

# Remote Harm-Diagnostics

Markus Jakobsson  
School of Informatics  
Indiana University  
Bloomington, IN USA  
markus@indiana.edu

Ari Juels  
RSA Laboratories  
Bedford, MA USA  
ajuels@rsasecurity.com

Jacob Ratkiewicz  
Dept. of Computer Science  
Indiana University  
Bloomington, IN USA  
jpr@cs.indiana.edu

## ABSTRACT

We propose Remote Harm-Diagnostics (RHD) as a new defensive tactic against malware and phishing attacks. RHD is a server-side tool that harvests data from clients to detect signs of previous or ongoing compromise. RHD can supplement client-side protections such as virus checkers and anti-phishing toolbars. Given its “soft,” i.e., probabilistic nature, RHD interfaces naturally with risk-based authentication engines.

We introduce RHD techniques that *require no installation of client-side executables*. These techniques permit servers to harvest selected Internet browsing history from visiting clients. We describe a lightweight, prototype RHD system that offers server-side detection of many forms of client compromise.

Privacy-protection being a keystone of our approach, we introduce mechanisms for client-side filtering of data. We also briefly discuss audit and policy enhancements to help RHD systems comply with regulatory guidelines like the OECD Fair Information Practice Principles.

## Categories and Subject Descriptors

K.4.4 [Electronic Commerce]: Security; D.4.6 [Security and Privacy Protection]: Invasive Software

## General Terms

Security, Malware, Privacy

## Keywords

detection, diagnostic, malware, phishing, reactive, server-side, privacy-preserving

## 1. INTRODUCTION

Internet users are facing a broadening and ever more virulent host of attacks on their computing devices and personal information. “Phishing,” the luring of users to malicious Web sites, has been an especially high-profile problem. As educational efforts mount and filtering of e-mail and web pages reduces the yield of phishing attacks, attackers are exploring other mechanisms for compromising user data. Malware plays a prominent role in this recent trend. While older viruses usually confined their misbehavior to self-propagation and relatively innocent pranks, today’s emerging generation of malware supports infrastructure-wide attacks. Some malware assimilates infected machines

into bot networks that propagate spam, perform click-fraud, and mount denial-of-service and other attacks. Other malware steals passwords for use in financial fraud, or even launches hidden sessions and performs covert financial transactions after a user has authenticated to a financial institution.

Malware commonly gains access to a victim’s computer in one of two ways. It may be installed as part of a seemingly legitimate program, making it what is called a *Trojan Horse* program or simply a “Trojan”. Alternatively, by exploiting security flaws in web browsers (notably Internet Explorer and ActiveX), malware may automatically install itself on a client that visits a malicious web site. Such *drive-by installation*, as it is called, does not require any user intervention. Users do not learn that drive-by installation has taken place until they notice the side-effects—if they ever do.

Good virus scanners, regular security updates, and prudent Internet browsing habits are effective proactive measures that, in theory, can stymie most if not all common types of malware. Malware continues to be a problem, however, because many users and even system administrators fail to run and/or update virus scanners, leave critical system components unpatched, download and install untrusted executables, and in general exercise insufficient vigilance while web surfing. Phishing, likewise, continues to be problematic as perpetrators devise ever more alluring e-mail and branch into sophisticated confidence games. In current computing environments, malware infection and phishing are perhaps just an inevitable byproduct of widespread Internet use.

In this paper, we propose an approach to phishing and malware attacks that is distinctive for being *reactive*, rather than *proactive*. Our aim, in other words, is to detect client exposure to phishing and malware attacks after the fact and to mitigate the harm that these attacks cause Internet users. The use of reactive defenses is not intended to replace proactive measures, but to provide a second line of defense—and one with no client-side footprint. We refer to our system as a *Remote Harm-Diagnostic* or RHD.

Broadly speaking, our RHD techniques involve a web server learning whether a client browser has initiated dangerous Internet connections. Such connections can include browsing of known phishing sites as well as browsing that is indicative of malware infection. We note that for some malware, evidence of infection is best obtained *indirectly*, e.g., by detecting whether the client has been directed to certain ad-bearing web sites or has been attacked by *other* malware relying on similar vulnerabilities. This is an important ob-

ervation, as it suggests that we can potentially detect the presence of sophisticated malware that takes direct action to erase its tracks.

Depending on the information that an RHD-enabled server gleans about suspected malware and on the transactional relationship between the server and a client, the server or its administrators can initiate defensive action on behalf of the client’s user. For example, an Internet bank might reject transactions conducted by customers through clients with probable malware infections, and might take additional measures like contacting vulnerable customers by mail or telephone. (Abatement measures are a policy issue outside the scope of our work.)

Our main focus in this paper is an RHD technique that requires *no client-side software*. In other words, we propose a system of malware detection that relies exclusively on server-side implementation. The advantages of this approach are clear: Our system is perfectly transparent to users and avoids the complications and burden of user-mediated software installation. Furthermore, our system does not stoke the dangerous habit in users of downloading potentially troublesome executables from the Internet.

The basis for our RHD system is a web-browser feature that we refer to as *URL probing*. Most web browsers are configured to retain what is called a *history*, a list of the URLs visited by a user in the recent past—typically the last nine days. As we explain, by means of URL probing, a server can ascertain the presence of a particular URL in the browser of a visiting client. For example, a server can determine if a visiting client has browsed the site `www.xyz.com` in the past few days. It is important to note that URL probing only reveals information in the case of *exact matching*. For example, if a client has visited `www.xyz.com/nihil`, a server will not learn this fact by probing the client’s browser history for the URL “`www.xyz.com`.”

As the research community has recognized for some years, URL probing can serve as a vehicle for privacy abuses [7, 6], as it permits servers to harvest data for which they have no legitimate right or use. Some researchers have proposed browser-modifications to restrict URL probing. For example, [10] proposes a browser modification that limits the feature according to same-origin policies. URL probing, however, exists and has persisted for years in browsers as a side effect of beneficial features. (In particular, URL probing is a side effect of features in Cascading Style Sheets (CSS) [12] that permit flexible coloring of visited links in web pages.) We emphasize that our RHD proposal does not create or exploit a new form of URL probing, but makes use of an existing one as a new security tool.

We propose several mechanisms for protecting user privacy in RHD deployments. The first, and most important is *filtering*. We describe special techniques that permit a server to learn only relevant, aggregate information about URLs in a user’s browsing history—*without communicating information about specific, visited URLs*. Furthermore, we propose opt-in mechanisms for user participation in the RHD process and describe mechanisms that permit public auditing of RHD implementations.

Another challenge in deploying an RHD system is its “soft” nature. Any natural system for detecting phishing or malware infection will necessarily yield false negatives and, more problematically, false positives. Fortunately, as we explain, many enterprises today are embracing risk-based security

technologies precisely designed to operate on the basis of uncertain authentication inputs.

Thus we show how URL probing can benefit users by helping RHD-enabled servers detect and initiate protective measures against phishing attacks and malware. Our main challenge is to devise a system that is: (1) Effective in accurately detecting phishing and malware exposure of clients, but also (2) Strongly protective of users’ privacy.

## Organization

In section 2, we describe the intuition of our approach by considering how one particular type of malware can be detected by our system. In section 3 we survey related work and background literature. We describe an initial implementation in section 4. In section 5, we discuss surrounding privacy and ethical considerations. We conclude in section 6 with some further research directions.

## 2. INTUITION BY EXAMPLE: DETECTING DOWNLOADER.TROJAN

As an example of how our RHD system might work, let us consider a Trojan program that is subject to drive-by installation, namely the executable referenced by the antivirus community as `downloader.trojan`. This Trojan, released in 2002, is relatively old. Computers that lack up-to-date virus protection, however, are still vulnerable. This Trojan has the ability to install itself from a web site without the user’s permission, making use of a security flaw in the Internet Explorer browser. Having done so, it downloads and installs other programs from other web sites without the user’s knowledge, and adds bookmarks (also referred to as *favorites*) to the user’s browser. There are two strategies for an RHD system to perform URL probing to detect the presence of `downloader.trojan` on a client. The sensor can attempt to detect browsing behavior that may have resulted in infection by the Trojan, or else it can search for browsing behavior characteristic of existing infection. In particular, the sensor can probe for:

1. **Browser visits to malware-installing sites:** The server checks to see if the user has visited any sites that are known to attempt to install the Trojan. Note that this may work only for a short time subsequent to infection, namely the length of time that the user’s browser retains entries (usually nine days).
2. **Browser visits that imply malware infection:** The server checks to see if the user has visited sites that match the Trojan’s *signature*—that is, sites that the Trojan, once installed, causes a client to visit and/or bookmark<sup>1</sup> a set of sites. The presence of a cluster of such sites is indicative of infection. Malware that triggers browser downloading of web-based advertisements will likewise produce a characteristic footprint that in some instances can be detected as well.

A naïve deployment of either approach would involve the RHD server harvesting highly specific information about user browsing habits, raising considerable privacy concerns.

<sup>1</sup>Some browsers, notably Safari, consider any site that is bookmarked to also be visited—and permanently so.

While acknowledged traces of malware might not be privacy-sensitive for users, some of the URLs associated with malware are also sites that receive actual user-originated traffic (as opposed to traffic resulting from malware infection). This is particularly so in the case of `downloader.trojan`, which bookmarks large numbers of sites with pornographic content. We show, however, that it is possible to deploy an RHD system that filters data and returns only a binary result. For example, we can design an RHD system that returns only a rough “positive” or “negative” classification of the perceived probability of infection of a given client by `downloader.trojan`, and no information about which sites reside in the client’s browser history.

### 3. RELATED WORK

Phishing countermeasures today assume a number of different forms. There are social countermeasures such as legislation and policy—which effectively deter misbehavior by establishing codes for conduct—as well as end-user education. Existing technical countermeasures can be very roughly categorized, according to their base of deployment, as infrastructural, client-side, or server-side.

While a much-promoted remedy, educational campaigns often achieve disappointing results. For example, a recent study by Whalen and Inkpen [18] has shown that people *observe* the SSL lock icon but rarely *interact* with it, in spite of broad educational efforts encouraging users to verify the establishment of secure connections before divulging sensitive information. In a similar vein, Friedman et al. [9] demonstrated the low aptitude of non-specialists for recognizing and identifying secure connections. Fogg et al. [8] determined that users often make security decisions based largely on the design of the material under consideration; this frustrates any educational efforts given the ease of copying material and spoofing messages. Jakobsson and Ratkiewicz [11] determined that the absence of personalized greetings in communication appearing to come from eBay does not substantially impact the behavior of message recipients, suggesting that at least some often-touted security features are commonly ignored by users. In some cases, education can have unintended negative consequences. For example, if a service provider raises public awareness of malware to the extent that users become willing to download “patches” from entities claiming to be associated with the service provider in question, new risks can result.

While legislation plays an important role in raising the stakes for criminal behavior, and has made notable progress in the last few years, the rising tide of attacks shows that such measures are not in themselves sufficient. While phishing is illegal in almost all jurisdictions, the borderless nature of the crime makes it hard even for closely collaborating governments to apprehend criminals. As an example, see [16] for recent statistics regarding the increase of phishing attacks on U.S. brands originating in South Korea.

Among technical countermeasures to phishing and malware, *infrastructural* changes, as potent as they can be, are often impractical to deploy. Broad changes on the Internet conflict with requirements for backwards compatibility and the difficulties of protocol migration in a distributively maintained system. Next-generation Internet infrastructure may ultimately support more robust security foundations. In the meantime, there is a need for ad-hoc tools to fill important holes. Most common and effective today are *client-*

*side* countermeasures in which software resident on a user’s computer intercepts attacks, often with the aid of server-side updates. Examples include anti-virus software and phishing filters.

On the *server side*, some of the most popular and powerful tools today are what are sometimes referred to as *risk-based authentication engines*. These engines rely on heuristic techniques to accept or reject authentication attempts—or even individual transactions—by clients. Credit-card associations and telecommunication companies rely heavily on such tools; financial-service providers are doing so increasingly for their Internet customers. Risk-based authentication engines, e.g., [2] draw on macroscopic transaction data. They also draw on user and client-specific data, such as time-of-use records, geographical data, the presence of authenticators like cookies in browsers, and even the specifics of a given, requested transaction. Our RHD techniques can serve as a natural complement to risk-based authentication engines. They furnish an additional factor by which to assess the risks associated with a given client, and draw on more fine-grained data than cookies or login times. In this sense, our work follows guiding principles like those of [15], who propose a new form of ad-hoc authenticator in the context of pharming detection.

Our RHD techniques rely on server-side probing of client browser caches to determine what URLs a user has visited. Such probing was first studied by Felten and Schneider [7], who considered invasive forms of timing analysis. SecuriTeam [6] later described more accurate privacy-infringing techniques that directly access browser history files. In the vein of more recent work [14], our RHD work explores the flip-side of browser probing, namely how it can create *benefits* for users.

Our proposed tool is effectively a remote security-scanning service. In this sense, it is akin to a range of Internet services that scan hosts for potential vulnerabilities resulting from dangerous port configurations. A commercial example of such online tools is distributed by Qualys [1]; SAINT [3] is a command-line version of the same type of tool. Remote scanning is related in principle to identification of client machines, which is often done by means of cookies. Cookies are developed for and used to establish browsing patterns. First party cookies are commonly used to identify repeat visitors, and customize their experience based on past actions; third-party cookies and first-party cookies with an aggregator (such as Double-click) are used to track browsing patterns across domains. In contrast, RHD does not identify users, but simply attempts to identify browsing patterns and online accesses.

A number of widely deployed systems relay potentially sensitive client information to centralized data repositories. For example, the Microsoft operating system relays information on software failures to Microsoft; the system explicitly prompts users for permission before doing so. A number of toolbars, such as the Alexa toolbar and the Google toolbar, perform centralized harvesting of the browsing behavior of individual users. Users assent to the transmission of this information by merit of their installing the toolbars; all browsing activity is associated with a client-specific identifier, communicated and recorded. In comparison, our tracking is substantially less invasive, and can—where appropriate—dispense entirely with identities or pseudonyms. More importantly, our techniques involve dynamic *filtering* of data

on the client to ensure protection of user privacy, in the spirit of academic proposals like, e.g., [13].

## 4. IMPLEMENTATION

### 4.1 Intuition

In broad terms, our method is as follows. When the user requests a page from an RHD server, the server returns a page that, in addition to the normal content, defines an invisible section of the page containing a number of links. Each of these links is given a CSS style that causes the browser to “call home”—that is, to notify the server if the link has been visited. This is implemented by directing the browser to download a CSS style sheet from the server in order to color visited links. The server is then able to tell which links are visited by tracking which style sheets are downloaded. Due to the fact that browser caching normally prevents the same URL from being fetched twice in a page load, user privacy can be preserved by a careful partitioning of the links to be detected into classes, each of which is given the same “call home” link.

### 4.2 Proof-of-concept details

We first outline the server-side infrastructure, then how detection is carried out on the client side. Of particular interest are our techniques for protecting client privacy during the RHD process. In our exposition, we assume an RHD server that requires users to authenticate and log in, as would be the case for, e.g., a financial-services web site. RHD scanning takes place after<sup>2</sup> the user has logged in; the aim is to prevent users from performing potentially damaging actions (that might be subverted by session hijacking, for instance) if it is determined that they are compromised. On detecting an at-risk user, a server might suspend access to certain account features or even temporarily disable the user’s account. Again, though, we emphasize that RHD provides an indication of risks like malware infection, not a definitive litmus test.

Our server-side RHD implementation consists of three scripts:

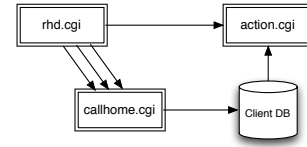
**rhd.cgi** This script produces a page that invisibly performs detection, refreshing when complete to the page that shows the results (**action.cgi**).

**callhome.cgi** This is the script that is notified when a class of links is triggered. It is responsible for keeping track of the classes of links triggered by a particular client.

**action.cgi** This script is called when the logged-in user attempts to perform another action. It consults the database maintained by **callhome.cgi** to see if the user has triggered any dangerous classes; if she has, it may redirect her to a warning page or perform some other action.

We now discuss each of these scripts in turn. Figure 1 illustrates the sequence of events in a single detection session.

<sup>2</sup>It is possible to perform detection before login as well, but this complicates liability and privacy issues, as it makes it harder to determine that a client has opted in before scanning his or her machine.



**Figure 1: Sequence of events in a single detection session.** **rhd.cgi** produces a page which is interpreted by a user’s web browser. CSS code in the web browser causes communication with **callhome.cgi**, which registers any hits in a database associated with each client. When the user visits **actions.cgi**, this database is consulted to make a final determination as to the possibility that the client is contaminated with malware.

#### 4.2.1 rhd.cgi

This script is responsible for performing the browser reconnaissance, which it accomplishes as follows. It first reads a description of the links it should detect from an XML file on the server. This file defines a number of link classes, each with a unique name and a type, which may be “AND” or “OR.” Each link class also contains a list of URLs. If the type of the class is “AND,” the class registers a hit when all of its associated URLs have been visited by the client’s browser; if its type is “OR,” it registers a hit when any non-zero number of its links are visited. Note that each class registers a binary result, either no hit or one hit, regardless of the number of visited links in the class which. How each class of links is detected is outlined below; further details on the link-detection techniques we employ are available in [12].

In each of the following discussions, we assume that each user has a unique ID number. In our demo system this is computed by incrementing a persistent ID, but other unique numbers (such as a customer ID number) would serve equally well. This ID is indicated in URLs by **XX**. Figure 2 gives a graphical depiction of each scenario.

##### 4.2.1.1 Detecting an OR-class..

Suppose that the detection XML file specifies a class of type OR that contains the links **www.a.com**, **www.b.com**, and **www.c.com**. Thus, this class should produce a single hit if any of these three sites has been visited by the user. This is implemented by the following CSS and HTML code, which is automatically generated by **rhd.cgi**:

```
<style type="text/css">
#someclass:visited {
  background: url(
    'http://www.rhdserver.com/callhome.cgi?id=XX&
      class=someclass&type=or' );
}
...
</style>
...
<a href="http://www.a.com" id="someclass"></a>
<a href="http://www.b.com" id="someclass"></a>
<a href="http://www.c.com" id="someclass"></a>
```

When the HTML-rendering subsystem of the user’s browser renders the page, it consults the CSS code to determine how to format the first visited link (if any of the links are visited).



(a) **Detecting links in an AND class.** The server is signalled by the first *un*-visited link. Note that at most one signal will be received, so the server will not know which (or how many) links were unvisited.

(b) **Detecting links in an OR class.** The server is signalled by the first visited link. Note that subsequent visited links will not signal the server, as they will be handled by browser cache; thus, the server cannot tell how many of the links were visited.

**Figure 2: Privacy-preserving detection of visited links by means of link classes. In each case, the server can tell only if the conditions of the class are satisfied; not the degree to which they are (or are not). Note that these strategies are through application of a variant of De Morgan’s laws for propositional formulae.**

This results in the URL given as the link background being loaded, causing `callhome.cgi` to be called. This registers with the server the fact that one of the links was visited, but the server does not learn *which* link. Even if the user has visited multiple links in this class, the client will load the link-background URL at most one time; once loaded, the link persists in the browser’s cache. Thus, the server will also never know how many of the links in the class the user has visited, and the user’s privacy is preserved.

#### 4.2.1.2 Detecting an AND-class..

Preserving the user’s privacy in this case is slightly more involved. Suppose in this example that we wish to tell if a user has visited sites `www.x.com`, `www.y.com`, and `www.z.com`. We wish the server to be notified if the user has been to *all three* of the sites; if the user has been only to some subset of the sites, we wish to know nothing. The following CSS and HTML perform this detection with a class named `anotherclass`:

```
<style type="text/css">
#anotherclass:link {
  background:url(
    'http://www.rhdserver.com/callhome.cgi?id=XX&
    class=anotherclass&type=and' );
}
...
</style>
...
<a href="http://www.x.com" id="anotherclass"></a>
<a href="http://www.y.com" id="anotherclass"></a>
<a href="http://www.z.com" id="anotherclass"></a>
```

Note that the RHD server receives exactly one request if any of `x.com`, `y.com`, or `z.com` are *un*-visited. This is in contrast to the detection of OR-classes, in which the server is notified when links are visited. If the server receives a request, the server knows that *at least one* (and possibly all) links are unvisited, and thus the class is not a hit. If

it receives no request, it knows that all links are visited. The fact that the class is a hit when the server receives *no* requests means that the server must adopt a “guilty until proven innocent” approach in determining if the class is a hit. Thus, if a user’s browser crashes, or the user navigates away from the page, before detection is complete, the class might erroneously be marked as a hit. We accept this possibility because RHD is by nature a probabilistic detection, and we consider the probability of these false positives to be relatively low given the speed of rendering. It should be noted that in this case, if the class is a hit the server knows exactly which links the user has visited; namely, all the links in the class. However, this knowledge does not violate the user’s privacy if the class is chosen carefully. This is due to the fact that while one or two of the links being visited may mean the user visited those sites of his own volition, all of them being visited is much more likely to mean that the sites were visited through the actions of malware, as the sites are not related otherwise.

#### 4.2.2 `callhome.cgi`

This script is triggered by all CSS styles. It simply records in a database the user’s unique ID, as well as the set of classes that the user has triggered.

#### 4.2.3 `action.cgi`

Called when detection is finished, this script consults the database built by `callhome.cgi` to see if the client is deemed to be at risk.

### 4.3 Implementation Alternatives

#### 4.3.1 Error-handling for RHD:

For server-side detection of client visits to certain types of sites, we can employ a special technique. This technique is only usable if the appearance of a target site is different for clients that have previously visited it (or entered credentials). The technique would work, for example, for an email portal or a financial site that displays a special portal page for users that have recently logged in (from clients with appropriate cookies). For target site `a.com`, the steps are as follows:

1. Create a `<script>` tag that draws its source from `a.com`.
2. Register a custom error-handler that the browser calls when errors occur during JavaScript parsing.
3. The user’s browser interprets the source of `a.com` as JavaScript; since it is not JavaScript, but HTML, errors occur and the custom error-handler is called.
4. Based on the location and type of the errors, the custom error-handler may be able to determine whether the generic version or the customized version of the site was shown.

Note that the custom error-handler described in step 4 would need to be hand-written for each potential site to be recognized; thus, this technique is much more limited and special-purpose than the more general one described in previous parts of this section. Similarly with the above techniques, we can use this method to formulate AND and OR predicates for sites that are able to be detected.

### 4.3.2 Fuzzy RHD:

At the cost of some loss in privacy, we can broaden our privacy-preserving RHD techniques that a server can determine if a client has visited at least  $k$  in a target list of  $n$  sites. By returning a randomized number of spurious hits to the list, a client-side script can conceal the exact number from the server. (This technique resembles a number of other statistical privacy-preserving approaches, e.g., [4].)

Let  $U = \{u_1, \dots, u_n\}$  be a set of target URLs. Let us suppose that the RHD server wishes to detect clients that have visited at least  $k$  URLs in  $U$ . The server sends the client a script that does the following:

1. Makes a call to X.cgi (a client-specific CGI script) for each  $u \in U$  that the client has visited.
2. Makes  $r \in_U [0, m]$  additional calls to X.cgi.
3. Accepts a single query  $r'$  from the server, and returns '1' if  $r \geq r'$  and '0' otherwise.

On receiving  $R > k$  total hits to X.cgi from a client, the server computes  $r' = R - k$ , and sends the query  $r'$  to the client. The server determines that the client has visited  $k$  links in  $U$  if the client returns a '1.'

The degree of privacy preservation in this scheme depends on the number of client visits  $v$  to sites in  $U$ , as well as the value of  $m$ . (It also depends on any *a priori* server knowledge about  $v$  for a given client.) In general, a client enjoys the minimum degree of privacy when  $v = k$ ; in that case, the server learns  $v$  exactly with probability  $1/m$ . We believe, however, that  $m$  can be made fairly large—on the order of thousands—thereby acceptably minimizing exposure of client data. Of course,  $r$  can be drawn from a non-uniform probability distribution to meet particular privacy needs.

### 4.3.3 Non-standard browsers:

In addition to the above browser-independent approach, we outline some features of particular browsers that can enhance detection on these browsers.

- Internet Explorer version 6 (but not version 7) allows CSS descriptors to call JavaScript functions. This allows finer-grained detection (such as the “fuzzy RHD” described above) without any compromise in privacy, as all computation can be done on the client side.
- In Safari, a link which is bookmarked counts as permanently visited (while a normally-visited link reverts to unvisited after a few days). This is beneficial to malware detection efforts, as malware that affects browsers often adds sites as bookmarks as well as visiting them. Under Safari, such sites would be detected even long after the initial malware installation due to this feature.

## 5. PRIVACY

A common approach to protection of consumer data is anonymization (or pseudonymous identification). This approach is impractical, however, for many of the environments in which RHD can be most beneficially deployed. Online financial and e-commerce web sites, in particular, require strong authentication of user identities. As we have explained, therefore, data-filtering is the crux of our approach

to privacy. The granularity and exact nature of RHD filtering may of course be adjusted to meet various security and privacy needs, and ultimately depends upon the policy of an RHD deployer. We propose two other important measures, however, that empower users to monitor and control their privacy in an RHD environment:

1. **User notice and consent:** We recommend that RHD be deployed on a strict opt-in basis. Potential users should be notified of the goals and operating parameters of an RHD system before they are enrolled in an RHD program. Only when an informed user has explicitly consented to enrollment in writing or on a web form should RHD scanning of her browser take place. Furthermore, we believe that a server should perform RHD scanning of the browser of a consenting user only after adequately authenticating the user as well as her client machine. This requirement helps ensure that an RHD deployment does not transgress the bounds of user consent by scanning a client machine or operating environment that does not belong to the user.<sup>3</sup> Institutions or web services wishing to deploy RHD might choose to offer adopting users incentives (such as a bonus to their account, or a service upgrade).
2. **Auditability:** Because our RHD system harvests information through a (sandboxed) executable loaded onto the client, *any client can determine exactly what information the RHD system gathers and reports*. Most users of course lack the motivation and/or expertise to reverse-engineer RHD executables. Detection of widespread abuse in an RHD system, though, requires that only a small number of users monitor its behavior. RHD executables can be short and transparent.

### 5.0.3.1 Fair Information Practice Principles..

The Organization for Economic Co-Operation and Development enunciated a set of eight basic privacy principles in 1980 [17] that have served as a basis for a number of subsequently published Fair Information Principles (FIP), such as the EU privacy directive and the FIP guidelines of the United States Federal Trade Commission (FTC). Appendix A enumerates the eight principles in an excerpt from the original OECD document. These principles are an excellent touchstone for gauging the privacy properties of an RHD system.

We believe that RHD may be readily deployed in compliance with the OECD principles and related FIPs. Referring the reader to Appendix A, we note that our filtering approach helps support the Collection Limitation and Use Limitation Principles by strictly limiting the amount of information harvested by an RHD server. By revealing exactly what information the RHD system scans and collects, the auditability property of RHD supports the Accountability and Purpose Specification Principles. Compliance with the remaining OECD principles requires careful notice and consent for consumers, as well as careful data security practices around RHD-enabled servers.

<sup>3</sup>Prior authentication also helps protect against denial-of-service attacks in which an attacker causes a user's machine to appear infected when it is not.

## 6. CONCLUSION

Our techniques rely on detecting the presence of particular URLs in the browser history of a client. If commonly deployed, it is reasonable to expect that attackers may attempt to evade our RHD techniques by use of dynamically changing URLs: If different users visit different URLs, an RHD server cannot determine which URLs to try to detect. Thankfully, phishers and malware distributors rely to a certain extent on static URLs today in order to promulgate their web-site addresses through search engines, bulletin boards, etc. If RHD becomes popular, however, savvy malware writers and phishers can look to “personalize” the URLs they broadcast to their victims, as well as the URLs of any URLs visited *as a result* of infection.

RHD techniques might still be valuable in a new incarnation. Malware and phishing detection can be performed using a level of indirection. For example, suppose that an RHD server aims to detect the presence of a piece of malware  $M_1$  on a given client machine, but  $M_1$  is associated with a URL that is “personalized” to evade detection. Instead, the RHD server can attempt to detect a second type of malware  $M_2$  on the same machine, where  $M_1$  and  $M_2$  exploit the same vulnerability to achieve infection.

RHD techniques can also be extended into client-side software, such as toolbars as plug-ins. In this case, RHD software can draw on many more sources of data and use arbitrarily sophisticated filtering techniques. Moreover, while client-side software is subject to malware corruption, tools like forward-secure audit logs [5] are useful countermeasures. For example, if a client maintains a log of all downloaded software, an RHD server can detect the presence of malware  $M$  on finding: (1) A log entry recording the downloading of a version of  $M$ , or (2) A modified or missing log.

We do not claim that our techniques are a silver bullet in the fight against phishing and malware infections. It is our opinion that there exists no technical panacea to either problems. Rather, good protection is likely to consist of a *collection* of measures, many of them heuristic. We believe that our techniques fill an important void and complement existing countermeasures.

## 7. REFERENCES

- [1] Qualys vulnerability management tools, 2006. Referenced 2006 at [www.qualys.com/products/trials](http://www.qualys.com/products/trials).
- [2] RSA consumer solutions: Anti-phishing, authentication, and transaction monitoring, 2006. Online service description. Referenced 2006 at <http://www.rsasecurity.com/node.asp?id=3017>.
- [3] SAINT network vulnerability scanner, 2006. Product description. Referenced 2006 at [www.saintcorporation.com](http://www.saintcorporation.com).
- [4] AGRAWAL, R., AND SRIKANT, R. Privacy-preserving data mining. In *ACM SIGMOD Conference on Management of Data* (2000), ACM Press, pp. 439–450.
- [5] BELLARE, M., AND YEE, B. Forward security in private-key cryptography. In *CT-RSA* (2003), Springer-Verlag, pp. 1–18. LNCS no. 2612.
- [6] CLOVER, A. Timing attacks on Web privacy (paper and specific issue), 20 February 2002. Referenced 2006 at <http://www.securiteam.com/securityreviews/5GP020A6LG.html>.

- [7] FELTEN, E. W., AND SCHNEIDER, M. A. Timing attacks on Web privacy. In *ACM Conference on Computer and Communications Security* (2000), ACM Press, pp. 25–32. Referenced 2006 at <http://www.cs.princeton.edu/sip/pub/webtiming.pdf>.
- [8] FOGG, B., SOOHOO, C., DANIELSON, D., MARABLE, L., STANFORD, J., AND TAUBER, E. How do users evaluate the credibility of web sites?: a study with over 2,500 participants. In *Proceedings of the 2003 Conference on Designing For User Experiences* (2003), ACM Press, pp. 1–15.
- [9] FRIEDMAN, B., HURLEY, D., HOWE, D., FELTEN, E., AND NISSENBAUM, H. Users’ conceptions of web security: a comparative study. In *CHI ’02 extended abstracts on Human factors in computing systems* (2002), ACM Press, pp. 746–7.
- [10] JACKSON, C., BORTZ, A., BONEH, D., AND MITCHELL, J. Web privacy attacks on a unified same-origin browser. In *15th Intl. World Wide Web Conference* (2006). To appear.
- [11] JAKOBSSON, M., AND RATKIEWICZ, J. Designing ethical phishing experiments: A study of (rot13) rnl query features. In *Proceedings of The 15th annual World Wide Web Conference (WWW2006)* (2006).
- [12] JAKOBSSON, M., AND STAMM, S. Invasive browser sniffing and countermeasures. In *Proceedings of The 15th annual World Wide Web Conference (WWW2006)* (2006), pp. 523–532.
- [13] JUELS, A. Targeted advertising ... and privacy too. In *RSA Conference – Cryptographers Track (CT-RSA)* (2001), D. Naccache, Ed., Springer-Verlag, pp. 408–424. LNCS no. 2020.
- [14] JUELS, A., JAKOBSSON, M., AND JAGATIC, T. Cache cookies for browser authentication (extended abstract), 2006. To appear.
- [15] JUELS, A., JAKOBSSON, M., AND STAMM, S. Active cookies for browser authentication, 2006. Manuscript. Referenced 2006 at [www.ravenwhite.com](http://www.ravenwhite.com).
- [16] KOREA INTERNET SECURITY CENTER. Korea phishing activity trends report, 2006. Referenced 2006 at [www.antiphishing.org/reports/200606\\_KoreaPhishingActivityReport\\_Jun2006.pdf](http://www.antiphishing.org/reports/200606_KoreaPhishingActivityReport_Jun2006.pdf).
- [17] ORGANIZATION FOR ECONOMIC CO-OPERATION AND DEVELOPMENT. OECD guidelines on the protection of privacy and transborder flows of personal data, 1980. Accessed 2006 at [http://www.oecd.org/document/18/0,2340,en\\_2649\\_34255\\_1815186\\_119820\\_1\\_1\\_1,00.html](http://www.oecd.org/document/18/0,2340,en_2649_34255_1815186_119820_1_1_1,00.html).
- [18] WHALEN, T., AND INKPEN, K. Gathering evidence: use of visual security cues in web browsers. In *Proceedings of the 2005 Conference on Graphics interface* (2005), ACM International Conference Proceeding Series, vol. 112, pp. pp. 137–144.

## APPENDIX

### A. OECD FAIR INFORMATION PRACTICE (FIP) PRINCIPLES

In its 1980 Guidelines on the Protection of Privacy and Transborder Flows of Personal Data [17], the OECD enunciated eight basic principles for data privacy. These have served as a basis for a number of other regulatory guidelines,

which are often referred to generically as Fair Information Principles (FIP). The United States Federal Trade Commission FIP and the EU Privacy Directive draw on the OECD document. The eight principles, drawn verbatim from the OECD document, are:

1. *Collection Limitation Principle*: There should be limits to the collection of personal data and any such data should be obtained by lawful and fair means and, where appropriate, with the knowledge or consent of the data subject.
2. *Data Quality Principle*: Personal data should be relevant to the purposes for which they are to be used, and, to the extent necessary for those purposes, should be accurate, complete and kept up-to-date.
3. *Purpose Specification Principle*: The purposes for which personal data are collected should be specified not later than at the time of data collection and the subsequent use limited to the fulfilment of those purposes or such others as are not incompatible with those purposes and as are specified on each occasion of change of purpose.
4. *Use Limitation Principle*: Personal data should not be disclosed, made available or otherwise used for purposes other than those specified in accordance with [the previous principle] except:
  - (a) with the consent of the data subject; or
  - (b) by the authority of law.
5. *Security Safeguards Principle*: Personal data should be protected by reasonable security safeguards against such risks as loss or unauthorised access, destruction, use, modification or disclosure of data.
6. *Openness Principle*: There should be a general policy of openness about developments, practices and policies with respect to personal data. Means should be readily available of establishing the existence and nature of personal data, and the main purposes of their use, as well as the identity and usual residence of the data controller.
7. *Individual Participation Principle*: An individual should have the right:
  - (a) to obtain from a data controller, or otherwise, confirmation of whether or not the data controller has data relating to him;
  - (b) to have communicated to him, data relating to him:
    - within a reasonable time;
    - at a charge, if any, that is not excessive;
    - in a reasonable manner; and
    - in a form that is readily intelligible to him;
  - (c) to be given reasons if a request made under subparagraphs(a) and (b) is denied, and to be able to challenge such denial; and
  - (d) to challenge data relating to him and, if the challenge is successful to have the data erased, rectified, completed or amended.
8. *Accountability Principle*: A data controller should be accountable for complying with measures which give effect to the principles stated above .