC features a working implementation on both Linux and FreeBSD.

Access Control and Integrity infrastructure

The SELinux system [71], developed by the NSA, is an access control system module for Linux that allows very fine-grained control over security-relevant operations. It seeks to offer *complete mediation* over all such operations; that is, it is possible to control whether each individual operation is allowed or denied. Because of its flexibility and power, SELinux policies can be large and complicated, but it provides a platform on which other, more targeted systems can be layered. Indeed, our work on integrity verification (see Chapter 3) uses SELinux in its implementation.

In order to have a stronger system integrity guarantee, it is important to have trust in the process by which the operating system and a module like SELinux themselves were loaded. A trusted hardware module, such as the IBM 4758 Trusted Platform Module [34], can be used to bootstrap trust through the system startup process. Sailer et al.’s Integrity Measurement Architecture [83] is an attestation mechanism that can be used to provide proof of loaded code and files to a remote party. Jaeger et al. [53] extended this approach to include an attestation of our CW-Lite integrity property (Chapter 3).

Policy-checking tools

Schneider [85] defines the class of policy enforcement mechanisms that work at the operating system or firewall level as “Execution Monitors (EM)”. An EM mechanism can watch some external behavior of a process and restrict it, but does not look inside the program itself. This is as compared with mechanisms that rely on program analysis, e.g. [72, 63]. In our work on CW-Lite, while we consider the effects of internal program behavior by sending additional signals to the operating system, all enforcement takes place at the EM level.

The apol [91] tool allows the user to search for information flows between pairs of subjects in an SELinux policy. The SLAT tool [49] enables further analysis by model

checkers by converting an SELinux policy into a labeled graph. The Gokyo tool of Jaeger et al. [52, 54] allows a deeper analysis, including an all-pairs information-flow analysis which can detect integrity-violating flows. Gokyo is used as part of the CW-Lite verification procedure. Further work [55] shows how Gokyo may be used to help resolve illegal flows that are found.

2.1.5 Specialized models

Two information flow models used for particular domains have garnered attention: the Chinese Wall model of Brewer and Nash [14], used to model the rules governing analysts at firms also providing business financial products; and the BMA (British Medical Association) model, a set of guidelines set forth by Ross Anderson and enacted by Denley and Weston-Smith [30] for restricting access to medical records in a clinical setting. Both models address secrecy and confidentiality.

The Chinese Wall model is meant to separate financial analysts who cover a sector from 1) analysts covering other companies in that sector and 2) the people at the firm selling services to companies in the sector. The interesting twist is the adaptive access control: what the analyst can read depends on what she has already read.

The BMA model covers access to medical records. There are several principals: the patient, the doctors, the specialists, the clerks, etc. This model has a temporal component, for example limiting a ward nurse's read access to the records only of those patients who have been in the ward in the past 30 days. There are also accesses that are contingent on patient or other notification and, in some cases, explicit consent.

2.1.6 Firewall policy management implementations

While our work in this dissertation does not deal with network firewalls *per se*, considerable attention has been given recently to properly configuring and managing them. Their policy languages tend to be somewhat simpler than OS policies, but interactions between firewalls, a large number of site-specific special cases, and their place as the first line of defense can make secure configuration difficult. The reader may find these case studies and

implementations useful as a real-world example of the translation and verification approach to policy.

Wool [96] performs an analysis of configuration errors in real large installations and presents data detailing their respective frequencies, as well other factors that correlate with misconfiguration. Fang [65] is an interactive query tool that takes distributed firewall policies and topology as input and allows the user to see what the rules imply. FIREMAN [100] can perform a static analysis using model checking across a distributed ruleset to discover flaws. Permpoontanalarp and Rujimethabhas [79] take the converse approach, offering a formalism for generation of correct policies. FACE [92] and Firmato [10] can generate distributed rulesets from a central knowledge base and security policy.

2.2 Browser privacy and usability

In this section, we will first explore the history and privacy implications of browser cookies, then look at tools that help users to manage cookies. Next, we will look at relevant previous work on usable security and privacy as well as the economics of security and privacy. The latter will help to motivate our approach in Doppelganger and to explore its implications. Finally, we will look at Recovery-Oriented Computing, whose techniques we use in Doppelganger's error-recovery mechanism.

2.2.1 Cookies and tracking

Cookies are the state-tracking mechanism for the HTTP protocol. A server can send the client a cookie, a small string, which the client will store and in turn send with subsequent requests back to that server. The internet standards for HTTP cookies are in the IETF RFCs 2109 [56] and 2965[57], the latter nominally superseding the former. In fact, much of the latter standard remains unimplemented on a wide scale. Cookies can be used for tracking users, since the server can use them to associate requests to a particular user over long periods of time. Besides this privacy implication, which is the main one which we address in Doppelganger, there are security concerns stemming from cookies. Among these are the ambiguity of the domain-matching algorithm that decides to which servers cookies should

be sent and the use of cookies for security-relevant tasks like authentication. Another standards document, RFC 2964 [67], lays out best practices for cookie use in browsers and by servers. Its recommendation not to use cookies for authentication is widely flouted [41]; nonetheless its recommendations are generally sound. David Kristol's reflections [58] on the development of the cookie standard are interesting and help shed light on the way in which controversial aspects of the standard were decided. A more detailed description of cookie types and implementation can be found in Section 4.2.

Because of the threat of tracking, many users have taken steps to mitigate the threat by disabling or deleting [75] *third-party* cookies, which are those sent by a site other than the primary one being viewed. The most common example of these are cookies sent by advertising images, which are often served by a large third-party advertising firm. Such cookies can be used for cross-domain tracking. In response to third-party cookie blocking, sites have employed two common countermeasures: redirection [81], in which the user is sent through the advertising site on the way to another destination, using HTTP redirect commands; and using IFRAMEs [88], which can act like images visually but in the Firefox browser are treated as being in first-party context, where cookies are typically allowed.

There are other ways to track users besides cookies. A server can use *web bugs*, or invisible images, whose URLs carry an identifier unique to a user. By viewing a page containing the image, the server can know that the user has visited the page. This becomes worse for privacy when the web bug is served by a site that serves such bugs for many other sites. Web bugs in an HTML e-mail can serve as an implicit (and possibly unwanted) confirmation that the message has been read, and from which computer. Bugnosis [5] is a browser plugin that notifies the user of the presence of web bugs, but does not attempt to disable or otherwise mitigate their threat. Disabling cookies makes web bugs less privacy-compromising by making it more difficult to correlate web bugs from different locations.

2.2.2 Existing tools for cookie management

Most of the existing tools for managing cookies are fairly simple, allowing users to view and edit cookies and perhaps enable or disable cookies for the current site. None of these make the decision-making process particularly easier, and keep the focus on the

opaque cookie data items themselves rather than the larger privacy and functionality issues. Cookie Button [26], Cookie Toggle [28], and Permit Cookies [78] add toolbars and enable keyboard shortcuts to help users quickly change cookie policies for the current domain. Add'n'Edit Cookies [4], Cookie Culler [27], and View Cookies [93] add shortcuts to easily view and delete cookies stored for a particular domain.

Millett et al. [66] develop the notion of "informed consent" online, whereby users understand what is being disclosed about them and their actions, and can consent to this disclosure. The authors apply this framework to the cookie interface design in the Netscape and Internet Explorer browsers from 1995 to 2000, generally finding them lacking. Friedman et al. [40] continue with this theme, identifying a specific problem with cookie management: users need awareness but not intrusiveness for a cookie management tool to be useful. The authors implemented a "Cookie-Watcher" plugin for the Mozilla browser [68] that showed the user's cookies to the user along with color coding based on the type of cookie. Cookies could be removed, and there were also help screens to explain cookie concepts. Ultimately, though, the system did not really help users figure out which cookies were useful, and the authors only measured the system on user satisfaction rather than privacy outcomes.

The Acumen [45] system is a bolder step in the right direction. It implements a recommendation system for helping users decide which cookies to accept. Essentially, users can see how many other people accepted a cookie from the current site, and make a decision accordingly. This approach certainly has promise, but suffers from some drawbacks: it does not have personalized recommendations, so although a user can know if a cookie has been accepted by 50% of others, he does know which half of the population he belongs to; it uses a central repository of data, which itself could compromise users' privacy; and such systems are vulnerable to poisoning of the data by sites which inject false positive recommendations for themselves or negative ones for other sites. Acumen gives a higher weight to the opinion of "mavens"; however, mavens are chosen based on the volume, rather than the quality, of their decisions.

2.2.3 P3P

The Platform for Privacy Preferences (P3P) Project [90] 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 [22], not least the difficulty in constructing policies and the lack of any enforcement guarantee. Egelman et al. [35] found that as of 2005, only 13.6% of popular sites had P3P policies.

Privacy Bird [21] attempts to make P3P more useful by showing an icon that indicates the level of privacy protection offered by the site being viewed. Privacy Finder [19] extends this idea by annotating search results to help users decide between search results based on privacy.

There are also products to show and analyze P3P policies. Privacy Fox [8] can parse a P3P policy and present it to the user in a more readable form. Byers et al. [20] and Levy and Gutwin [62] describe tools for automated understand of sites' privacy policies. It is our hope that either P3P or some other privacy standard becomes more widely adopted, so that a cost-benefit analysis can be more accurately performed by users and their browsers.

2.2.4 Privacy and usability

There are two important domains to look at in the area of security and privacy and usability: an economic analysis of people's attitudes towards online security and privacy, and the usability of security and privacy software. Both are relatively nascent fields, for the simple reason that a mass audience did not have to concern itself with these issues until the widespread adoption of web browsers, starting about ten years ago.

Economic analysis of privacy attitudes

There is still no consensus on how people make privacy decisions, partly because it is hard to find a good natural experiment; it is hard to measure privacy in a laboratory setting without using and potentially compromising subjects' personal information.

Gideon et al. [44] conducted a user study with the Privacy Finder system (see above) to measure privacy sensitivity vs. subject matter; subjects did use their own information. Each participant had to purchase two items with his or her own credit card: a power strip and a pack of condoms. Perhaps unsurprisingly, the authors found that people were more likely to prefer privacy-friendly sites for condom purchases than for power-strip purchases.

There are more counterintuitive results, however. Acquisti and Grossklags [1, 2] identified some obstacles to economically efficient outcomes. One that is very relevant to our discussion is the authors' finding that users' lack of information about privacy threats makes it difficult to make a good decision. Furthermore, it is hard to know in advance what the real cost of a privacy violation is going to be.

There are many ways for sites to get users to accept privacy risks, at least in study conditions. If a site agrees to protect personal data from disclosure to third parties, users are more likely to accept the risk [29]. Subjective feelings of trust in the site have also been shown to induce users to accept more privacy costs [23]. Users will also give up privacy for money or other rewards [51]; one may be tempted to put this value on their privacy, but without simulating the privacy-compromised situation it is hard to know the real valuation. So there is a paradoxical situation here: as we discussed earlier, there is a lemons market for privacy [94], but since users don't always know when their privacy is being compromised, and because they (apparently) discount future risks heavily enough to accept compromise for a low price in the present, privacy protections are still not very good, or the result of an efficient market. It is possible that the nonlinear nature of privacy compromise is a factor: independent personal facts may not be too damaging (and thus not valued very highly), but as soon as an adversary gathers enough for identity theft, the damage increases dramatically.

Lessons about the usability of security and privacy software

In the last five years, there has been a sharp increase in usability studies of security and privacy software. Whitten's seminal study [95] on the (un)usability of PGP 5.0 was enlightening: it highlighted the fact that users of security software face not only poor user interface design, like users of all other software; they are also using software where it is not clear if the outcomes they achieve are correct. Smetters and Grinter[89] point out that traditional measures of usability like completion time and user satisfaction are insufficient for security features, which are often a byproduct or tangential to the actual task at hand. At the least, we must see if the security goal was in fact achieved. They go on to suggest that to make progress, user interface advances are unlikely to be good enough unless coupled with underlying changes that make success easier. Kuo et al. [59] had similar criticisms of the application of conventional techniques to usability of security features. In response, they advocate a combination of four approaches: mental model interviews, surveys, "contextual inquiries" (observation of natural work patterns), and usability studies to gauge the ability of participants to complete tasks.

DeWitt and Kuljis [33] found that users were "apathetic" about security and would routinely bypass security mechanisms to get work done faster. 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 [47, 99]. Zurko et al. [102] analyzed user behavior when faced with security decisions in Lotus Notes. They found that users often simply did not understand the implications of security decisions, and would make incorrect choices as a result. Context also matters: when users were told in a one-time email to increase their security by a trusted party, they did so; but when faced with similar questions during ordinary workflow, they would frequently make the opposite choice. Nonetheless, Adams and Sasse argue [3], users can be motivated to care about security and even do a good job of it, if they are educated properly and use security mechanisms that match what they need to do. Both findings accord well with a rational incentive model of security; if users feel like security will have a big payoff, and it is not too disruptive to achieve it, they will take the necessary steps.

2.2.5 Recovery-Oriented Computing (ROC)

The error recovery mechanism in Doppelganger uses concepts from Recovery-Oriented Computing (ROC). Brown and Patterson [15] lay out the principles behind ROC: recognizing that failures, including human ones, are inevitable, and a focus on building mechanisms to detect and recover from those errors. The main principle used in Doppelganger—the rewind, recover, replay idiom—is laid out by Brown and Patterson in [17]. Lastly, a working instance of this model is used [16] in building an "undoable e-mail store", which can roll back and replay transactions as needed to recover from failures such as misconfigurations.

Chapter 3

CW-Lite: Verifying an information-flow property of OS security policies

3.1 Introduction

In this chapter, we will look at a system for verifying a high-level security property, *CW-Lite*, for a low-level operating system security policy. The system is aimed at system administrators, who often need to make fine-grained, site-specific customizations to their security policies but want to ensure that secure information-flow rules are preserved. Before we look at the implementation, we must first understand why this problem exists.

3.1.1 Motivation and Goals

While operating systems provide isolation through separate memory spaces, processes still interact via files, pipes, network connections, shared memory, and other mechanisms. We say that there is an *information flow* between a process A and a process B if A can write to some resource (e.g., a file or pipe) on which B depends. (We do not consider side-channel attacks here.) The *information-flow integrity verification problem* is to prove that a security-critical, or *high integrity*, process does not depend on information flows from untrusted, or *low integrity*, processes.

Let us consider an example. If an untrusted user can write to the trusted OpenSSH con-

figuration file, `sshd_config`, that is a violation of information-flow integrity and a clear security breach. Transitive flows must also be checked: if an untrusted user can run a `cron` job that writes `sshd_config`, there is obviously still an integrity violation. Merely setting file permissions does not prevent attacks that operate via, say, pipes or shared memory: we must consider all kinds of inputs. In general, if a trusted program depends on untrusted inputs, an attacker may be able to gain escalated privilege or compromise the system. To maintain information-flow integrity, a system must be properly *configured*, i.e., its set of permissions must be such that illegal flows from untrusted processes to trusted ones are not possible. That is, we must verify a higher-level integrity property of the configuration rules. Such is the approach taken by the Biba [12] model, where the trusted process is said to depend on a resource merely by reading it. That is, no untrusted inputs were allowed to trusted processes.

This picture is complicated by the fact that many trusted processes must accept some untrusted input to function. We say that each `open()` call (or equivalent, such as `connect()` or `accept`) in the program constitutes an *input interface*, or simply an *interface*. Network daemons must accept some input, such as HTTP requests or session logins, from the network. Input to network interfaces may be controlled by an attacker. The programs running on system must perform sanitization or *filtering* of inputs that come from untrusted sources. By using *filtering interfaces*, the program can read from an untrusted resource, while controlling the extent to which it depends on that resource.

In short, information-flow integrity requires a combination of two elements: (1) proper configuration, which ensures that inputs that a program trusts (like configuration files) cannot be written by untrusted users, and (2) filtering code, which ensures that inputs that a program does not trust (like network input) are checked for well-formedness and application-specific restrictions. Without the ability to communicate with trusted processes except by very narrow interfaces, untrusted users' attack options, and therefore potential exploits, are limited.

The Clark-Wilson integrity model [25] covers both aspects and is a good match for current trusted processes, requiring that all of processes' inputs be filtered or sanitized. However, Clark-Wilson is relatively heavyweight, requiring formal verification for programs. These and other integrity models were developed at a time when deep, complete

program analysis for security was thought to be coming in the near future. That vision has not been realized and, as a result, most systems in widespread use operate without any kind of information-flow integrity verification. Landwehr's 1981 survey [61] of security models noted that many models worked well for a very particular class of applications, but often did not, which easy can lead to (in his words) a "slippery slope" where more and more rules are broken to make systems operational. It is important that whatever model we use be practical not only in terms of meeting application needs, but also in ease of verification. Previous models have not met both simultaneously.

For these reasons, the long-identified information-flow problem is not solved in practice: even though preventing untrusting users from unduly influencing trusted processes is almost universally desirable, and even though system administrators are charged with maintaining this property through a variety of complex mechanisms, for at least thirty years most systems have operated without any assurance that they were configured to meet that integrity goal.

Our aim here is to change this situation; to do so, we define a lighter-weight version of Clark-Wilson integrity, which we term *CW-Lite*, that retains the same interprocess dependency semantics as Clark-Wilson but omits the requirement that programs undergo full formal verification. We then show how a combination of new and existing tools allows practical verification of CW-Lite. These tools address both aspects of integrity verification: they help administrators to find and fix configuration errors, and application developers to find and annotate interfaces that require input sanitization. Verifying the CW-Lite property on a system is largely automated: administrators only need to make manual decisions when violations are found, and developers must only annotate untrusted input interfaces, which are identified with our tools, with a simple macro. Naturally, developers must implement filtering code on all reads from these interfaces in any case; the annotation serves to allow static verification of CW-Lite.

Rather than have a tool that simply says that a system has violations, we have tried where possible to make resolving the problems easier as well. We demonstrate the effectiveness of our tools—and thus, the feasibility of achieving Clark-Wilson-style information-flow guarantees—by applying them to privilege-separated OpenSSH, which interacts with many system objects, and has the challenge of containing trusted and untrusted compo-

nents within one application. We also analyze vsftpd to illustrate the general applicability of our approach. We found several security policy configuration errors that permitted unnecessary, possibly insecure flows. We also determined that certain other programs, such as `rlogind` and `xdm`, caused insecure flows, and should not be run on systems that desire information-flow integrity guarantees. Indeed, one of the benefits of checking information flows on a system is that it makes the formerly implicit TCB of the system explicit, and highlights programs whose presence on the system can cause insecure flows. Although not every insecure information flow leads to an exploitable hole, by eliminating all such flows, we eliminate all related exploits as well.

3.1.2 Contributions

In this work, we make the following contributions:

- We develop an information-flow integrity property, CW-Lite, which captures the interprocess dependency semantics of Clark-Wilson integrity, but is verifiable on real systems using tools and only a modest amount of manual effort;
- We develop a suite of tools as well as modifications to SELinux to support CW-Lite enforcement;
- We apply our approach to OpenSSH and vsftpd, and find several integrity-violating permissions in their default SELinux policies;
- In short, we have demonstrated practical verification of Clark-Wilson interprocess information-flow integrity.

3.1.3 Roadmap

Section 3.2 contains a high-level overview of the CW-Lite model and its verification process. In Section 3.3, we define CW-Lite formally, starting with the Clark-Wilson model and weakening certain requirements. Section 3.4 describes our system modifications and the algorithm used to verify the CW-Lite of a trusted application. In Section 3.5 we apply our approach to OpenSSH and vsftpd on Linux with SELinux, describing how we used our

tools to (1) identify filtering interfaces necessary to handle low integrity inputs; and (2) resolve potentially harmful information flows that would not be filtered. We discuss related work in Section 3.6 and summarize our findings and future work in Section 3.7.

3.2 Overview of CW-Lite and its Verification

There are two motivating observations behind CW-Lite. The first is that because the Clark-Wilson model contains a formal verification requirement in addition to an interprocess data flow model, it has proven too heavyweight for widespread use. However, the two goals are separable, and we may profitably try to solve the latter goal independently. The second observation is that Clark-Wilson requires filtering of all interfaces, but most trusted programs only need to open untrusted interfaces at a small number of locations. Since interfaces that read trusted inputs do not need to be filtered, this can lead to a lot unnecessary work; deciding which inputs only take trusted inputs, however, cannot be done in a vacuum since it requires knowledge of the system's security policy.

The first observation led us to focus our work here on a concrete solution to the first Clark-Wilson goal: securing interprocess information flows in an application-independent way. Accordingly, CW-Lite duplicates the interprocess information-flow semantics of Clark-Wilson. Full formal verification of the programs themselves is a separate and difficult problem; also, verifying semantic correctness is an application-specific task. Obviously this kind of verification is very useful and can prevent other kinds of integrity compromises (e.g., those resulting from buffer overflows), but we believe that separating the two problems will allow simpler, more flexible solutions to each.

Verifying CW-Lite means ensuring that no unfiltered information flows exist from untrusted processes to trusted ones. To do this, we must first identify all possible interprocess information flows. We do so by using a mandatory access-control (MAC) system that interposes access checks on all interprocess flows; with a fine-grained MAC, we can make meaningful statements about which flows are and are not possible. In this paper, we use the SELinux [71] module for Linux, a fine-grained MAC system for Linux that implements Role-Based Access Control with Type Enforcement. SELinux is now a standard part of

Fedora Core Linux and is being integrated into many other Linux distributions.

Our second observation led us to extend the MAC system, so developers would not have to filter on trusted inputs. In order to enforce least privilege for both trusted (non-filtering) interfaces and untrusted (filtering) interfaces in one process, the MAC must distinguish between the two kinds of interfaces, allowing only trusted flows to trusted interfaces but allowing additional, untrusted, inputs to the others. By modifying SELinux to associate two security contexts, or *subject types*, with each process instead of one, we can allow only inputs from trusted processes by default, while enabling a special context, a *filtering subject type*, for filtering interfaces that allows necessary kinds of untrusted inputs as well. This separation reduces the burden on the developer relative to Clark-Wilson by requiring filtering only of untrusted inputs, which they must do in any case. The only requirements are that developer annotate filtering interfaces with a simple macro and put relevant permissions in the filtering subject type. The annotate serves as a stipulation by the developer that all reads from the interfaces are properly filtered. To facilitate this process, we also developed a tool that enables developers to identify which inputs should be annotated filtering interfaces, based on the extra permissions they need. These small changes to the development process and SELinux are also what enable automatic verification by administrators on end systems.

Now that we have isolated the trusted interfaces into a separate subject type, we need a way for administrators to detect illegal flows to it from untrusted sources, i.e., verify the CW-Lite property on their systems. In previous work, Jaeger et al. developed the Gokyo tool [52, 54], which can determine information flows from an SELinux policy by looking at read-type and write-type permissions, then flag illegal ones when supplied with the system's TCB (assuming other kinds of processes are untrusted). We leverage Gokyo here by having it ignore flows to the filtering subject type for the target application, reporting information-flow violations only for the base subject type.

3.3 CW-Lite

In this section, we state the CW-Lite model more formally. As we noted previously, CW-Lite is a weakened version of the Clark-Wilson [25] integrity model, but the focus is on controlling interprocess information flows, rather than formal verification of the programs themselves (which is a separate, important problem). In particular, we do not discuss the application-specific task of verifying the semantic correctness of filtering interfaces in this paper. That is, we seek to provide assurance that filtering code has not been omitted, but not assurance of that code's semantic correctness.

3.3.1 Basic Information-Flow Integrity

In basic information-flow integrity models, dependence on low integrity data is defined in terms of information flows. Such models require that no low integrity information flows may be input to a high integrity subject.

We start with a definition of information flow based on two standard operators, *modify* and *observe* where: (1) $mod(s, o)$ is the modify operator where a *subject* (e.g., a process or user) with subject label s writes to an *object* (e.g., a file or socket) with object label o and (2) $obs(s, o)$ is the observe operator where a subject of subject label s reads from an entity of object label o. Additionally, we abuse mod to allow $mod(s1, s2)$ where both $s1$ and $s2$ are subjects, to indicate when one process modifies another directly, for example using signals.

Definition 1 (Intransitive information flow) $flow(s1, s2)$ *holds if information flows from subject* $s1$ *to subject* $s2$ *in one step.*

$$flow(s1, s2) := (\exists o : mod(s1, o) \wedge obs(s2, o)) \vee mod(s1, s2)$$

Next, the operator $int(x)$ defines the integrity level of x where x may be either a subject or an object. In information flow integrity models, integrity levels are related by a lattice [31] where $int(x) > int(y)$ means that y may depend on x, but not vice versa. For our purposes, this means that trusted processes may not depend on untrusted ones. We

assume in our discussion that $int(x)$ is fixed for all x, but as a practical matter, the system can be re-verified if this changes (for example, if a program is no longer trusted).

Definition 2 (Biba integrity) *Biba integrity [12] is preserved for a subject s if (1) all high integrity objects meet integrity requirements initially and (2) all information flows to s are from subjects of equal or higher integrity:*

$$\forall s_i \in S, flow(s_i, s) \Rightarrow (int(s_i) \geq int(s)).$$

where S is the set of all subjects.

Some information-flow based integrity models, such as LOMAC [39], operate differently but have the same information-flow integrity semantics as Biba. LOMAC allows subjects to go from a higher to a lower integrity level if they read untrusted data, which preserves the Biba definition in the future. It does not, however, deal with programs that must read trusted data again in the future.

A note on transitivity. We note that while information flow is transitive in general, only intransitive information flows need to be examined to detect a Biba integrity violation. Suppose that A and B are untrusted and X and Y are trusted. If we have a transitive information flow from $A \rightarrow B \rightarrow X \rightarrow Y$, only the flow from B to X is needed to trigger a Biba integrity violation, i.e., there is always some flow that crosses the boundary between untrusted and trusted. It does not impact Biba integrity further that information can flow from A to B. While we find Biba too restrictive, we want to preserve the need only to check flows independently.

3.3.2 Clark-Wilson Integrity

The Clark-Wilson integrity model [25] provides a different view of dependence. Security-critical processes may accept low integrity information flows (*unconstrained data items or UDIs*), but the program must either discard or upgrade all the low integrity data from all input interfaces. The key to eliminating dependence on low integrity information flows is the presence of *filtering interfaces* that implement the discarding or upgrading of low integrity

data. The Clark-Wilson integrity model does not distinguish among program interfaces, but treats the entire security-critical program as a highly assured black box. As a result, all interfaces must be filtering interfaces.

In the original Clark-Wilson model, trusted processes are known as *transformation procedures (TPs)*, typically operate on CDIs (inputs that are trusted to meet application-specific invariants), but may also accept UDIs if the TP is known to filter all its inputs. Thus, our notion of trusted applications maps closely to Clark-Wilson's transformation procedures. Clark-Wilson also defined special trusted processes, called *integrity verification procedures (IVPs)*, that check the integrity of CDIs by performing appropriate integrity checks on each data item. These are used to establish system-wide integrity at the start of operation; we do not specifically consider such programs in our model.

We now define information flow in terms of a connection between subject labels and their program interfaces. For this we need a more precise obs operator: $obs(s, I, o)$ means that the subject s reads an object of type o on interface I. An interface for a subject is a distinct input information channel, and is created by, e.g., a particular `open()` call in a program.

Definition 3 (Interface information flow) *$flow(s_i, s, I)$ holds if information flows in one step from subject s_i to subject s through an interface I in a program running as subject s.*

$$flow(s_i, s, I) := (mod(s_i, o) \wedge obs(s, I, o)) \wedge obs(s, I, s_i)$$

We also define the predicate $filter(s, I)$ to mean that a subject s filters or sanitizes input on an interface I, such that any required invariants are satisfied.

We are now ready to state the Clark-Wilson property.

Definition 4 (Clark-Wilson integrity) *Clark-Wilson integrity is preserved for a subject s if (1) all high integrity objects meet integrity requirements initially (i.e., the high-integrity invariants are met initially); (2) the behavior of all programs that are loaded using s are assured to be correct; and (3) all interfaces filter (i.e., upgrade or discard) low integrity information flows:*

$$\forall s_i \in S, flow(s_i, s, I) \Rightarrow filter(s, I),$$

where S is the set of all subjects.

While the Clark-Wilson model does not require separate multi-level secure processes for upgrading, as does Biba, it requires a significant assurance effort. An important point to note is that since a Clark-Wilson application programmer does not know the system's information flows in advance, Clark-Wilson requires that all interfaces be filtering interfaces. This is unnecessarily restrictive. In practice, often only a small number of interfaces actually need to capable of filtering in the context of a real system to achieve the same security. This set can be derived by analyzing the system's security policy; that is, by using system knowledge in application development (since the developer can ship a security policy with the application), we can reduce the filtering burden on the developer. We use this observation in developing CW-Lite.

3.3.3 CW-Lite

Definition 5 (CW-Lite) *CW-Lite is preserved for a subject s if: (1) all high integrity objects meet integrity requirements initially; (2) all trusted code is identifiable as high integrity (e.g., from its hash value as for NGSCB [36]); and (3) all information flows are from subjects of equal or higher integrity unless they are filtered:*

$$\forall s, s_i \in S : (\exists I : flow(s_i, s, I) \wedge \neg filter(s, I)) \Rightarrow (int(s_i) \geq int(s))$$

, where S is again the set of all subjects.

That is, CW-Lite requires that the application's information flows either adhere to Biba integrity or that untrusted (low-integrity) inputs are handled by a filtering interface. Note that this provides equivalent integrity to Clark-Wilson, since the only flows not being filtered come from trusted sources. Recall that CW-Lite also does not require formal verification of filtering interfaces; it simply requires the developer to mark filtering interfaces as such so they can be handled correctly by the MAC system.

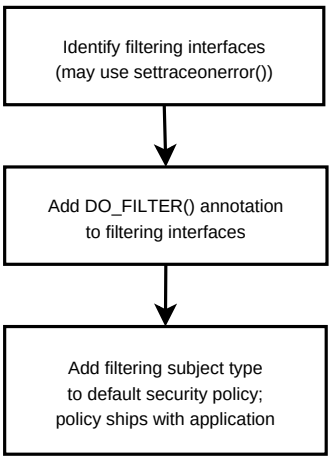


Figure 3.1: **Application developer tasks required to enable CW-Lite verification.** Filtering interfaces are those that accept inputs from untrusted sources, and must sanitize, or *filter* the input. An interface is marked by a distinct call to `open()`, `accept()`, or other call that enables data input. The `DO_FILTER()` annotation on an interface tells the access-control system to grant additional permissions allowed by the filtering subject type to that interface. The default subject type, used on all other input interfaces, only allows inputs from the the system TCB.

3.4 Developing CW-Lite-Compliant Systems

In this section, we tackle the CW-Lite tasks implied by the previous section. Recall that the CW-Lite property is one that is verified for a particular target application running on a particular system. Application developers must enable verification with small changes to their programs and security policies, while the administrators perform the actual verification on their systems. (See Figures 3.1 and 3.2 for flowcharts of these tasks.) For our discussion, we use the term *TCB* (trusted computing base) to indicate the set of subjects that must be trusted on the system in order to trust the set of target applications (e.g., `sshd`, Apache, bind).

An application developer must:

1. Assuming some TCB and application configuration, identify untrusted inputs to the program and implement filtering interfaces for each. This may be done using the process in Section 3.4.4.

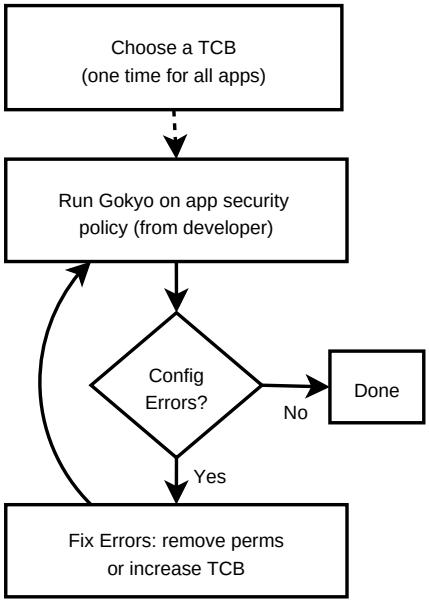


Figure 3.2: **System Administrator tasks required to verify CW-Lite.** The system administrator decides on a system TCB initially. Then, when she wants to verify CW-Lite for a particular trusted application, she runs Gokyo on its security policy. If no errors are reported, CW-Lite integrity is verified. If it reports an illegal flows, the offending permissions must be removed, or the TCB expanded to include the source of the illegal flows.

2. Annotate those interfaces with the `DO_FILTER()` annotation. These annotations are used by the access control system as described in Section 3.4.2.

3. (Possibly in conjunction with a distribution maintainer:) Construct a default security policy for the application that has two subject types: the default only allows inputs from the TCB, and the other, for filtering interfaces, allows required types of untrusted inputs as well.

Since application developers should be sanitizing their untrusted inputs anyway, this represents only a small amount of additional work to enable system-level integrity verification.

A system administrator must:

- One time only, choose a system TCB. (A TCB may be chosen per-application for a multilevel trust model, but this is not necessary or common. In this scenario, each target application would be associated with only the set of subjects on which it depended, independent of other applications.)

- Run the security policy analysis tool for the target application as described in Section 3.4.3.

- If no integrity-violating permissions are detected, then skip the next step.

- Classify each integrity-violating permission found by the tool to decide how to remove the illegal flow. See Section 3.4.5 for details on how to do this.

Note that verifying the CW-Lite property is done automatically using Gokyo; it is only resolution of problems that requires manual intervention. In addition, our approach allows each sysadmin to decide which applications trust on her system. She can evaluate the risk of running a particular application in terms of what must be trusted in order to run it.

In the remainder of this section, we will first describe the SELinux access control system, then show how we modified SELinux to support filtering interfaces and, therefore, CW-Lite verification. We continue by addressing the developer and sysadmin tasks above, including performing policy analysis and finding filtering interfaces.

3.4.1 SELinux

The SELinux module [71] is a Linux Security Module (LSM) [98] that provides fine-grained, comprehensive MAC enforcement. It ships standard with Fedora Linux, among others, and it is quickly becoming standard to include attendant SELinux policies with applications. SELinux implements an extended form of Type Enforcement (TE) [13] with domain transitions that enables expression of policies covering over 30 different kinds of objects with about 10 operations each. SELinux is comprehensive because it aims to control all programs' accesses to all security-relevant system objects. In this paper, we do not examine verifying that the SELinux/LSM implementation is a correct reference monitor. Previous work verified the LSM reference monitor interface [101], but verifying the correctness of the SELinux implementation properties remains.

Key notions in SELinux are those of *subject types* and *object types*. A process' security context is determined by its subject type, much as the security context of an ordinary UNIX process is determined by its effective UID. Likewise, non-process objects like files are associated with an object type. Permissions are attached to a subject type in policy files; if an Apache process has the subject type `apache_t`, and its configuration file has object type `apache_config_t`, we might say something like

```
allow apache_t apache_config_t:file
   {stat read}
```

to allow Apache to call `stat()` on or read from its configuration file. SELinux does not include a "deny" operation; all permissions are denied by default.

Although there are several access control concepts in the SELinux policy model besides `allow` permissions by subjects on objects, only one other is relevant to information flow. The `relabel` operations[1] enable a subject to change the object label of an object. This enables information flow from the old object label to the new one.

While it allows us great control and flexibility, such fine-grained, comprehensive control results in very large and complex access control policies. Frank Mayer describes the SELinux policy model as an "assembler level" policy. In the August 19, 2004 release, the default build results in a 500 KB compiled policy file. There are over 5,000 permission

[1] A subject needs the *relabelfrom* and *relabelto* permissions to implement a relabel.

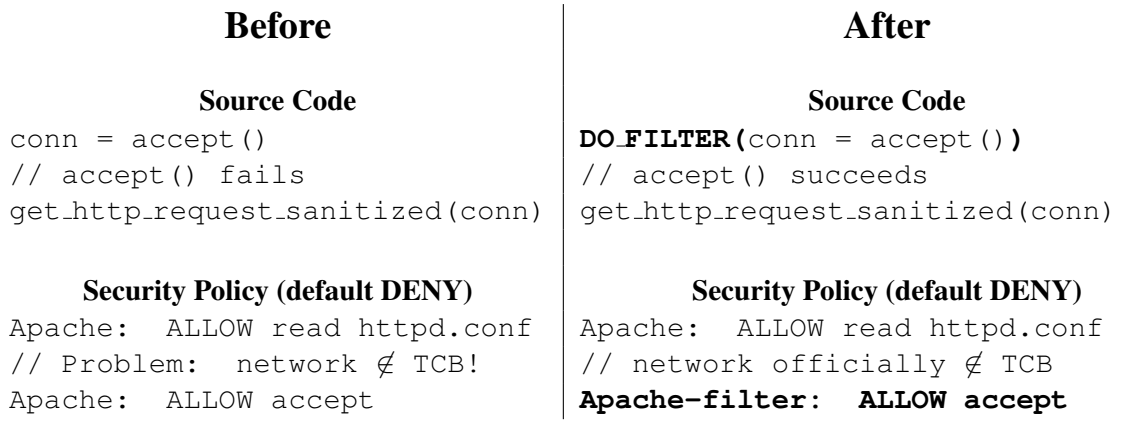


Figure 3.3: **Supporting filtering interfaces.** Initially, the program above is not allowed to accept network input, because the network is not in the TCB. In order to accept such input, the source code must filter it and the programmer must supply an annotation indicating that the interface is filtered. Then the policy must be modified to allow the network input only for the filtering interface. The `DO_FILTER()` macro tells the MAC system to use the filtering subject type permissions for the enclosed operations. We annotate `accept()` (which implies a read/write socket), rather than subsequent socket read/write operations, because that is where the MAC system performs access checks. This is analogous to how file access checks, including read/write permission checks, are performed once on `open()`, not for every `read()` or `write()` call.

assignment (allow) rules in the policy itself (in the file policy.conf). Note that this policy contains just the base subjects; the complete policy, including policies for all shipping applications, is about ten times greater in size. As a result, understanding the higher-level properties that a policy implies, such as information flow, cannot be done manually.

3.4.2 Supporting filtering interfaces in the MAC Policy

SELinux cannot distinguish among input interfaces in a single process. Some interfaces may only have to process high integrity data, such as the interface that reads a configuration file. Others have to be able to validate and upgrade certain types of low-integrity data such as network input: these are filtering interfaces. In order to support filtering interfaces (and therefore to check CW-Lite), we modified the SELinux user space library and kernel module to support two subject types per process instead of one. The default subject type is used for ordinary operation and allows inputs only from subjects in the application's TCB; this is

enforced by the Gokyo policy analysis in Section 3.4.3. The new *filtering subject type*, with additional permissions, is used for interfaces with the appropriate `DO_FILTER()` source code annotation.

```
DO_FILTER(interface creation code) :=
  use_filtering_subject_type();
  interface creation code
  use_default_subject_type();
```

The annotation serves as a contract with the programmer, who stipulates that input from the interface is filtered. Our macro-like approach is deliberate, to discourage running a large amount of code with higher privilege. Typically, only a single `open()`-type call requires the permissions. An example of the required changes to the program and the security policy for filtering interfaces is given in Figure 3.4.2.

Note that the `accept()` system call is still constrained by the MAC policy for the filtering subject type. For example, the filtering subject type permissions for the application might allow accepting connections from one network card, but not another.

3.4.3 MAC Policy Analysis

Once the target application's untrusted inputs have been isolated into its filtering subject types, we need only check that there are no untrusted inputs to the application's default subject type s.

We employ the Gokyo tool to compute information flows from an SELinux policy [52]. Gokyo represents access control policies as graphs where the nodes are the SELinux subject types and permissions, and the edges are assignments of permissions to subject types. Based on whether the permission allows a mod operation, an obs operation, or both, Gokyo computes all information flows to s. That is, it computes the set of subject types

$$F(s) = \{s' : \exists o : mod(s', o) \wedge obs(s, o)\},$$

where o is an object type. Gokyo also correctly handles flows implied by object relabeling; see Figure 3.4 for details.

Because the SELinux policy model also permits object relabeling, we must consider information flows caused by modifying an object and relabeling it to another object type. The $relabel(s, obj, o, o')$ operation enables subject s to change an object obj's label from o to o'. Since relabeling does not change the contents of an object, we do not really care who does the relabel, just that it can occur. Also, it does not matter which specific object can be relabeled, since all objects of the same object type are equivalent from an information flow perspective. Thus, we use a refined predicate $relabel(o, o')$.
Next, we consider successive relabeling operations $o_1 \rightarrow o_2 \rightarrow ... \rightarrow o_i$. The transitive closure of the $relabel$ operation is defined by $\overline{relabel}(o_1, o_i)$. The ***relabel information flow rule*** states that

$$mod(s_1, o_1) \wedge \overline{relabel}(o_1, o_i) \wedge obs(s_i, I, o_i) \Rightarrow flow(s_1, s_i, I).$$

Gokyo accounts for information flows due to arbitrary relabeling.

Figure 3.4: **Gokyo support for SELinux object relabeling**

Some of the non-target subjects may be designated as *trusted subjects*, and they form the system's TCB. The TCB includes subjects such as those that bootstrap the system (e.g., *kernel* and *init*), define the MAC policy (e.g., *load_policy*), and do administration (e.g., *sysadm*).

Given the set of information flows and the TCB, the untrusted subjects with flows to s are given by

$$U = F(s) \cap \neg TCB.$$

If this set is empty, then CW-Lite holds for the target application. If not, Gokyo outputs the set of permission assignments P that need to be examined, i.e., those that allow the offending mod and obs operations:

$$P(s) = \{p : \exists u \in U, \exists o : flow(u, s) \wedge (p \Rightarrow mod(u, o) \vee p \Rightarrow obs(s, o))\}$$

where p is a permission assignment, $u \in U$, and o is some object type.

3.4.4 Finding filtering interfaces

Although we modified SELinux to support mediation for filtering interfaces separately from other interfaces (Section 3.4.2, above), the developer still needs to make annotations to tell SELinux whether a given interface performs filtering or not. As part of this process, the developer needs to determine which interfaces require filtering. Some may be obvious, but there may be permissions to access untrusted data that are used in a subtle way. The developer can find these by running the security policy analysis on the default policy and analyzing all integrity-violating permissions for the application.

The problem of determining where in a program a permission is used is outside the scope of SELinux's goals, so we implemented our own mechanism. We defined a new operation in SELinux called `settraceonerror` using the *sysfs* interface and made appropriate changes to both SELinux's user library and its kernel module. When `settraceonerror(true)` is called from user space, our modified SELinux kernel module signals the process whenever a violation of the SELinux policy is found. The user library catches the signal and traps the process into a separate xterm debugger (gdb). If the process forks, additional

xterm windows with debuggers on the child processes are launched. Once in the debugger, it is much easier, using stack traces and data inspection, to determine where and why a permission error occurred and take appropriate action, either removing the offending operation or implementing a filtering interface. If the permission is never actually needed, then it can simply be removed from the policy.

Some filtering interfaces may not need to actually filter the incoming data contents, since some interfaces do not interpret the incoming data. For example, *logrotate* enables automatic rotation of log files, but does not depend on the data in the files. Likewise, the *cp* utility copies files, but does not consider their contents. In these cases, a `DO_FILTER()` annotation is still appropriate, rather than allowing the program to accept all inputs. This is because (1) filtering based on metainformation (like input length) may still be needed; and (2) the kinds of inputs may need to be restricted (for example, disallowing copies from named pipes). Naturally, if the program semantics change later to include interpretation of the untrusted data, the programmer should implement additional filtering code.

One may wonder why we use a dynamic approach to finding filtering interfaces. A simple example is revealing: consider the interface `fd = open(filename)`. In order to decide statically if filtering is required, we would need to know the value of `filename`. This may be partially addressed with a program analysis, though of course it is undecidable and may come from dynamic data, say from the system's configuration. The mapping of filename to object type (which is what matters for integrity) is also system dependent, and each administrator may keep files in different locations. Our approach works well enough in practice, since the number of filtering interfaces is usually relatively small; while it may have the coverage problem of dynamic analysis, it does not have the scalability and decidability problems of static analysis.

3.4.5 Handling illegal information flows

If a sysadmin's invocation of the policy analysis tool detects illegal information flows implied by a set of permissions, one of a few actions is required. Some such permissions are simply unneeded and may be removed. Some information flows may be generated by programs that are untrusted, but optional to the system. An easy way to remove this in-

formation flow is to exclude the offending code and subject types from the system. Some permissions are needed by optional components of the target app; the options may be disabled, and the permissions removed. If the permission is used by the core application, then either the sysadmin may be assuming a smaller TCB than the developer or the developer has not added a `DO_FILTER()` annotation. The sysadmin can either not run the target application or get the developer to write and annotate additional filtering interfaces.

3.5 Example: CW-Lite Integrity Verification

3.5.1 Goal

So far, we have defined CW-Lite and shown in general how applications can be constructed to satisfy its requirements. We have several goals in applying CW-Lite to our primary example application, OpenSSH, and to vsftpd. First, we want to see how easy it is to build applications to meet CW-Lite in practice. Since OpenSSH is a very popular, complex, and security-critical program that has been architected to preserve the integrity of its privileged components, verifying a useful integrity property can be of value to of millions of systems and validate the security efforts of its developers. vsftpd is a somewhat simpler example, and illustrative of a common case. Second, we want to see how close the default SELinux policy is to enabling satisfaction of CW-Lite. Third, if either application (with the standard shipping policy) does not initially meet CW-Lite integrity, we want to see why it fails and how difficult it is modify the application or policy to enable success.

3.5.2 Setup

Provos *et al.* decomposed the server-side daemon of OpenSSH into privileged and unprivileged components in order to minimize the amount of code that needs to run with privilege. The privileged component exports a narrow interface to the unprivileged components, such that only specific operations in a specific order may be requested, which reduces the risk of the privileged component being compromised by a hijacked unprivileged component. Privilege-separation has been added as an option to the main OpenSSH

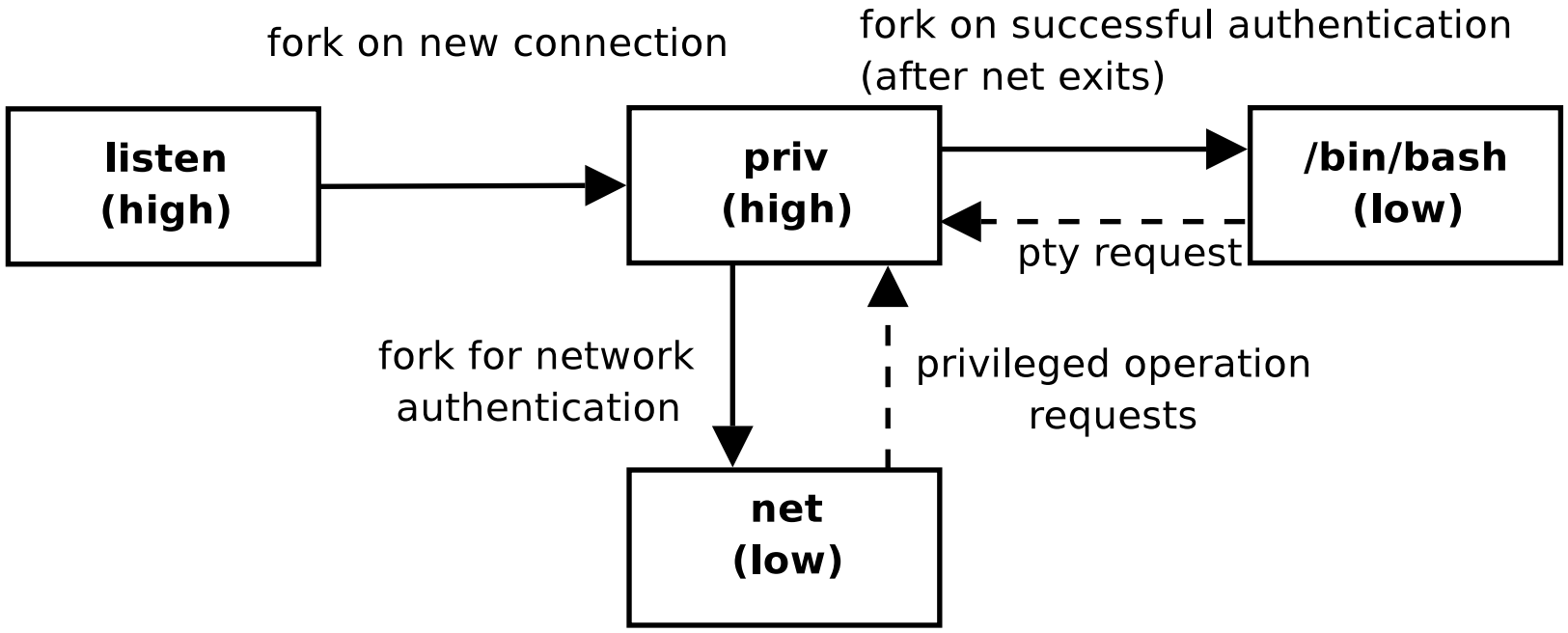


Figure 3.5: **Process structure of privilege separated OpenSSH.** The *listen* process simply processes new connection requests, forking a *priv* process to handle each one. The *priv* process forks a *net* process to perform the network portion of user authentication, providing a narrow interface to privileged operations necessary to complete authentication. After *net* completes its task, *priv* spawns the authenticated user's requested process, in our example a bash shell.

distribution.

The process graph for privilege-separated OpenSSH is shown in Figure 3.5. One privileged component, *listen*, listens for new connections via port 22 and forks a new privileged component, *priv*, per connection. This *priv* component performs the privileged operations required by OpenSSH: authentication of the remote user, creation of pseudo-terminals, and transition to a particular, authenticated userid. The *priv* component in turn spawns unprivileged components to handle various types of user interaction. The *net* component is used to perform the remote interaction part of the authentication phase, which has in the past been subject to compromise; it uses the *priv* component as a privileged server to handle secret data operations. After successful authentication, *priv* spawns a shell or other process requested by the user in that user's security context.

vsftpd is the FTP server included with Fedora Core Linux. It too employs separate trusted and untrusted processes, though its policy treats both the same. We do not discuss its analysis in as much detail, but give a summary of the analysis and the results.

For testing, we used OpenSSH 3.6 and vsftpd 2.1.3 on an Intel x86 platform with the Linux 2.6 kernel installed. We use SELinux (see Section 3.4.1) as our MAC system, using the strict (not targeted) policy configuration for Fedora Core 4.

3.5.3 Roadmap for OpenSSH

The problem of verifying CW-Lite for privilege-separated OpenSSH is addressed by ensuring that all information flows into the privileged components (*listen* or *priv*) either contain only high integrity data or discard/upgrade the data via declared filtering interfaces.

Enabling OpenSSH to satisfy CW-Lite requires work by the application developer to modify OpenSSH to find where filtering interfaces are necessary, build acceptable filtering interfaces, and declare the presence of the filtering interfaces to SELinux. Then, the administrator of the SELinux system needs to configure an SELinux policy that enables satisfaction of CW-Lite. Recall that this policy will have a base set of permissions (subject type) allowing only trusted input for normal interfaces and additional subject type to accept requires types of untrusted input at the filtering interfaces.

The first step is to use the Gokyo policy analysis tool to identify the illegal informa-

tion flows to *priv* and *listen*. The next task is to determine whether the remaining low integrity flows can be handled by filtering interfaces. We use our tools (see Section 3.4.4) to find the interfaces that accept low integrity data in the privileged components and add the `DO_FILTER()` annotation.

3.5.4 Inter-process Flow Analysis

Given the new SELinux policy for the OpenSSH components and the remainder of the SELinux example policy for the rest of the system, we are ready to use Gokyo to find low integrity information flows to the privileged OpenSSH components *priv* and *listen* and revise the policy to remove any unnecessary flows. Gokyo computes the information flows in the SELinux policy that violate the policy analysis constraints given the sets of trusted subjects, excluded subjects, and filter rules. A short introduction to Gokyo is in Section 3.4.3.

We define a TCB including the system bootstrap components, such as *bootloader*, *kernel* and *init*, and components that modify the SELinux policy itself (e.g., *checkpolicy*, *load_policy*, *setfiles*, etc.) or other objects upon which the system integrity depends (e.g., administrative subjects *sysadm*, *staff*, *rpm*, etc.).

We then run Gokyo and identify several information flow conflicts shown in Table 3.1. The table shows each instance where the target subjects (*priv* and/or *listen*) have a read permission on an object that may be modified by untrusted source subjects (*write-up subjects*); some objects may be written to by many write-up subjects. The problem then is to find a resolution that prevents the target subjects from being dependent on the write-up subjects. For each entry, one of these resolutions is applied in the following order of precedence: (1) we can EXCLUDE the write-up subject from the system if it is not required on the system; (2) we can identify that the permission does not actually result in a data dependency (FILTER_NO_DEP) (for example, the `cat` program, which simply passes data through), and so requires only a lightweight filtering interface that prevents only metainformation attacks like buffer overflows; (3) we can FILTER the use of the permission via a filtering interface; or (4) we can REMOVE the permission assignment from the target subject or the write-up subject if not required by the subject.

Target Subject	**Permission** (object:class)	**Source Subjects** (names or count)	**Resolution**
Resolutions requiring primarily system knowledge			
priv, listen, ftpd	devlog:sock	privlog	FILTER_NO_DEP
priv	lastlog:file	6	FILTER_NO_DEP
priv, ftpd	etc_runtime:file	xdm, hotplug	EXCLUDE
listen	initrc_var_run:file	7 (includes rlogind)	EXCLUDE
listen, ftpd	net_conf:file	dhcpc	EXCLUDE
priv, ftpd	wtmp:file	7 (includes rlogin)	EXCLUDE
Resolutions requiring application knowledge			
listen	sshd_listen:tcp (`accept()`)	[network]	FILTER
listen	userpty:chr_file	7	FILTER
priv	sshd_priv:unix	sshd_net	FILTER
ftpd	ftp_port_t:tcp (`accept()`)	[network]	FILTER
listen	sshd_listen:tcp (`read()`)	[network]	REMOVE
priv	xserver_port:tcp	165	REMOVE
priv, listen	devtty:chr_file	200	REMOVE (Used for walk-through.)
priv,listen	port_type:tcp	[network]	REMOVE
listen	sshd_listen:unix	unpriv_userdomain	REMOVE
listen	sshd_listen_devpts:chr_file	5	REMOVE
priv	sshd_tmp:file	(staff/sysadm/user)ssh	REMOVE
priv	sshd_tmp:lnk	(staff/sysadm/user)ssh	REMOVE
priv	sshd_tmp:sock	(staff/sysadm/user)ssh	REMOVE
priv	sshd_tmp:fifo	(staff/sysadm/user)ssh	REMOVE
priv	system_chkpwd:fd	27	REMOVE
priv, listen	unpriv_domain:fd	33	REMOVE
ftpd	ftp_port_t:tcp (`read()`)	[network]	REMOVE*
ftpd	nfs_t/cifs_t:file	27	REMOVE*
ftpd	user_home:file	4	REMOVE*

FILTER = The permission is necessary, but requires a filtering interface. It should be put in the filtering subject type.
FILTER_NO_DEP = Similar to FILTER, except that the filtering code need only check metaproperties of the input because the program does not interpret the data. Since the data merely passes through, there is no *dependence* on the data.
EXCLUDE = Exclude the source subject from the SELinux policy, as it causes insecure flows; any associated programs must not be run on the system. Subjects in this group must either be removed or added to the trusted set, which eliminates the illegal flow by definition.
REMOVE = Remove the untrusted input-enabling permission from the target subject, breaking the information flow.
* = The vsftpd policy did not fully reflect its process structure; see Section 3.5.6 for details.

Walkthrough for shaded row: "priv, listen" indicates that the illegal flows were inputs to both the *priv* and *listen* components of OpenSSH. The object that they have permission to read from is devtty:chr_file, that is, a TTY from `/dev/tty`. Two hundred untrusted subjects have permission to write to that object. The illegal flows are broken by removing the read permissions, since they are not necessary: the TTY is actually read only by the *net* component, which handles remote user input.

Table 3.1: **Information flows to our target subjects (*priv* and *listen* for OpenSSH and *ftpd* for vsftpd) that may lead to integrity problems.** The permissions leading to these flows were identified by the Gokyo tool. The top half of the table indicates conflicts resolved based on system knowledge. The bottom half required examining the behavior of the target application using the tools described in Section 3.4.4. Each target subject was analyzed independently.

The table groups the resolutions into two categories: (1) resolutions based on information flow only and (2) resolutions based on application configuration and information flow. The decision between removal of permission assignments and filtering generally requires application knowledge; some permissions are needed to support optional components of the application and some are needed for core operation. An administrator would need to decide which options were required on her system and trust the corresponding inputs. 14 of the 20 conflicts require some understanding of the needs of the OpenSSH application.

Next, we recognize that the use of *devlog* and *lastlog* does not result in any form of dependence. They manipulate log data which is not interpreted, for example by rotating log files.

Finally, we exclude a few write-up subjects from the system if they cause illegal flows to OpenSSH. This is a judgment call; a sysadmin may decide to trust these subjects instead. If they are trusted, then they must implement appropriate filtering interfaces. *dhcpc* is an example of this as some vulnerabilities have been found for it. For the purpose of our example, we deem dynamic system extension via *hotplug* not necessary. We also eliminate the untrusted subjects that write to the login records of *wtmp*, such as *rlogind*.

Only 3 of the 15 remaining read-type permissions are actually needed for operation in our OpenSSH configuration: the permissions identified by Provos *et al* for creating the pseudo-terminal; initiating OpenSSH connections (by *listen*); and processing user commands via the socket from *net* to *priv*. We remove the 12 unnecessary permission assignments. We note that the *port_type:tcp* permission which permits access to most systems sockets is much coarser-grained than necessary. *listen* only needs access to *sshd_listen:tcp* on port 22. We note that the replication of some permissions for *listen* and *priv* was unnecessary. For example, there is no need for *listen* to accept requests from *net*.

Figure 3.1 shows how challenging it can be to get the permission assignments correct for a given system. The hand-constructed SELinux policy shipped with Fedora Core 4 contained several permissions that needed to be removed. (Our hand-distribution of OpenSSH permissions to *net*, *priv*, and *listen* did not impact these.) Some, such as for *sshd_tmp*, enable actions that we do not want in our configuration (e.g., user administration). Others, though, are simply mistakes that enable information flows that could compromise the integrity of our privileged components. While investigating the source of these errors, we

found that, often, large blocks of permission assignments were made using SELinux convenience macros when only a subset were actually needed. SELinux policies only allow assignment of permissions, not their removal, so we urge policy writers to be careful in their use of such macros.

3.5.5 OpenSSH Filtering Interfaces

We now describe how we identified which permission assignments to classify as FILTER. First, the OpenSSH application developer needs to find where filtering interfaces are necessary. A filtering interface is necessary where low integrity data may be input. For OpenSSH, the interfaces where *listen* receives connection requests from the network and where *priv* receives commands from *net* are the two obvious cases. However, other interfaces may also require filtering in OpenSSH. To find all filtering interfaces, an analysis of the SELinux policy is necessary to see if low integrity inputs may be used by other OpenSSH interfaces.

We use the `settraceonerror` mechanism described in Section 3.4.4 to test our configuration against the default SELinux policy to determine if other interfaces besides the two above require filtering interfaces. We located one: the *userpty* pseudo terminal used by *priv* to communicate with the user shell process.

Next, the application developer must construct effective filtering interfaces. It is the application developer's task to build the filtering interfaces and prove effectiveness to the community. For OpenSSH, the construction of a filtering interface for *priv* to read commands from *net* is one of the main tasks in the privilege-separation done by Provos *et al* [82]. The interface to accept connections in *listen* does not have any special filtering per se, as the connection is not interpreted by *listen*. Also, the *userpty* pseudo terminal in *priv* is only used to pass data to the remote user from the shell process with an encryption step; the contents are not examined.

Finally, once filtering interfaces are found, they must be declared to SELinux in order to use the low integrity permissions. We use the `DO_FILTER()` annotation to declare such interfaces as described in Section 3.4.2.

3.5.6 Verifying vsftpd

We applied the same approach to verifying vsftpd; the results are in Table 3.1. One difference is that the SELinux policy did not reflect the nature of the FTP daemon, which forks per-connection helper processes in a manner very similar to OpenSSH. Instead, there was one subject type for all processes. The child processes do drop Linux privileges (versus than SELinux ones), so they are still largely confined (if a permission is denied in either model, it is denied to the process). The three starred permissions in the table are those that should belong to the unprivileged child processes only, not to the trusted server process, which is why we specify their disposition vis-a-vis the trusted subject type as "REMOVE". The interface between the two is a filtered domain socket. The additional violating permissions were eliminated by excluding some of the same excluded subjects as for OpenSSH, like *rlogind* and *xdm*.

3.6 Related Work

3.6.1 Integrity Models

System integrity has been a difficult problem for security researchers over the years. Most work on integrity has focused on information flow models, supplemented by high assurance (i.e., formal, validation of program correctness, such as Common Criteria EAL7 evaluation [74]).

The Biba integrity model [12] is essentially a dual of the Bell-LaPadula secrecy model [11], where information flows from low integrity subjects to high integrity subjects are prohibited. Like Bell-LaPadula, high assurance components are required to overcome restrictions, but unlike the case for secrecy, illegal (low-to-high) integrity information flows are common (e.g., user requests).

Attempts in subsequent models have not grappled with the fundamental problem that low-to-high integrity flows are common. Denning's work on secure information flow models [31] models information flows between subjects of different labels as a lattice, but models illegal flows as any flow that violates the lattice structure. The LOMAC (low water-

mark) integrity model also prevents high integrity subjects from acting on low integrity information flows, in this case by downgrading the level of a high integrity subject upon receipt of a low integrity information flow [39]. The Clark-Wilson model acknowledges that interfaces are required that can sanitize (or discard) low integrity data, but all interfaces must be capable of sanitization (or discard) and the basis for trusting these interfaces is still high assurance [25]. We are significantly influenced by the Clark-Wilson model's view of requirements on interfaces of high integrity processes, though. Lastly, the recent Caernarvon model allows subjects to span multiple integrity levels such that a subject (i.e., process running with an integrity label range) may be able to read from lower integrity data within its integrity range securely while writing to higher integrity data within its integrity range [84]. Unfortunately, the integrity ranges must be justified by assurance, where significantly broad ranges will still require high assurance. Even after 25 years, we cannot escape the requirement for high assurance, which places too high a burden on too many applications to be practical.

More recent work by Li and Zdancewic [63] present a formal type system that captures intraprogram labeled information flow, with provisions for downgrading of data; type-checking may be used to ensure information-flow security. One may imagine applying their method to interprocess flow, which is controlled by a security policy rather than program source. The `DO_FILTER()` primitive we present may be seen as a downgrading operation in this context.

3.6.2 Policy Analysis

Several access control policy analysis tools have emerged, particularly in the context of SELinux. While the early tools mainly supported query handling, recent tools, such as Gokyo [52], SLAT [50], and Apol [91], now support different kinds of information flow analysis. For example, SLAT enables verification of particular information flow policies, and Gokyo identifies and enables resolution of illegal information flows [55]. Understanding information flows is key to achieving CW-Lite integrity.

Brewer and Nash describe [14] a "Chinese Wall" policy which adds a temporal component. The motivation for this is the financial industry, where analysts may not access a

company's data if they have already accessed the data of a competing company. While they provide a formal model, they do not describe implementation of a verifier.

Permpoontanalarp and Rujimethabhas [79] describe a method for static verification of firewall rules with respect to a defined set of desired information flows, and also give a method of translating desired information flows into rules. They do not present an implementation of their work.

Fu et al. [42] describe a system for verifying IPSEC/VPN policies. Since ensuring that only trusted flows reach a target node requires examining the composition of many upstream nodes' policies, their algorithm spans machines boundaries.

3.6.3 Whole-system Analysis

While there is widespread agreement that whole-system analysis is desirable, there have been relatively few efforts that actually do so on widely-used operating systems. Recently, Chow et al. used hardware-level simulation on a virtual machine in order to perform a dynamic cross-process taint analysis [24]. By contrast, our work focuses on static analysis to prove certain properties about applications' information flow rather than infer them dynamically. Thus, we see our approach as complementary to theirs.

3.7 Summary

Maintaining information-flow integrity is an old and important problem, but as yet an unsolved one in practice: most administrators have not verified that untrusted users cannot compromise inputs to trusted programs on their systems. We have developed a way to automatically verify a meaningful information-flow integrity property with very small changes to existing trusted applications. Because the verification process for system administrators is automated, it can be easily made an integral part of system maintenance, helping ensure that changes to the system's low-level security policy do not inadvertently violate the high-level information-flow integrity constraint.

We call our integrity property *CW-Lite*, since it has the same interprocess dependency semantics as the well-established Clark-Wilson model, but does not address Clark-Wilson's

whole-program formal verification requirement. CW-Lite requires filtering only for untrusted input interfaces, as determined by the system's security policy, and just simple annotations to existing applications to enable least-privilege enforcement and automatic verification. Only conflict resolution requires manual effort by the sysadmin. This reduced goal was chosen to capture most of Clark-Wilson's utility with low manual effort.

We modified the SELinux access control system to enforce CW-Lite and developed tools that support the implementation of compatible program. We verified the practicality of our tools by analyzing privilege-separated OpenSSH and vsftpd, finding and fixing several integrity-violating configuration errors in the shipping SELinux policy.

In the next chapter, we will look at bridging the gap between high-level properties and low-level in a different way. Rather than focusing on static verification of policies for a non-interactive mechanism (the OS security monitor), we will try to translate high-level user actions to a low-level web browser privacy policy dynamically.

Chapter 4

Doppelganger: better browser privacy without all the bother

4.1 Introduction

In this chapter we will first look at the challenges facing implementation of a good browser cookie management system before describing Doppelganger, a novel system which approaches the problem in a new way that eschews a focus on the cookies themselves in favor of a privacy-vs.-functionality tradeoff model.

4.1.1 Background

An HTTP *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 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.

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 4.3.2)
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 4.3.3)
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 4.1: **Summary of Doppelganger's cookie management mechanisms.**

In particular, *third-party cookies* (see Section 4.2 for more details) pose significant privacy risks. Advertising companies such as DoubleClick serve advertisements and web bugs [5] on various web sites and set persistent tracking cookies when browsers automatically fetch these objects during page rendering. Since the user's browser will send back (say) `doubleclick.com` cookies with every subsequent advertisement and web bug request, DoubleClick can track users across every site that serves its ads.

Browser vendors are well aware of this problem. To combat it, browsers include settings to block third-party party cookies [66]. However, the option to block all third-party cookies is disabled by default in both Internet Explorer and Mozilla Firefox [37], and web sites employ a variety of tricks such as HTTP redirection [81], inline frames [88], and Javascript to ensure Web browsers accept their cookies, even when third-party cookies are nominally being blocked (see Section 4.2.2 for details).

Cookies may also offer substantive benefits to users. 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 so accept the

global defaults in their browsers, which apply to all sites. These defaults tend to err on the side of functionality rather than privacy; in part, this is because users can tell when something does not work, but they are not obviously inconvenienced when data about them is silently gathered. 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.

4.1.2 A solution: Doppelganger

We introduce Doppelganger, a web browser privacy tool to help each user formulate her ideal cookie policy. Three basic notions underlying Doppelganger's design are: (1) users don't care about cookies so much as *privacy* and *functionality*; (2) 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 [47, 99]; and (3) users should not be asked to do anything manually that can be done automatically.

A corollary is that it is reasonable to expect users to make a small number of high-level decisions in situations where Doppelganger cannot automatically deduce the correct decision. In our system, the user needs to know little or nothing about the existence of cookies *per se* in order to take advantage of Doppelganger. The principle of using spare resources to do useful work in the background of interactive applications has been applied more widely in recent years. For example, the Microsoft Outlook e-mail client can automatically correct common typos on the fly and the Firefox browser can prefetch pages linked from the current one. Doppelganger takes this idea one step further and uses *client-side parallelism* to explore multiple different possibilities simultaneously.

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 policy, but also takes action to restore the user's session to a good state, as though cookies had been accepted from the start. In short, Doppelganger tries to translate high-level browsing actions and decisions into a low-level policy that closely matches the user's privacy preferences.

4.1.3 Some implications of our approach

A significant qualitative change in using Doppelganger is the exposure of the costs and benefits of some privacy decisions. Previous work suggests that: (1) lack of privacy information (costs), and in particular information in an easily-digestible form, may be a significant obstacle to achieving good outcomes for users [2]; (2) users are sensitive to privacy protections, and are more willing to accept a privacy risk if data about them is protected [29]; and (3) users are willing to compromise some amount of privacy if they are offered meaningful incentives to do so [51]. These are all issues that our system seeks to address.

As with most markets, more complete information has the potential 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 other sites with less favorable exchanges. Indeed, since subjective feelings of trust in users have been shown to induce users to accept more privacy costs [23], 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.

4.1.4 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 [16] 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 4.1.

4.2 HTTP cookies

HTTP cookies are a general mechanism for web servers to store and retrieve persistent state on web clients [80, 56, 57]. 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 a HTTP request to a server, the server has option of including one or more `Set-Cookie` headers in its response. Client 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.

Browsers differ in their determination of context; content whose URL matches the domain of the main page (i.e., the one in the URL bar) is always considered *first-party*, but additional elements may be considered first-party as well. For example, Firefox considers content in IFRAMEs to be first-party regardless of domain. 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. In our system and in our discussion, we do not consider IFRAME content to be first-party if would not otherwise be.

4.2.1 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 [41]: after a user logs in, a web site can also 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 [75] have found that about 58% of users have deleted their cookies at some point. To prevent her web surfing habits 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.

4.2.2 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-

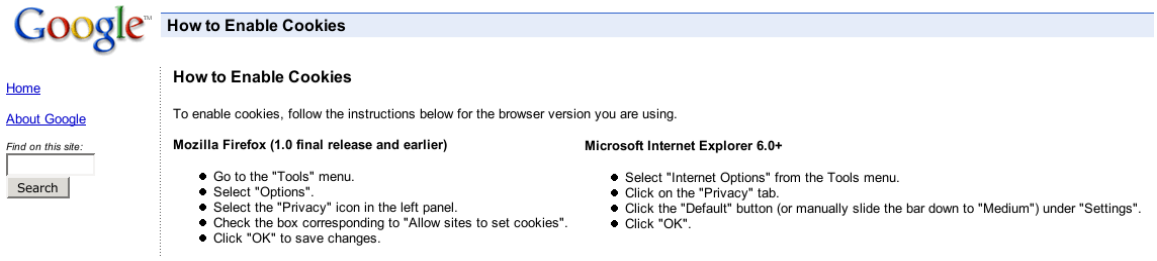


Figure 4.2: **Example of a site's instructions on how to enable cookies.** The instructions for both Firefox and Internet Explorer tell the user to enable all cookies, including third-party cookies.

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, since most web sites do not require clients to accept third-party cookies to operate correctly. However, current web browsers' implementations of this policy fall short of expectations. Advertisers' IFRAME [88] tricks to violate this policy and cause browsers to accept and send some third-party cookies. Also, click-tracking services and advertisers use HTTP redirection [81] 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 on 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 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 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 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 to user to tailor her cookie policy at a fine level of granularity, the usability costs are severe [66]. 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 corrects her cookie policy.

4.2.3 "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 a 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. Figure 4.2 shows an example of such 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.

4.2.4 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 [26], BPS Cookie Shield [18], Cookie Toggle [28], and Permit Cookies [78] add toolbars and enable keyboard shortcuts to help users quickly change cookie policies for the current domain. Add'n'Edit Cookies [4], Cookie Culler [27], and View Cookies [93] 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 [45], 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.

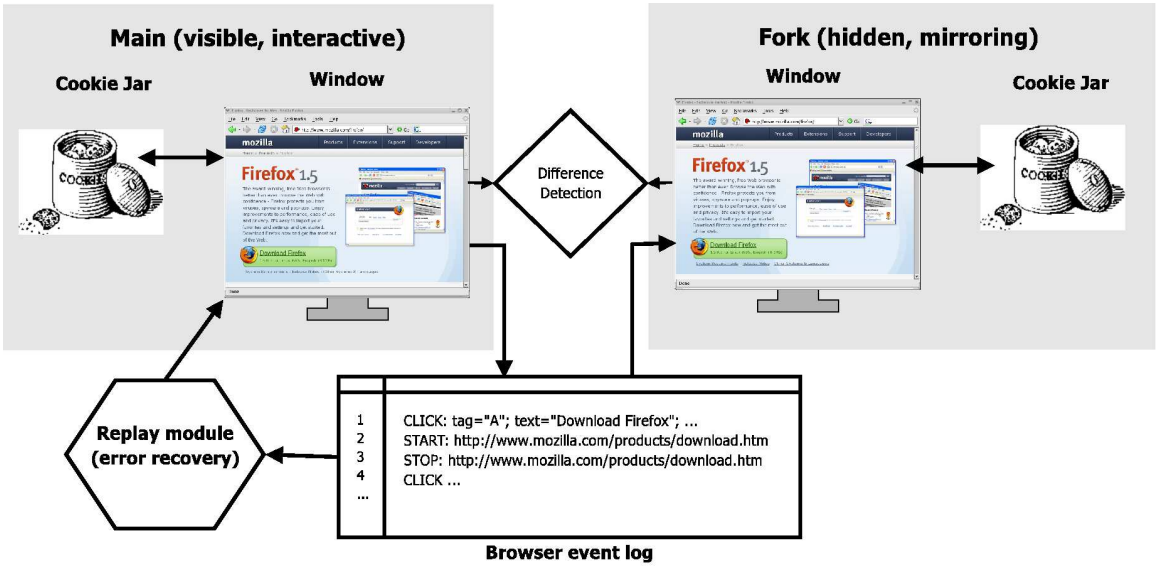


Figure 4.3: **An overview of Doppelganger.** Doppelganger mirrors the user's web session in a hidden fork session whose only configuration difference is the cookies accepted and sent. When Doppelganger detects a difference between the contents of the main window and fork window, it reveals the fork window and asks the user to compare the two (see Figure 4.6). Doppelganger also maintains a log of the user's actions for error recovery.

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 [90] 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 [22]. 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 [8, 20, 62], 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 [66]. 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 a accepting a cookie. Likewise, Bugnosis [5] alerts users to the presence of "web bugs"—invisible images used for tracking, sometimes via cookies—but does nothing to mitigate their effect.

4.3 How Doppelganger works

If a user wants to decide whether or not a particular cookie is beneficial, she must determine is 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 "par-

tial 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 4.3.

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 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 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. As we discussed in Section 4.2.2, the privacy risks from most session cookies are relatively small, but certain session cookies can carry much higher privacy risks. 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 4.3.4.

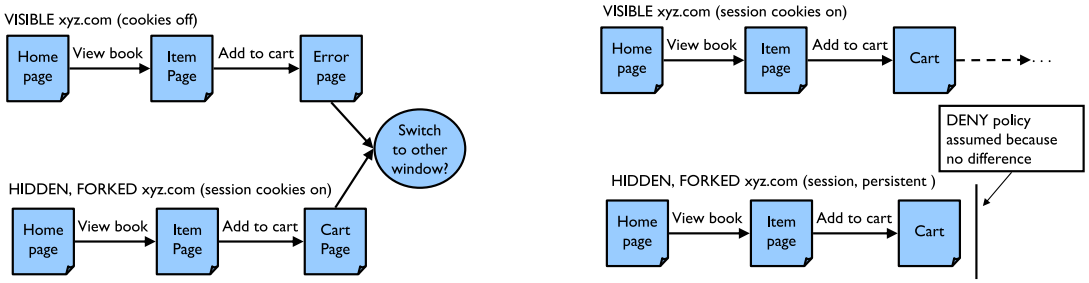
We implemented Doppelganger primarily as an extension to the Mozilla Firefox browser (we used version 1.5.0.2), in about 6000 lines of Javascript code installed independently from the standard browser. We also made a small (30 line) change to the main C++ source code, which we have submitted for inclusion 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 [64], used for watching HTTP request and response traffic, that enabled us to monitor and configure our system.

4.3.1 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 configured in high paranoia mode, the most conservative configuration mode. 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 4.3.2), Doppelganger mirrors each user action and form field value in the fork window (Section 4.3.5); 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



(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 xyz.com.

Figure 4.4: **Graphical representation of mirroring for the example in Section 4.3.1.**

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, Doppelganger 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. Next, suppose the user ultimately decides not to purchase the item yet, and closes her browser.

[1] Low and medium paranoia modes eliminate this dialog box. See Section 4.3.4 for more detail.

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 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 xyz.com. The user gets the same final page as when she pressed Fix Me, but now with any additional benefits of sending the xyz.com persistent cookies received in the first browsing session. We discuss Doppelganger's error recovery mechanism further in Section 4.3.3.

4.3.2 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,

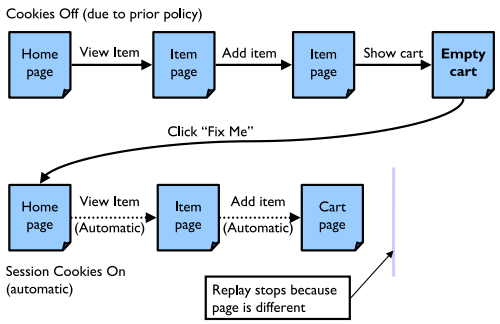


Figure 4.5: **An example use of Doppelganger's error recovery mechanism.** Suppose a decision not to accept cookies was made in the past, but cookies are needed for a shopping cart feature on a site. Doppelganger does not employ mirroring because it has already formed a policy. If needed, the user can 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.

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 4.3.5. In the rest of this section, we will describe how Doppelganger formulates cookies policies and how it maintains two separate cookies spaces for the fork and main windows (Section 4.3.2). We then show how Doppelganger uses the fork window to make automatic decisions affecting the cookie policy (Section 4.3.2).

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:

[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).

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 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 [88] and redirection [81] 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 a IFRAME request as an independent request for a top-level HTML page rather than request for an element of a larger page. Therefore, Firefox classifies cookies for IFRAME requests as first-party cookies instead of third-party with respect to the enclosing page. Doppelganger addresses the IFRAME problem by more reliably determining the context of a HTTP request by matching its tail domain against that of the topmost page's URL. We discuss the countermeasure to redirection tricks in Section 4.3.2.

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

[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.

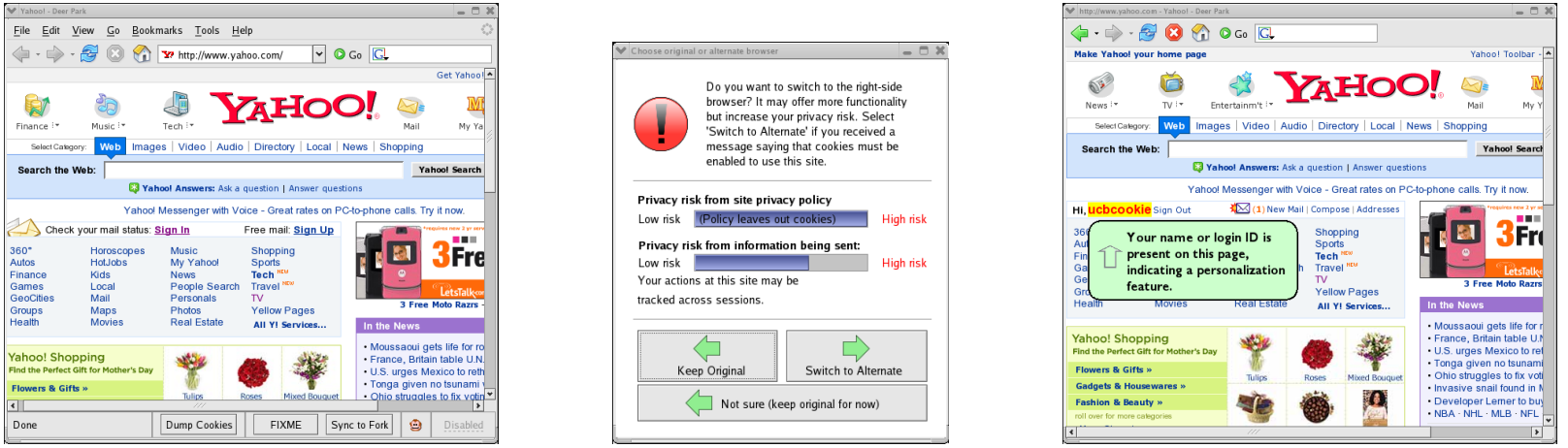
NAME=VALUE pair and impose different access controls on them. Doppelganger achieves this by implementing an optional interface in Firefox, `nsICookieConsent`, designed for deferring cookie policy decisions to an external module. Its original intention was for use with P3P, but that functionality has 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.

We then changed each cookie being set by the fork window's browser by prepending the string "FORK" to the beginning of the cookie's name. For outgoing cookies, our callback filtered the cookie set, allowing only "FORK" cookies for the fork window (and removing that prefix for the outgoing request), and allowing only non-"FORK" cookies for the main browser. This strategy let us use the same cookie management interfaces for fork cookies as ordinary ones, and allowed the browser to handle tasks like removing expired cookies in the usual way.

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 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 4.6. If the user



(a) Main window (b) Comparison dialog (c) Fork (mirroring) window

Figure 4.6: **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.

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.

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.

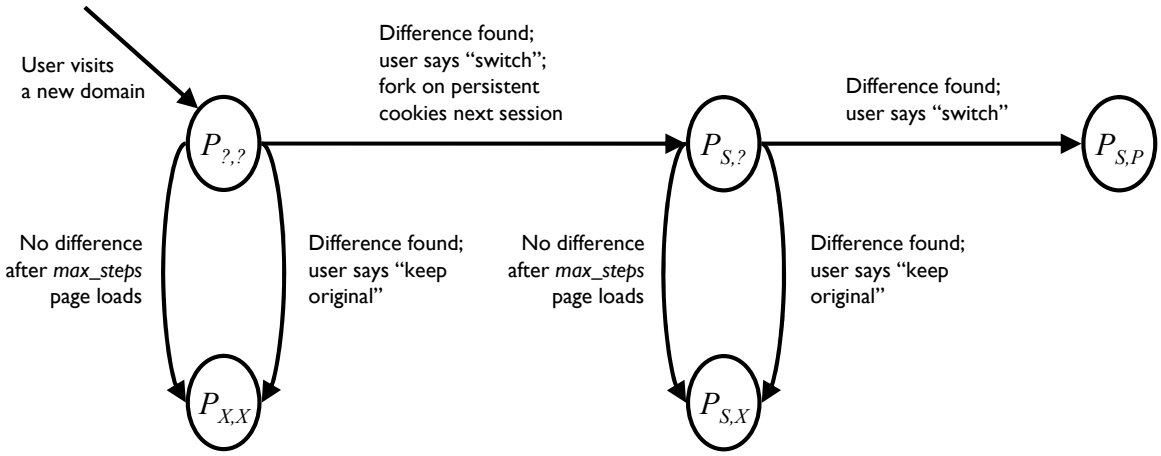


Figure 4.7: **How Doppelganger determines a cookie policy for a domain during the mirroring process.** When Doppelganger detects a difference between the main and fork windows, it prompts the user to decide whether the additional features are worth the potential privacy risk. Doppelganger makes an automatic 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 4.3.4).

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 4.3.3). 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 (i.e., click) 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.

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 [8] 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 4.6.

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 [81] (discussed in Section 4.2.2), 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

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 4.8: **Summary of Doppelganger's different privacy modes.**

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.

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.

4.3.3 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 [16]: 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 `POST`s. 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 other reasons for sites to use these requests appropriately as well.

4.3.4 Higher-privacy modes

Always-on internet connections and more effective computer 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 (potentially) privacy-preserving, but most intrusive mode; remember, though, that comparisons only have to be made when a difference is detected by the mirroring process.

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 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. (If, however 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 4.8.

4.3.5 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 repli-

cates 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.

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.

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 ¡pre¿ ¡B¿click here¡B¿ ¡/pre¿ 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 ¡pre¿<A href='target.html'><B>click here</B><A>"¡/pre¿ 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.

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.

If there were ties, it was often because a link's "href" attribute had been rewritten to include a unique identifier at the end for click tracking. To break ties, we sorted the candidates by smallest edit distance on the "href" attribute from the original.

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.

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.

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.

Other issues with replicating user actions

Frames One complication we encountered in replicating user actions is the presence of frames. 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. One problem that arises is that there can legitimately be elements which are identical, but for which frame they appear in. Currently, we do not record frame-identifying information with each click, except to record the enclosing frame's URL for clicks in IFRAMEs.

Precision 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.

4.4 Evaluation

We evaluated the effectiveness of Doppelganger versus various built-in browser settings by performing a script of common browsing tasks, summarized in Figure 4.9. 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 4.10, 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 4.6); or he can be forced to login upon each visit to a site where he has an account. This latter case occurs when her browser could have accepted a persistent cookie that would serve as an authenticator, but

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 4.9: **Summary of browsing session for evaluation.**

did not for some reason. Analysis of these inconveniences for each setting are discussed in their respective subsections below.

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

4.4.1 All cookies

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

4.4.2 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

Run	Number of sites setting:				
	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 4.10: **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, such as from third-party images, IFRAMEs, and click-tracking HTTP redirections.

more than half as many sites as in the default settting. However, the browser still accepted many cookies that either do not 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 of them set cookies vs. 3 for Doppelganger—since we did not need cookies from, e.g., `cnn.com`. The main advantage of the first-party only setting over more restrictive ones is that we were not asked any questions.

4.4.3 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.

4.4.4 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 3 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 information to help the user decide whether to accept the cookie, except for the domain name.

4.4.5 Doppelganger

Our last test used Doppelganger for cookie management, in High paranoia mode. 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, the same as for Ask.

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 would not appear in Low or Medium paranoia mode, since session cookies would have been automatically enabled (see Section 4.3.4). 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.

4.5 Web site countermeasures

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

4.5.1 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. Alternately, 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 persisent 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.

4.5.2 Cause spurious differences

A web site might try to always create subtle or inconsequental 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.

4.5.3 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' machine to improve performance. Fully understanding the effect of this approach on the user's web browsing experience requires further study.

4.6 Future Work

Doppelganger currently 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 cookies policy is an area for future exploration.

We also believe that our approach may be applied in a broader context. Configuring security and privacy systems can be difficult; oftentimes, the user knows the desired result, but doesn't know what policy will achieve it. Automatic exploration of multiple policies can make that process much easier. In addition, our replay mechanisms can be useful for web application testing and recovering from other kinds of errors. Ideally, web sites would be designed to make replaying easier by removing or exposing currently hidden state.

4.7 Conclusion

We introduced Doppelganger, a novel system for creating and enforcing fine-grained, privacy preserving cookie policies in web browsers from high-level user input 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. We believe Doppelganger has the potential to greatly increase individuals' privacy on web, while retaining functionality and without undue inconvenience.

In the next chapter, we describe a controlled usability study to test the usability and performance of Doppelganger with ordinary users.

Chapter 5

Doppelganger usability study

5.1 Introduction

In order to test the efficacy and usability of Doppelganger, we conducted a controlled usability study which compared Doppelganger against two browser settings: the *Default*, allow-all policy, and the *Ask* policy, which requires user approval for each cookie. The first of these represents the worst case for privacy, since no cookies are rejected, and the second represent the potentially best case for privacy since the user can reject every cookie that she does not want.

The goal was to get both objective and subjective evaluations of these cookie management options. We would expect there to be a tradeoff between a few variables: (1) ease of use, including intrusiveness and ease of decision-making with respect to the mechanism itself; (2) functionality, which here means that users were able to access all the sites' offerings that they (the users) wished to access; and (3) privacy, measured in cookies accepted. We hoped to learn what these were for our three candidates.

Subjective measures include perceived ease of completing the task list for each scenario and ease of decision making in each. Objective measures includedcompletion rate for each task and the number of cookies accepted.

The full text of the survey questions and reponses may be found in the Appendices.

5.2 Setup

5.2.1 Profile of the subjects

Our tests were conducted with the help of the XLab (Experimental Social Science Laboratory) at UC Berkeley, which recruited and scheduled subjects, and provided space and laptop computers for conducting the study. The XLab maintains a pool of prospective subject candidates for experiments; subjects can register for the studies online. Per the XLab protocol, all of the subjects were students, staff, or faculty at UC Berkeley. Subjects were not selected for any other characteristics. No attempt was made to control for gender or any particular privacy preference; this choice was justified by the finding by Hann et al. [51] that users' willingness to choose one site over another with respect to privacy is unaffected by all the measured personal characteristics of the user, including gender. (They found that preferences did vary with site characteristics, however.)

We conducted our experiment with 19 subjects; however, one subject's data could not be used due to data corruption, yielding a final total of 18. Subjects were regular users of web browsers, most browsing for 10-20 hours a week. Tabulated data for these survey responses is provided in Figures 5.1 and 5.2. They were "somewhat concerned" about their online privacy in general, and slightly more concerned about tracking in particular. All the subjects had heard of cookies; 5 had heard of them but didn't understand how they work; 9 basically understood how they work; and the remaining 4 had a deeper knowledge, understanding the differences between types of cookies. All but 4 subjects had taken steps to manage or monitor their own cookies; this is consistent with findings in a more general population that show that as many as 58% of users delete their cookies [75], with 40% doing so each month.

5.2.2 Method

Subjects were given a packet of information that explained what browser cookies are and the privacy risks that can result from accepting cookies. They were also given a task list containing the web browsing tasks they would perform. They were instructed to perform the entire list three times—three different "scenarios"—each with a different privacy

Hours	Count
0-5	0
5-10	2
10-20	9
20+	7

(a) "How many hours a week do you use a web browser?"

Level of concern	Count
(1) Not concerned	2
(2) Somewhat concerned	11
(3) Concerned	5
(4) Paranoid	0
Average level	2.2

(b) "On a scale of 1-4, how concerned are you about your privacy online?"

Level of concern	Count
(1) Not concerned	3
(2) Somewhat concerned	8
(3) Concerned	6
(4) Paranoid	1
Average level	2.3

(c) "In particular, how concerned are you about your browsing habits and actions being recorded by the sites you visit or by third party sites?"

Figure 5.1: **Usability study survey results — General questions**

Choice	Count
Not at all	0
I had heard of them, but didn't really understand how they work	5
I basically understood how cookies work	9
I understood terms like 'persistent cookies' and 'third party cookies'	4

(a) "Before this study, how familar were you with web browser cookies?"

Choice	Count
Yes	14
No	4

(b) "Have you taken any steps to manage or monitor the cookies in your browser? For example, have you deleted cookies, examined cookies, or changed your cookie preferences, or used third-party software that did so?"

Figure 5.2: **Usability study survey results — Familiarity with cookies**

setting. The first scenario used the Default setting, and the second and third used Ask and Doppelganger in some order. During the browsing session, they were to represent the privacy preferences of a hypothetical user rather than their own preferences. The task list is shown in Figure 5.4; the trust levels of the hypothetical user are shown after the site URLs. After performing the tasks in each of the three scenarios, the users took an exit survey.

During the browsing sessions, we recording screen-capture movies for each user, so we could review the sessions later if necessary. We also installed an extension in the browser that recorded the contents of the cookie jar, including session cookies, to a file. This extension was used in each of the three scenarios.

5.3 Results

5.3.1 Ease of use (tabular data in Figure 5.3).

Subjects indicated that performing the tasks under the Default scenario was quite easy, averaging 5.8 on a scale of 1 to 6, 6 being "Very easy". This was to be expected, and indeed was an intentional study design decision. We did not want the tasks themselves to be

difficult to perform, so that we could ascribe essentially all the difficulty the subjects had in the Doppelganger and Ask scenarios to those respective cookie management mechanisms.

Completion of the tasks while using Doppelganger was judged to be somewhat more difficult than for Default, averaging 4.8, and still more difficult with Ask Me, averaging 3.8. In order to understand what made the latter two settings more difficult, we asked users a survey question about what they thought of the number and kind of questions they were asked:

> How did you feel about the number and difficulty of cookie management tasks and decisions you faced in Scenario #2 [or 3]? By "easy" we mean that it was generally clear which button to push and when, and that you knew the correct choice for each dialog box. By "difficult" we mean that it was generally unclear what was the right action in each situation.
>
> 1. There were too many tasks (dialogs and button presses) but they were easy to handle
> 2. There were too many tasks and they were difficult to handle
> 3. There were not too many tasks and they were easy to handle
> 4. There were not too many tasks but they were difficult to handle

Since the primary difference between Default and the other two settings was the dialog boxes presented to the user, we felt that this question would capture some of the reasons behind the usability gap. The results were striking:

Setting	**Too many**	**Not too many**	**Difficult**	**Easy**
Doppelganger	2	16	3	15
Ask	16	2	10	8

While users felt that Doppelganger did not impose too many tasks on them, and that those tasks were easy, they felt quite the opposite about Ask. We may also see how the ease-of-completion numbers vary with subjects' answers to these questions:

	Average ease (Doppelganger)	**Average ease (Ask)**
Easy	4.9	4.4
Difficult	4	3.3
Not too many	4.9	5
Too many	4	3.6

Users who had trouble with the decisions in volume or clarity also had a significantly more difficult time completing the tasks in both scenarios. This suggests that these two metrics may be good predictors of ease of use.

The distributions of the ease-of-completion numbers for Ask and Doppelganger were of different shapes (see Figure 5.3). While 12 of the 18 subjects decided that completion with Ask was "relatively easy" (the rest were uniformly distributed from "very difficult" to "very easy"), 10 of 18 found completion with Doppelganger to be "easy" or "very easy". Only 2 subjects deemed completion with Ask "easy" or "very easy" despite the task list's intrinsic ease (as measured by Default). This seems to indicate that Ask is not satisfactory for a large majority of users, but that Doppelganger may be well suited for a significant portion of the population.

Subjects were given the opportunity to provide free-form feedback for each scenario in response to the question "What did you like or not like about Scenario #N?". Comments were generally positive for Default; the only criticism offered was by one subject who said that s/he "did not know which cookies were being accepted by [his/her] browser". Subjects were much more critical of the Ask scenario. Several complained about the number of dialog boxes. Another common refrain was that subjects were confused; they hard a hard time figuring out what cookies they were enabling, and at the end were not sure if they had made the right choices. A few noted that the presence of dialogs asking about cookies from other sites (i.e., third party cookies) was confusing, and they did not always notice when this happened.

5.3.2 Performance

We measured performance in a few ways: completion of each scenario; whether the user encountered any problems along the way (even if the scenario was completed); and the number of cookies accepted.

Every subject was able to complete the Default scenario without any problems. Six subjects were unable to complete Task 2 (Netflix) using Ask, and 3 were unable to do so using Doppleganger. In each case the subject had chosen to deny cookies from Netflix, making it impossible to access the site. This is not surprising, given that Netflix had been

Ease	Count
(1) Very difficult	0
(2) Difficult	0
(3) Relatively difficult	0
(4) Relatively easy	1
(5) Easy	3
(6) Very easy	14
Average	5.8

(a) "How easy was it for you to complete the tasks in scenario [Default]?"

Ease	Count
(1) Very difficult	0
(2) Difficult	0
(3) Relatively difficult	2
(4) Relatively easy	6
(5) Easy	4
(6) Very easy	6
Average	4.8

(b) "How easy was it for you to complete the tasks in scenario [Doppelganger]"

Ease	Count
(1) Very difficult	1
(2) Difficult	1
(3) Relatively difficult	2
(4) Relatively easy	12
(5) Easy	1
(6) Very easy	1
Average	3.8

(c) "How easy was it for you to complete the tasks in scenario [Ask]?"

Figure 5.3: **Survey results — ease of completion**. Doppelganger and Ask were the second and third scenarios; for 10 of the 18 subjects, Ask came before Doppelganger.

Figure 5.4: **Task list used in Doppelganger usability study.**

Open the Firefox browser, using the icon corresponding to the scenario you are on.
Task 1: www.excite.com Trust Level: HIGH Excite.com is a web portal which offers services such as email, news, and stock quotes
0) If you are using Doppelganger, take a moment to look at the lower right corner of your browser. You should see the FIXME button and a status indicator saying "Synced". You can read more about how to use these in the Doppelganger scenario description, above. 1) Visit http://www.excite.com 2) Log in to an email account (ID: ucbtest; password: gobears) 3) Read the message from Testy McTest (ucbcookie@yahoo.com). Make sure you can see the "secret phrase". 4) Return to www.excite.com 5) Find a news article and click it. 6) If you can read the article, then you are done with Task 1.
Task 2: www.netflix.com Trust Level: LOW Netflix is an online DVD movie rental service 1) Visit http://www.netflix.com 2) Choose "Browse Selection" 3) Search for the movie "Shrek". 4) If you can see the name of the director of "Shrek", you are done with Task 2
Task 3: www.weather.com Trust Level: LOW Weather.com provides weather forecasts and historical data 1) Visit http://www.weather.com 2) Find the forecast for zip code 11217 using the "Local Weather" box 3) If you see the city name for zip code 11217, you are done.
Close the brower and wait a few seconds. Now, re-open the browser, using the same link.
Task 4: Revisit www.excite.com 1) Visit http://www.excite.com 2) Check your email again, and reply to the message from ucbcookie@yahoo.com. Write "I'm almost done!"
Task 5: Revisit www.weather.com 1) Visit www.weather.com 2) Find the current temperature for zip code 11217 again.

marked as having a low trust level, but we should take that into account in interpreting the cookie numbers, which were deflated relative to subjects that accessed Netflix. Subjects did not report problems other than their annoyance or confusion with the dialog boxes, which was much more prevalent with Ask than with Doppelganger (see above).

The accepted-cookies data, shown in Figure 5.5, is revealing. For every type of cookie, the Default setting allowed the most sites to set cookies by a wide margin, followed by Ask and then Doppelganger. The Default setting also led to the acceptance of a large number of third-party sites' cookies (almost 8 sites on average). Notably, subjects did not allow any third-party cookies using Doppelganger; this represents a significant improvement, since even a small number of third-party sites' cookies can allow tracking at an enormous number of other sites. The variance was much larger for the Ask setting than for the other two. This is to be expected given subjects' confusion about what to do, leading to, in many cases, arbitrary decision-making. Whereas the flexibility of Ask might lead to large variance in the real world, since users many have very different preferences, we would expect to see much less variance in a controlled environment where users were told to represent the same privacy preferences. That we did not see small variance with Ask, but did see it with Doppelganger, says a lot about the need to help users make informed choices. Variance in the Default setting was zero for first-party cookies, which is to be expected: each of the three first-party sites set as many cookies as it wanted. Third-party variance was due to randomness in the set of advertisements being displayed.

As we discussed above, in order to do a more fair comparison, we would need to consider the effects of the missing Netflix cookies for some subjects. Since only one site is affected, and it is first-party, the maximum increases we could see in the averages are 6/18 = 0.33 for Ask in each of the first-party categories, and 3/18 = 0.17 for Doppelganger. Neither would make a significant difference in our interpretation.

5.4 Discussion

Perhaps unsurprisingly, Doppelganger's ease of use was rated in between that of the Default setting and that of the Ask setting. Since Doppelganger also performed as well or bet-

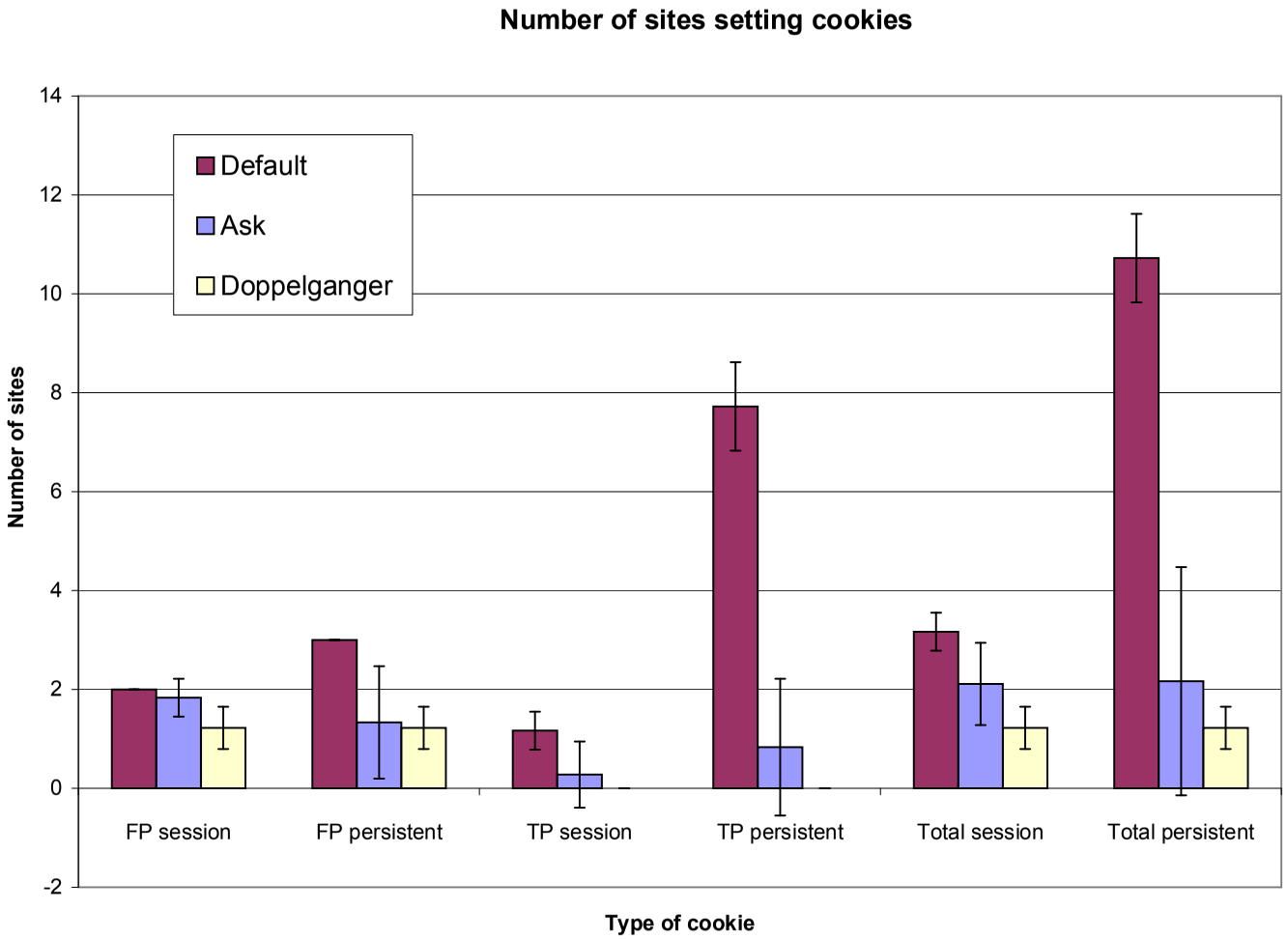


Figure 5.5: **Cookies accepted for each cookie management setting.** The figures used here are the averages across all users, taken from the cookie jar as of the end of task list (Figure 5.4). Each type of cookies from each site counted as "1" in the tally; i.e., we did not distinguish between accepting two first-party session cookies from a site and accepting three first-party session cookies from the site. The end of the task list represents the end of the second browsing session, since users had to close and restart the browser after Task 3. Error bars represent the standard deviation.

ter than Ask in terms of privacy protection, in every cookie category, we may fairly say that the Ask setting is dominated by Doppelganger. The remaining question is whether the privacy protections offered by Doppelganger—and the data suggest that they are significant—outweigh the usability decline suffered by users in using it relative to the Default setting. Since users care enough about cookies to take action to manage them, there is reason to believe that if we offer a reasonable option, users will actually use it.

Comments from the subjects seemed to indicate that they like to see how they are doing; that is, how their choices are affecting their privacy. A visual feedback may be a solution to this, perhaps showing the cookie status of the current site, or how many sites' cookies the user has blocked. This is particularly useful since information may saved in a non-obvious way. For example, `weather.com` remembers the zip code that a user types in using a persistent cookie, even if the user is asked to re-enter the zip code on future visits. While there is a "customize" feature that saves users having to re-enter the zip code, in reality it is likely no worse from a privacy standpoint, since the originally entered zip code is sent via cookie in both cases. Some method of showing potential risks to the user may increase users' willingness to make more privacy decisions.

There were also a few comments that suggested that a better layout of some of the Doppelganger dialogs would be helpful. Our observations of the subjects as they used Doppelganger suggested that its asynchronous, active nature may require better explanations of what it is doing, as it is doing it.

There were some potential sources of error and uncertainty in our study. First, the subjects were not representative of the general web browsing population; since all were affiliated with UC Berkeley, they were likely more educated than average. In addition, it is possible that more users are needed to confirm our findings. A possible source of bias is that users were informed that we (the experimenters) were also the authors of Doppelganger; this may have affected their experience or reports of their experience.

Ultimately, what is really needed is a large-scale, long-term deployment. It is difficult to predict the steady-state usability of Doppelganger from a one-hour study session in which the subject is not only seeing Doppelganger for the first time, but may well be learning quite a bit about cookies and online privacy as well. Nonetheless, we believe that the study made a fairly good case for the viability of Doppelganger, and users made good suggestions

that can be used to improve the system. People found it to be significantly more usable that its privacy-comparable counterpart, Ask; none accepted third-party cookies, while the majority did accept third-party cookies with Ask. Doppelganger also fared well in some absolute sense; most users found it "easy" or "very easy" to complete the browsing tasks while using it.

More data is also needed to assess the Doppelganger's performance in more demanding browsing situations where users may have trouble performing tasks even in the Default setting. We also did not evaluate the usability of the replay mechanism in our study, preferring to focus on the left-or-right comparison mechanism.

5.5 Conclusion

Difficulty in configuration is a significant obstacle to achieving practical security and privacy. This task assumes two forms: writing down low-level policies that match high-level needs (translation) and ensuring that a low-level policy meets a high-level requirement (verification). Both can be challenging to do manually, in no small part because powerful systems and complex policy configuration languages tend to go hand-in-hand. We do not want to sacrifice this power and flexibility, but we do want to make it feasible to make sure that the broad strokes are always right and that making a detailed, personalized policy is not always painful.

We have explored two systems, respectively tackling the verification and translation problems. We've seen that when it comes to verification of operating system security policies, defining the security property itself is important: we want a property that is both useful and amenable to automated analysis. We developed a suitable lightweight property, *CW-Lite*, for this purpose, and provided automated tools to verify the property. Additional tools that we implemented make it easier to debug violations of *CW-Lite* and easier for developers to write conforming applications. The net result is that system administrators can more easily have confidence in the integrity (in an information-flow sense) of their trusted applications. With *Doppelganger*, we targeted a wider audience, introducing a more intuitive and powerful browser cookie management mechanism. Again, formulating

the problem was central: we recast the abstruse cookie-acceptance decision problem into a more accessible privacy-vs.-functionality tradeoff. We also tried to make as many decisions automatically as possible, and to automate all mechanical series of actions (such as replaying sessions after fixing a site's cookie policy). The results from our own tests and from a controlled usability study suggest that Doppelganger is very effective at improving privacy by reducing the number of accepted cookies, and imposes a modest burden on the user in doing so. Only time and wider testing will show if this burden is above the threshold at which users will continue to use it in the long term.

It is our hope that these two efforts to mitigate the configuration problem serve as useful steps in a larger effort to apply the same principles to all security and privacy systems. In the end, an unusable system is very likely to be an insecure one; it is our belief that usability is currently the weakest link in the security chain, and that the greatest net rewards are to be found in improving it.

Bibliography

[1] ACQUISTI, A. Privacy in electronic commerce and the economics of immediate gratification. In *EC '04: Proceedings of the 5th ACM conference on Electronic commerce* (New York, NY, USA, 2004), ACM Press, pp. 21–29.

[2] ACQUISTI, A., AND GROSSKLAGS, J. Privacy and rationality in individual decision making. *IEEE Security and Privacy 3*, 1 (2005), 26–33.

[3] ADAMS, A., AND SASSE, M. A. Users are not the enemy. *Communications of the ACM 42*, 12 (1999), 40–46.

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

[5] ALSAID, A., AND MARTIN, D. Detecting web bugs with bugnosis: Privacy advocacy through education. In *Proceedings of the 2002 Workshop on Privacy Enhancing Technologies* (2002).

[6] ANDERSON, R., STAJANO, F., AND LEE, J.-H. Security policies. In *Advances in Computers vol. 55*, M. Zelkowitz, Ed. Academic Press, 2001, pp. 185–235.

[7] Apache web server. Available from `http://www.apache.org`.

[8] ARSHAD, F. Privacy Fox - A JavaScript-based P3P Agent for Mozilla Firefox. `http://privacyfox.mozdev.org/PaperFinal.pdf`, 2004.

[9] BALFANZ, D., DURFEE, G., GRINTER, R. E., AND SMETTERS, D. K. In search of usable security: Five lessons from the field. *IEEE Security and Privacy 2*, 5 (2004), 19–24.

[10] BARTAL, Y., MAYER, A., NISSIM, K., AND WOOL, A. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst. 22*, 4 (2004), 381–420.

[11] BELL, D., AND LA PADULA, L. Secure computer systems: Mathematical foundations (volume 1). In *Technical Report ESD-TR-73-278, Mitre Corporation* (1973).

[12] BIBA, K. J. Integrity considerations for secure computer systems. In *Technical Report MTR-3153, Mitre Corporation* (June 1975).

[13] BOEBERT, W. E., AND KAIN, R. Y. A practical alternative to hierarchical integrity policies, 1985.

[14] BREWER, D. F. C., AND NASH, M. J. The chinese wall security policy. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy* (1989), pp. 206–214.

[15] BROWN, A., AND PATTERSON, D. Embracing failure: A case for recovery-oriented computing. In *2001 High Performance Transaction Processing Symposium* (2001).

[16] BROWN, A., AND PATTERSON, D. Undo for operators: Building an undoable e-mail store. In *USENIX 2003 Annual Technical Conference* (June 2003).

[17] BROWN, A., AND PATTERSON, D. A. Rewind, Repair, Replay: Three R's to dependability. In *10th ACM SIGOPS European Workshop* (Saint-Emilion, France, Sept. 2002).

[18] Bulletproofsoft bps popup & cookie shield. `http://www.bulletproofsoft.com/popupshield.html`.

[19] BYERS, S., CRANOR, L., KORMANN, D., , AND MCDANIEL, P. Searching for privacy: Design and implementation of a p3p-enabled search engine. In *2004 Workshop on Privacy Enhancing Technologies (PET2004)* (2004).

[20] BYERS, S., CRANOR, L. F., AND KORMANN, D. Automated analysis of P3P-enabled web sites. In *ICEC '03: Proceedings of the 5th international conference on Electronic commerce* (New York, NY, USA, 2003), ACM Press, pp. 326–338.

[21] BYERS, S., CRANOR, L. F., KORMANN, D. P., AND MCDANIEL, P. D. Searching for privacy: Design and implementation of a p3p-enabled search engine. In *Privacy Enhancing Technologies* (2004), pp. 314–328.

[22] CENTER, E. P. I., AND JUNKBUSTERS. Pretty poor privacy: An assessment of P3P and Internet privacy. `http://www.epic.org/reports/prettypoorprivacy.html`, June 2000.

[23] CHELLAPPA, R. K., AND SIN, R. G. Personalization versus privacy: An empirical examination of the online consumer's dilemma. *Inf. Tech. and Management 6*, 2-3 (2005), 181–202.

[24] CHOW, J., PFAFF, B., GARFINKEL, T., CHRISTOPHER, K., AND ROSENBLUM, M. Understanding data lifetime via whole system simulation. In *Proceedings of the* 13^{th} *USENIX Security Symposium* (2004).

[25] CLARK, D. D., AND WILSON, D. R. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy* (1987).

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

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

[28] Cookie Toggle. `http://cookietoggle.mozdev.org/`.

[29] CULNAN, M. J., AND ARMSTRONG, P. K. Information privacy concerns, procedural fairness, and impersonal trust: An empirical investigation. *Organization Science 10*, 1 (1999), 104–115.

[30] DENLEY, I., AND WESTON-SMITH, S. Implementing access control to protect the confidentiality of patient information in clinical information systems in the acute hospital. *Health Informatics Journal*, 4 (1999), 174–178.

[31] DENNING, D. E. A lattice model of secure information flow. In *Communications of the ACM, vol. 19, no. 5* (May 1976), pp. 236–243.

[32] DENNING, D. E., AND DENNING, P. J. Certification of programs for secure information flow. *Communications of the ACM 20*, 7 (1977), 504–513.

[33] DEWITT, A. J., AND KULJIS, J. Aligning usability and security: a usability study of polaris. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security* (New York, NY, USA, 2006), ACM Press, pp. 1–7.

[34] DYER, J., LINDEMANN, M., PEREZ, R., SAILER, R., SMITH, S. W., VAN DOORN, L., AND WEINGART, S. Building the ibm 4758 secure coprocessor. In *IEEE Computer, (34)10:57-66* (October 2001).

[35] EGELMAN, S., CRANOR, L. F., AND CHOWDHURY, A. An analysis of p3p-enabled web sites among top-20 search results. In *ICEC '06: Proceedings of the 8th international conference on Electronic commerce* (New York, NY, USA, 2006), ACM Press, pp. 197–207.

[36] ENGLAND, P., LAMPSON, B., MANFERDELLI, J., PEINADO, M., AND WILLMAN, B. A trusted open platform. In *IEEE Computer* (July 2003), pp. 55–62.

[37] Firefox web browser. `http://www.firefox.com`.

[38] FOLEY, S., GONG, L., AND QIAN, X. A security model of dynamic labeling providing a tiered approach to verification. In *IEEE Symposium on Security and Privacy* (Oakland, CA, 1996), IEEE Computer Society Press, pp. 142–154.

[39] FRASER, T. LOMAC: Low water-mark integrity protection for cots environments. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy* (May 2000).

[40] FRIEDMAN, B., HOWE, D., AND FELTEN, E. Informed consent in the Mozilla browser: Implementing value sensitive design. In *35th Annual Hawaii International Conference on System Sciences (HICSS'02)* (2002).

[41] FU, K., SIT, E., SMITH, K., AND FEAMSTER, N. Dos and Don'ts of client authentication on the web. In *10th USENIX Security Symposium* (August 2001), pp. 251–268.

[42] FU, Z., WU, S. F., HUANG, H., LOH, K., GONG, F., BALDINE, I., AND XU, C. IPSec/VPN security policy: Correctness, conflict detection, and resolution. In *POLICY* (2001), pp. 39–56.

[43] The gallery web photo album organizer. `http://gallery.menalto.com/`.

[44] GIDEON, J., CRANOR, L., EGELMAN, S., AND ACQUISTI, A. Power strips, prophylactics, and privacy, oh my! In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security* (New York, NY, USA, 2006), ACM Press, pp. 133–144.

[45] GOECKS, J., AND MYNATT, E. D. Social approaches to end-user privacy management. In *Security and Usability: Designing Secure Systems That People Can Use*, L. F. Cranor and S. Garfinkel, Eds. O'Reilly, 2005, ch. 25, pp. 523–545.

[46] GOGUEN, J. A., AND MESEGUER, J. Security policies and security models. In *IEEE Symposium on Security and Privacy* (1982), pp. 11–20.

[47] GOOD, N., DHAMIJA, R., GROSSKLAGS, J., THAW, D., ARONOWITZ, S., MULLIGAN, D., AND KONSTAN, J. Stopping spyware at the gate: A user study of notice, privacy and spyware. In *Symposium on Usable Privacy and Security (SOUPS) 2005* (July 2005).

[48] GORDON, L. A., AND LOEB, M. P. The economics of information security investment. *ACM Trans. Inf. Syst. Secur. 5*, 4 (2002), 438–457.

[49] GUTTMAN, J., HERZOG, A., AND RAMSDELL, J. Information flow in operating systems: Eager formal methods. In *Workshop on Issues in the Theory of Security (WITS)* (2003).

[50] GUTTMAN, J., HERZOG, A., AND RAMSDELL, J. Information flow in operating systems: Eager formal methods. In *Workshop on Issues in the Theory of Security (WITS)* (2003).

[51] HANN, I.-H., HUI, K.-L., LEE, T. S., AND PNG, I. P. L. Online information privacy: Measuring the cost-benefit trade-off. In *Proceedings of the Twenty-Third International Conference on Information Systems* (2002), pp. 1–8.

[52] JAEGER, T., EDWARDS, A., AND ZHANG, X. Policy management using access control spaces. In *ACM Transactions on Information and System Security (TISSEC) 6(3)* (August 2003).

[53] JAEGER, T., SAILER, R., AND SHANKAR, U. Prima: policy-reduced integrity measurement architecture. In *SACMAT* (2006), pp. 19–28.

[54] JAEGER, T., SAILER, R., AND ZHANG, X. Analyzing integrity protection in the selinux example policy. In *Proceedings of the* 12^{th} *USENIX Security Symposium* (2003).

[55] JAEGER, T., SAILER, R., AND ZHANG, X. Resolving constraint conflicts. In *Proceedings of the* 9^{th} *ACM Symposium on Access Control Models and Technologies* (2004).

[56] KRISTOL, D., AND MONTULLI, L. HTTP State Management Mechanism. RFC 2109 (Proposed Standard), Feb. 1997. Obsoleted by RFC 2965.

[57] KRISTOL, D., AND MONTULLI, L. HTTP State Management Mechanism. RFC 2965 (Proposed Standard), Oct. 2000.

[58] KRISTOL, D. M. HTTP cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology 1*, 2 (Nov. 2001), 151–198.

[59] KUO, C., PERRIG, A., AND WALKER, J. Designing an evaluation method for security user interfaces: lessons from studying secure wireless network configuration. *interactions 13*, 3 (2006), 28–31.

[60] LAMPSON, B. W. A note on the confinement problem. *Commun. ACM 16*, 10 (1973), 613–615.

[61] LANDWEHR, C. E. Formal models for computer security. *ACM Comput. Surv. 13*, 3 (1981), 247–278.

[62] LEVY, S. E., AND GUTWIN, C. Improving understanding of website privacy policies with fine-grained policy anchors. In *WWW '05: Proceedings of the 14th international conference on World Wide Web* (New York, NY, USA, 2005), ACM Press, pp. 480–488.

[63] LI, P., AND ZDANCEWIC, S. Downgrading policies and relaxed noninterference. In *Proceedings of the 2005 Symposium on Principles of Programming Languages* (2005).

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

[65] MAYER, A., WOOL, A., AND ZISKIND, E. Fang: A firewall analysis engine. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2000), IEEE Computer Society, p. 177.

[66] MILLETT, L., FRIEDMAN, B., AND FELTEN, E. Cookies and web browser design: Toward realizing informed consent online. In *Proceedings of the CHI 2001 Conference on Human Factors in Computing Systems* (April 2001), pp. 46–52.

[67] MOORE, K., AND FREED, N. Use of HTTP State Management. RFC 2964 (Best Current Practice), Oct. 2000.

[68] Mozilla web browser. `http://www.mozilla.org`.

[69] MYERS, A. C., AND LISKOV, B. A decentralized model for information flow control. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles* (New York, NY, USA, 1997), ACM Press, pp. 129–142.

[70] MYERS, A. C., AND LISKOV, B. Complete, safe information flow with decentralized labels. In *19th IEEE Symposium on Research in Security and Privacy (RSP)* (Oakland, California, May 1998).

[71] NATIONAL SECURITY AGENCY. Security-enhanced linux (SELinux). http://www.nsa.gov/selinux, 2001.

[72] NAUMOVICH, G., AND CENTONZE, P. Static analysis of role-based access control in j2ee applications. *SIGSOFT Softw. Eng. Notes 29*, 5 (2004), 1–10.

[73] NETCRAFT. Netcraft September 2006 Web Server Survey. http://news.netcraft.com/archives/2006/09/05/september_2006_web_server_survey.html.

[74] NIST, C. S. D. Common criteria for it security evaluation. csrc.nist.gov/cc/, 2004.

[75] O'MALLEY, G. Jupiter analyst: Nielsen research confirms users delete cookies. http://publications.mediapost.com/index.cfm?fuseaction=Articles.san&s=28883&Nid=12855&p=297686.

[76] ONESTAT.COM. Global usage share Mozilla Firefox has increased according to onestat.com. http://www.onestat.com/html/aboutus_pressbox44-mozilla-firefox-has-slightly-increased.html, 2006.

[77] Openssh secure shell server. Available from http://www.openssh.org.

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

[79] PERMPOONTANALARP, Y., AND RUJIMETHABHAS, C. A unified methodology for verification and synthesis of firewall configurations. In *ICICS '01: Proceedings of the Third International Conference on Information and Communications Security* (London, UK, 2001), Springer-Verlag, pp. 328–339.

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

[81] POULSEN, K. Microsoft cookies jump domains. http://www.securityfocus.com/news/83, September 2000.

[82] PROVOS, N., FRIEDL, M., AND HONEYMAN, P. Preventing privilege escalation. In *Proceedings of the* 12^{th} *USENIX Security Symposium* (2003).

[83] SAILER, R., ZHANG, X., JAEGER, T., AND VAN DOORN, L. Design and implementation of a tcg-based integrity measurement architecture. In *USENIX Security Symposium* (2004), pp. 223–238.

[84] SCHELLHORN, G., REIF, W., SCHAIRER, A., KARGER, P., AUSTEL, V., AND TOLL, D. Verification of a formal security model for multiapplicative smart cards. In *Proceedings of the* 6^{th} *European Symposium on Research in Computer Security (ESORICS)* (2000).

[85] SCHNEIDER, F. B. Enforceable security policies. *ACM Trans. Inf. Syst. Secur. 3*, 1 (2000), 30–50.

[86] SHANKAR, U., JAEGER, T., AND SAILER, R. Toward automated information-flow integrity verification for security-critical applications. In *Proceedings of the Network and Distributed System Security Symposium, (NDSS 2006)* (2006).

[87] SHANKAR, U., AND KARLOF, C. Doppelganger: Better browser privacy without the bother. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006)* (New York, NY, USA, 2006), ACM Press.

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

[89] SMETTERS, D., AND GRINTER, R. E. Moving from the design of usable security technologies to the design of useful secure applications. In *New Security Paradigms Workshop* (Virginia Beach, VA, September 2002).

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

[91] TRESYS TECHNOLOGY. Apol SELinux policy analysis tool. `http://oss.tresys.com/projects/setools`.

[92] VERMA, P., AND PRAKASH, A. FACE: A firewall analysis and configuration engine. *SAINT* (2005), 74–81.

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

[94] VILA, T., GREENSTADT, R., AND MOLNAR, D. Why we can't be bothered to read privacy policies models of privacy economics as a lemons market. In *ICEC '03: Proceedings of the 5th international conference on Electronic commerce* (New York, NY, USA, 2003), ACM Press, pp. 403–407.

[95] WHITTEN, A., AND TYGAR, J. D. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium* (1999).

[96] WOOL, A. A quantitative study of firewall configuration errors. *Computer 37*, 6 (2004), 62–67.

[97] WRIGHT, A. K., AND FELLEISEN, M. A syntactic approach to type soundness. *Information and Computation 115*, 1 (1994), 38–94.

[98] WRIGHT, C., COWAN, C., SMALLEY, S., MORRIS, J., AND KROAH-HARTMAN, G. Linux security modules: General security support for the linux kernel. In *Proceedings of the* 11^{th} *USENIX Security Symposium* (2002).

[99] YEE, K.-P. Guidelines and strategies for secure interaction design. In *Security and Usability: Designing Secure Systems That People Can Use*, L. F. Cranor and S. Garfinkel, Eds. O'Reilly, 2005, ch. 13, pp. 247–273.

[100] YUAN, L., MAI, J., SU, Z., CHEN, H., CHUAH, C.-N., AND MOHAPATRA, P. Fireman: A toolkit for firewall modeling and analysis. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 199–213.

[101] ZHANG, X., EDWARDS, A., AND JAEGER, T. Using CQUAL for static analysis of authorization hook placement. In *Proceedings of the* 11^{th} *USENIX Security Symposium* (2002).

[102] ZURKO, M. E., KAUFMAN, C., SPANBAUER, K., AND BASSETT, C. Did you ever have to make up your mind? What Notes users do when faced with a security decision. *Annual Computer Security Applications Conference 2002* (2002), 371.

Appendix 1: Usability survey questions

1. What is your user number?
2. In which order did you perform scenarios #2 and #3?

    Ask Me first, then Doppelganger

    Doppelganger first, then Ask Me
3. How many hours a week do you use a web browser?

    0-5

    5-10

    10-20

    20+
4. On a scale of 1-4, how concerned are you about your privacy online?

    1 (Not concerned)

    2 (Somewhat concerned)

    3 (Concerned)

    4 (Paranoid)
5. In particular, how concerned are you about your browsing habits and actions being recorded by the sites you visit or by third party sites?

    1 (Not concerned)

    2 (Somewhat concerned)

    3 (Concerned)

    4 (Paranoid)
6. Before this study, how familar were you with web browser cookies?

    Not at all

    I had heard of them, but didn't really understand how they worked

    I basically understood how cookies work

    I understood terms like 'persistent cookies' and 'third party cookies'
7. Have you taken any steps to manage or monitor the cookies in your browser? For example, have you deleted cookies, examined cookies, or changed your cookie preferences, or used third-party software that did so?

    yes

no

not sure

8. Were the tasks you were asked to perform similar to ones that you have performed frequently in the past?

1 (Quite dissimilar)

2 (Somewhat dissimilar)

3 (Similar)

4 (Very similar)

9. Were you able to complete all the tasks in scenario #1?

Yes

No (please elaborate)

10. How easy was it for you to complete the tasks in scenario #1?

(1) Very difficult

(2) Difficult

(3) Relatively difficult

(4) Relatively easy

(5) Easy

(6) Very easy

11. Did you encounter any problems in scenario #1 (even if you completed all the tasks)?

No

Yes (please elaborate)

12. What did you like or not like about Scenario #1?

13. Were you able to complete all the tasks in scenario #2?

Yes

No (please elaborate)

14. How easy was it for you to complete the tasks in scenario #2?

(1) Very difficult

(2) Difficult

(3) Relatively difficult

(4) Relatively easy

(5) Easy

(6) Very easy

15. Did you encounter any problems in scenario #2 (even if you completed all the tasks)?

No

Yes (please explain)

16. How well do you feel you represented Pat Smith's trust preferences during Scenario #2?

Pat's preferences made no difference to my choices because I ignored them

Pat's preferences made no difference because I wasn't sure how to use them

Pat's preferences made some difference

I represented Pat's preferences carefully in my choices

17. How did you feel about the number and difficulty of cookies management tasks and decisions you faced in Scenario #2? By "easy" we mean that it was generally clear which button to push and when, and that you knew the correct choice for each dialog box. By "difficult" we mean that it was generally unclear what was the right action in each situation.

(1) There were too many dialogs and button presses but they were easy to handle

(2) There were too many tasks dialogs and button presses and they were difficult to handle

(3) There were not too many tasks and they were easy to handle

(4) There were not too many tasks but they were difficult to handle

18. What did you like and not like about scenario #2? For example, "I found the interface too difficult to use" or "The choices presented to me were clear".

19. Were you able to complete all the tasks in scenario #3?

Yes

No (please elaborate)

20. How easy was it for you to complete the tasks in scenario #3?

(1) Very difficult

(2) Difficult

(3) Relatively difficult

(4) Relatively easy

(5) Easy

(6) Very easy

21. Did you encounter any problems in scenario #3 (even if you completed all the tasks)?

No

Yes (please explain)

22. How well do you feel you represented Pat Smith's trust preferences during Scenario #3?

Pat's preferences made no difference to my choices because I ignored them

Pat's preferences made no difference because I wasn't sure how to use them

Pat's preferences made some difference

I represented Pat's preferences carefully in my choices

23. How did you feel about the number and difficulty of cookies management tasks and decisions you faced in Scenario #3? By "easy" we mean that it was generally clear which button to push and when, and that you knew the correct choice for each dialog box. By "difficult" we mean that it was generally unclear what was the right action in each situation.

(1) There were too many dialogs and button presses but they were easy to handle

(2) There were too many tasks dialogs and button presses and they were difficult to handle

(3) There were not too many tasks and they were easy to handle

(4) There were not too many tasks but they were difficult to handle

24. What did you like and not like about scenario #3? For example, "I found the interface too difficult to use" or "The choices presented to me were clear".

25. Do you have any general comments about your experiences or ways we can improve the privacy management tools you used?

Appendix 2: Usability survey responses

General questions (I)

User #	In which order did you perform scenarios #2 and #3?	How many hours a week do you use a web browser?	On a scale of 1-4, how concerned are you about your privacy online?	In particular, how concerned are you about your browsing habits and actions being recorded by the sites you visit or by third party sites?
1	Ask Me first	5-10	3	3
3	Ask Me first	20+	2	3
4	Doppelganger first	5-10	2	1
5	Ask Me first	10-20	3	3
6	Doppelganger first	10-20	2	2
7	Ask Me first	20+	3	2
8	Doppelganger first	10-20	2	3
9	Ask Me first	20+	2	2
10	Doppelganger first	10-20	2	3
11	Ask Me first	10-20	1	1
12	Doppelganger first	10-20	3	2
13	Ask Me first	10-20	1	1
14	Doppelganger first	20+	2	2
15	Ask Me first	20+	2	2
16	Doppelganger first	10-20	2	2
17	Ask Me first	10-20	2	2
18	Doppelganger first	20+	2	3
19	Ask Me first	20+	3	4

General questions (II)

User #	Before this study, how familar were you with web browser cookies?	Have you taken any steps to manage or monitor the cookies in your browser? For example, have you deleted cookies, examined cookies, or changed your cookie preferences, or used third-party software that did so?	How many hours a week do you use a web browser?
1	I basically understood how cookies work	yes	5-10
3	I had heard of them, but didn't really understand how they worked	yes	20+
4	I basically understood how cookies work	yes	5-10
5	I understood terms like 'persistent cookies' and 'third party cookies'	yes	10-20
6	I understood terms like 'persistent cookies' and 'third party cookies'	yes	10-20
7	I basically understood how cookies work	no	20+
8	I basically understood how cookies work	yes	10-20
9	I basically understood how cookies work	yes	20+
10	I basically understood how cookies work	yes	10-20
11	I had heard of them, but didn't really understand how they worked	no	10-20
12	I understood terms like 'persistent cookies' and 'third party cookies'	yes	10-20
13	I basically understood how cookies work	yes	10-20
14	I had heard of them, but didn't really understand how they worked	yes	20+
15	I understood terms like 'persistent cookies' and 'third party cookies'	yes	20+
16	I basically understood how cookies work	no	10-20
17	I had heard of them, but didn't really understand how they worked	yes	10-20
18	I had heard of them, but didn't really understand how they worked	no	20+
19	I basically understood how cookies work	yes	20+

Default settings scenario

User #	Were the tasks you were asked to perform similar to ones that you have performed frequently in the past?	Were you able to complete all the tasks in scenario #1?	How easy was it for you to complete the tasks in scenario #1?	Did you encounter any problems in scenario #1 (even if you completed all the tasks)?	What did you like or not like about Scenario #1?
1	2	Yes	6	No	It did ask about 3rd-party tracking (liked)
3	4	Yes	6	No	
4	3	Yes	6	No	It's the easiest to use.
5	3	Yes	6	No	Seems to be a breeze to go to the next pages.
6	4	Yes	6	No	No prompts for accepting cookies. Simple browsing.
7	4	Yes	4	No	I liked how everything was remembered for me, and it was very easy to login to my email account.
8	3	Yes	6	No	
9	1	Yes	5	No	
10	3	Yes	6	No	I did not know which cookies were being accepted by my browser.
11	3	Yes	5	No	It was easy to complete the tasks once I was familiar with the pages.
12	3	Yes	6	No	Everything worked, and pretty quickly
13	3	Yes	6	No	Everything went smoothly without me having to keep okaying cookies.
14	4	Yes	6	No	Didn't have to make decisions about things I didn't know about.
15	4	Yes	6	No	The browser was configured the way I would normally have it at home or at the library.
16	4	Yes	6	No	Very simple. Excite saved all my information, which was helpful (they were at a 'high' trust level). Very easy.
17	2	Yes	6	Yes	nothing
18	2	Yes	6	No	Nothing really happened in Scenario #1.
19	2	Yes	5	No	It didn't ask me about any cookies but since firefox is generally safer than explorer, I didn't really think about it. I would usually delete all cookies from the browser's menu every week anyways.

Results for Doppelganger

User #	Able to complete tasks?	How easy to complete the tasks	Did you encounter any problems?		Represented trust preferences?	Hard Choices?	Too many choices
1	Yes	5	No		I represented Pat's preferences carefully in my choices	FALSE	FALSE
3	Yes	5	Yes	I had to choose whether or not to allow Excite to leave cookies, but I think I might have been too careful with them.	Pat's preferences made some difference	FALSE	FALSE
4	No: I didn't enable the cookies for netflix.com because the trust level is low.	3	Yes	There were too many pop up boxes.	I represented Pat's preferences carefully in my choices	FALSE	FALSE
5	Yes	4	No		Pat's preferences made no difference because I wasn't sure how to use them	FALSE	FALSE
6	Yes	6	Yes	When opening netflix.com, I was asked to choose the browser version I was currently using.	Pat's preferences made no difference to my choices because I ignored them	FALSE	FALSE
7	No: I didn't search for Shrek on netflix because I didn't enable cookies, since my trust level was low.	4	Yes	I couldn't access Netflix with my cookies disabled, so I had to use FIXME in order to search for Shrek.	I represented Pat's preferences carefully in my choices	FALSE	FALSE
8	Yes	4	Yes	Choosing between web browsers enabling cookies and not.	Pat's preferences made some difference	FALSE	FALSE
9	Yes	4	No		Pat's preferences made some difference	FALSE	FALSE

Results for Doppelganger

User #	Able to complete tasks?	How easy to complete the tasks	Did you encounter any problems?	Represented trust preferences?	Hard Choices?	Too many choices
10	Yes	6	No	Pat's preferences made some difference	FALSE	FALSE
11	No: Didn't complete netflix task. I was lost and not sure what I had to do to activate cookies and continue further into the site.	3	Yes problem with completing task 2, when I couldn't actually move forward unless I activated cookies. Or maybe I didn't have to do that, and I didn't know.	Pat's preferences made no difference because I wasn't sure how to use them	FALSE	FALSE
12	Yes	6	Yes not really a problem, but I did have to hit the fix it button to get the email to work.	Pat's preferences made some difference	FALSE	FALSE
13	Yes	5	Yes I couldn't get onto netflix without using the FIXME button.	Pat's preferences made some difference	FALSE	FALSE
14	Yes	5	No	Pat's preferences made some difference	FALSE	FALSE
15	Yes	6	No	I represented Pat's preferences carefully in my choices	FALSE	FALSE
16	Yes	6	Yes I wasn't sure what to do when I first went to the Netflix page. The page told me I had to enable cookies, but I wasn't sure I wanted to, as I didn't trust them. I tried to move around it by not altering my cookie settings, but Doppelganger did it for me (I imagine that it must be necessary to view the page)	I represented Pat's preferences carefully in my choices	FALSE	FALSE
17	Yes	6	No	Pat's preferences made no difference because I wasn't sure how to use them	FALSE	FALSE

Results for Doppelganger

User #	Able to complete tasks?	How easy to complete the tasks	Did you encounter any problems?	Represented trust preferences?	Hard Choices?	Too many choices
18	Yes	4	No	Pat's preferences made no difference because I wasn't sure how to use them	FALSE	FALSE
19	Yes	4	Yes I couldn't figure out why it kept on saying that 'there was no fix to the cookies' or something like that in my 'fixme' button and when I click on done, everything disappears, without showing me my two choices (this only ends when I restartec the browser). I don't know if that's supposed to be normal or not.	I represented Pat's preferences carefully in my choices	FALSE	FALSE

Results for Ask Me

User #	Able to complete tasks?	How easy to complete the tasks	Did you encounter any problems?		Represented trust preferences?	Hard Choices?	Too many choices
1	Yes	4	No		I represented Pat's preferences carefully in my choices	TRUE	TRUE
3	No: I chose to reject all the cookies from NetFlix and was not able to access any part of their website.	4	Yes	I could not finish Task #2, but I suppose that is a matter of choice rather than a true problem.	I represented Pat's preferences carefully in my choices	TRUE	TRUE
4	Yes	4	Yes	There were too many pop-up boxes.	Pat's preferences made some difference	FALSE	FALSE
5	Yes	4	No		Pat's preferences made no difference because I wasn't sure how to use them	TRUE	TRUE
6	Yes	5	Yes	Not really a problem but the numerous prompts to accept cookies made it a tedious effort.	Pat's preferences made no difference to my choices because I ignored them	FALSE	TRUE
7	No: Because my trust level for Netflix was low, I chose not to enable the cookies for that site. Because of that, I was denied access to the site.	4	Yes	See number 13	I represented Pat's preferences carefully in my choices	FALSE	TRUE
8	Yes	3	Yes	Continual popups for allowing/denying cookies.	Pat's preferences made some difference	TRUE	TRUE

Results for Ask Me

User #	Able to complete tasks?	How easy to complete the tasks	Did you encounter any problems?		Represented trust preferences?	Hard Choices?	Too many choices
9	No: I didn't complete the netflix task	3	Yes	I was stuck in netflix	Pat's preferences made no difference because I wasn't sure how to use them	TRUE	TRUE
10	No: I rejected a cookie from the Netflix site and was not able to complete the task.	1	Yes	I was unable to complete one of the tasks and I was annoyed by the constant popups.	I represented Pat's preferences carefully in my choices	TRUE	TRUE
11	Yes	4	Yes	Not knowing exactly what options to choose when cookies were presented.	Pat's preferences made some difference	TRUE	TRUE
12	No: netflix would not let me look up a movie without cookies; and since 'I' didn't trust the site, I wouldn't let it give me a cookie.	4	Yes	Problem with netflix stated in question 19. It was also moderately time consuming to click no to all the cookies on sites I don't trust and yes on the site I do trust.	I represented Pat's preferences carefully in my choices	FALSE	TRUE
13	Yes	4	Yes	I had to say yes to an annoying number of cookies for my most trusted site.	Pat's preferences made no difference because I wasn't sure how to use them	FALSE	TRUE
14	No: when i rejected to accept the cookies for netflix.com, I couldn't enter the website so was not able to find the Shrek movie	4	Yes	The endless cookies popups were getting to a point where I didn't even care if I could complete the task or not because it wasn't worth dealing with.	Pat's preferences made no difference because I wasn't sure how to use them	TRUE	TRUE

Results for Ask Me

User #	Able to complete tasks?	How easy to complete the tasks	Did you encounter any problems?		Represented trust preferences?	Hard Choices?	Too many choices
15	Yes	4	Yes	I had to continually click 'allow' since it asked for every cookie, especially when browsing through excite.com. It also asked to modify existing cookies. After browsing excite.com, it showed me that I had at least 10 cookies. It would have been very annoying to allow each one individually. Each new site that I entered, I was prompted to make decisions about the cookies. The option to 'allow this option for this site' (some variation of that) did not work as expected. After closing the browser and revisiting a site with a low trust level, I was prompted again to go make a decision about each individual cookie.	I represented Pat's preferences carefully in my choices	FALSE	TRUE
16	Yes	2	Yes	For some reason weather.com wouldn't open to Brooklyn the first two times I tried. That may have been because I wasn't sure whether to hit 'accept' or 'accept for this session' and may have mixed them up.	I represented Pat's preferences carefully in my choices	TRUE	TRUE
17	Yes	6	No		Pat's preferences made no difference because I wasn't sure how to use them	FALSE	FALSE

Results for Ask Me

User #	Able to complete tasks?	How easy to complete the tasks	Did you encounter any problems?	Represented trust preferences?	Hard Choices?	Too many choices
18	Yes	4	Yes Way too many pop ups regarding security loss.	Pat's preferences made no difference because I wasn't sure how to use them	FALSE	TRUE
19	Yes	4	No	I represented Pat's preferences carefully in my choices	TRUE	TRUE

Overall subjective evaluations

User #	Liked and didn't like (Doppelganger)	Liked and didn't like (Ask Me)	General comments
1	I made the choice to switch over to the 'right' browser when I returned to excite.com, but what if I wanted to go to another site for which my trust level is much lower? If the doppelganger knows this from previous visits, that would be cool. The interface was not difficult. The FIXME button was helpful but it didn't look like it was applicable when entering the netflix site.	Way too many cookie acceptance/change requests. I had no idea how many times cookies were set/reset. I really didn't like that other websites were trying to set cookies while I was on weather.com!	I found it interesting - the instructions could be a little clear. I was supposed to 'customize' the weather.com site and that wasn't in the instructions. By customizing, it didn't look like it did much anyway (except with the ask me scenario - it asked about more cookie stuff).
3	I think the Doppelganger settings are really cool, but if it were to do that every single time I wanted to browse a website, I think I would get a little bit sick of it. I liked that Doppelganger chose which cookies to accept for me.	I found it difficult to think as Pat would, as I feel that I took it to the extreme. Perhaps Pat would have allowed cookies for the session rather than rejecting them completely.	
4	It was confusing because I don't really now what FIXME/SYNCED is all about.	There were too many pop-up boxes, so it made me feel like I'm doing too much work.	
5	The choices were clearly defined.	Too many choices	All the scenarios were pretty easy to follow.
6	The interface was not difficult at all to use, but I didn't really see any significant advantage or benefit in using Doppelganger over the default browser.	The choices presented were clear; however, there were just too many prompts and after a while it definitely became annoying. Also, even after clicking the check box for 'Remeber my action for every cookie from this site,' it still prompted me for more decisions.	The tasks were all relatively simple and everything was straightfoward. I did forget that I was supposed to represent Pat Smith's actions so it probably would've helped me to see the Trust Level line arranged in a more eye-catching and noticeable orientation so I wouldn't forget.

Overall subjective evaluations

User #	Liked and didn't like (Doppelganger)	Liked and didn't like (Ask Me)	General comments
7	The interface was fine - clear and easy to use with just one button (FIXME). It was kind of confusing to see both the left and right browser at the same time, but after looking it over, it was easy to see the differences. I also liked the dialogue box between the two because it made very clear what were the risks to my privacy if I enable cookies - for that reason, I chose to disable cookies even though my trust level was high because it seemed like the risk presented by Doppelganger was too great.	The dialogue boxes asked me if I wanted to enable cookies or not, but I didn't know what they were for exactly. When I tried to access my email, it was clear they were asking for cookies to remember my login information. However, for the news, I didn't see any reason why cookies would be enable if other than for tracking what kind of stories I read.	No, it was very clear and concise.
8	Sometimes didn't know what choices to make, but otherwise pretty simple to use.	Sometimes wasn't sure if I should be allowing cookies or not.	
9	easy to use	Too difficult	no
10	I liked the interface because it was easy to use and it seemed like I was getting more protection from cookies compared to Scenario #1. Although I would have liked to have an idea what the program was syncing.	I found it wasted too much of my time and confused me as well. Even though the choices were for my own protection, I did not know what I was doing and after the session I wasn't sure wether the cookies I did accept were the right ones.	
11	I like that I was presented with options, but I didn't like that my experience with making a decision for these options wasn't there.	Even with instructions on what certain choices ment, the choices presented to me were unclear in that I wasn't sure what Pat would have wanted.	I felt like because I don't know much about computer privacy, I was doing what I thought I should be doing (even though I didn't really know), rather than what I would actually do when browsing at home
12	I found it easy to use for the level of security it seemed to be providing. It was more time consuming than I would like that first time that I used it but I expect that if I used it more, I would get more used to it and not have to stop and read dialogue boxes and so it would not be a serious impediment.	I would have preferred to just tell it which sites to ban cookies from and which to trust. Not confusing, but more clicking and reading than necessary.	Looks like a good extension. It's enough security and not too much trouble for most of what I do in the internet. I'll probably use it when it's released.
13	I understood what the dialogue boxes were asking me and there were not too many of them.	There were too many dialogue boxes.	

Overall subjective evaluations

User #	Liked and didn't like (Doppelganger)	Liked and didn't like (Ask Me)	General comments
14	I thought that weather.com interface that presented me with the choice of whether I wanted to switch to the increased functionality browswer was a little annoying and unnecessary. Because for simple tasks at hand, the increased functionality is not needed, esp. if the risk is increased. However, the added security is a comfort and lets me know that I am not completely at risk.	The interface was too difficult to use and I definitely did not understand the consequences of my actions. I felt that some of the cookies decisions could have already been memorized by the browser and did not need to pop up again and again.	Ask me scenario is too confusing and probably wouldn't be suitable for the general public unless everyone was super paranoid, and even then, I believe they would easily get tired of having to make a decision about cookies with every website they visited. I personally liked Doppelganger a lot more just because a simple yes or no would do and the consequences were clear.
15	It is reasonable to have to make a decision to allow cookies for a site, and the browser makes it easier to allow cookies with the FIXME button. A harder decision I had to make was which window to choose when I was allowed to pick the original vs. the other. Both looked exactly the same at first glance, and since I knew I only had to check my email, I chose the original. Otherwise, I appreciated the lesser amount of decisions I was required to make.	In general, it is very difficult to decide about each individual cookie consider the user may not know what it does. The option to allow for the session made me feel that the browser would not ask me for the same information again, until after I had closed the browser. Considering the trust level of Pat, it was annoying having to use a site that is not as trusted.	No.
16	Even though I messed up (tried to enter Netflix without enabling cookies), the browser took care of it for me. I assume Pat trusts his/her browser even if he/she doesn't trust the website. The choice between the two browsers when I entered excite a second time was easy and simple- they only problem I had was determining what the privacy risks were (the two bars numbered 1-10 aren't very helpful or are confusing)	So much work all the time. Even when I tried to close a browser, cookies would pop up. If I tried to go to another site, cookies would pop up. Terrible- I don't know how anyone could use that if they spent any amount of time on the web. Also, I still don't know whether to choose 'accept' or 'accept for this session' or the consequences.	Definitely an improvement if people are concerned about privacy. Make the two window choice (when Doppelganger doesn't know whether to approve cookies) a little clearer, maybe better laid out.
17	It was better then ask me because of i wasn't alwasys clicking buttons.	it was annoying to keep clicking allow for session	
18	It was easy to understand and know what to do when choices were presented to me.	I just felt the 'Ask Me' scenario presented too many pop ups. Other than that, it was relatively easy to complete this task.	

Overall subjective evaluations

User #	Liked and didn't like (Doppelganger)	Liked and didn't like (Ask Me)	General comments
19	It was easier in that during the second session of the third scenario, I get to see the two website being compared side by side. It was harder because I was not sure whether or not I did the right thing in the first session because the note/ warning box has a 'done' message and when I click on the 'fix me' button on the bottom, it gave me a 'no fix' message. The interface is quite easy and the choices were clear. I was a bit confused by the different website presented for the weather.com but I supposed that was part of the 'fix'?	I liked that they would ask me which website needs my confirmation as to the cookies but I didn't like that there were too many cookies (constantly in your face) and it took at least 3 seconds to figure out which website wanted the cookies. Furthermore the safety of these web-sites were not clearly indicated and that was annoying.	