Active Cookies for Browser Authentication

Ari Juels[1], Markus Jakobsson[2], and Sid Stamm[3]

[1] RSA Laboratories and RavenWhite Inc.
e-mail: ajuels@rsasecurity.com, ari@ravenwhite.com
[2] University of Indiana – Bloomington and RavenWhite Inc.
e-mail: markus@indiana.edu, markus@ravenwhite.com
[3] University of Indiana – Bloomington
e-mail: sstamm@indiana.edu

28 April 2006

Abstract. We propose *active cookies* as a tool for stronger user/client authentication on the Web. An ordinary cookie is automatically released to any server associated with a particular domain name. It is therefore vulnerable to capture by pharming, that is, spoofing of domain names. An active cookie, by contrast, resists such pharming attacks.
Active cookies rely on a new protocol we propose that channels client communications to a specific, valid IP address. This protocol exploits a combination of cookie-based (or cached-object-based) authentication with a new type of IP-tracing protocol. This IP-tracing protocol helps defend against the presence of an attacker in the loop during an authentication session, but is unaffected by IP-address changes in clients between sessions.
Active cookies are fully transparent to users. They require no explicit installation or behavioral changes from users, and thus avoid the security risks and deployment complications of plug-ins, new login procedures, and standalone applications. While active cookies have functional and security limitations, we believe that they are an attractive countermeasure to a range of phishing and pharming attacks and a useful complement to existing techniques for user authentication. We demonstrate the practicality of active cookies through experimental implementation.

Keywords: active cookie, authentication, pharming, phishing, Web browser

1 Introduction

Online identity theft is a problem not merely of growing size, but of multiple and changing facets. A major contributing factor is pervasive reliance of passwords as a user authentication mechanism on the Internet. Users tend to choose passwords susceptible to brute-force attacks, a fact recognized by the research community for years [13, 20]. In practice, Web browser and e-mail interfaces and infrastructure have proven a still weaker link, as the emergence of *phishing attacks* has demonstrated. Such attacks involve deceptively attractive e-mail messages that direct users to Web pages falsely representing trusted institutions. (For example, a customer of XYZ Bank might receive e-mail with an embedded link that reads “XYZ Bank Login,” but in fact points to the IP address of a server operated by fraudsters.)

Many financial institutions have sought to combat phishing by educating users about dangerous surfing habits. The research community has responded with a range of tools that strengthen passwords against direct capture by phishers. Attackers, however, have adapted to warnings and countermeasures with new ploys. Compromise of consumer PCs through malicious code in the form of viruses and Trojan horses is on the rise. Another emerging threat to consumer security is *pharming*, an attack that achieves malicious redirection of user traffic even when a user enters a valid domain name into a browser. In this paper, we use the term pharming to refer exclusively to attacks in which an adversary corrupts the process of domain lookups outside a valid client—via DNS (Domain-Name

Service) poisoning, for instance. (We do not use pharming to refer to browser redirection attacks initiated by malware—nor can our techniques defend against such attacks.)

Identity theft being a many-headed problem, a number of online services are embracing more holistic forms of user authentication and fraud detection. It is popular now for banking sites to display user-selected pictures to help consumers confirm that they are interacting with the server of their bank, rather than that of a phisher or pharmer. Financial institutions are also increasingly using fraud-detection engines that analyze broad contextual information and combine authentication factors to render policy decisions around user authentication and transaction execution. They have also swiftly adopted new user-authentication tools. In addition to passwords, banking sites today regularly require users to answer challenge questions, e.g., "What is your mother's maiden name?" as a form of supplementary authentication.

In addition to knowledge-based forms of authentication ("something you know"), hardware tools for consumer authentication are growing in popularity. Some systems transmit transaction confirmations to mobile phones via SMS; similarly, Google has recently come to employ mobile phones as a means of registering new users for their GMail service. Certain financial institutions—particularly in Europe and Asia, and increasingly in the United States—furnish their customers with hardware authentication tokens, such as RSA Security's SecurID [9].

Yet another arsenal of techniques for consumer authentication involves authentication of the home computing environments of users. Many systems look for properties in the originating IP addresses for login requests of individual users; for example, they may check whether the IP address is associated with the home geographical area of the user. Servers also commonly store authenticating information on Web browsers. Servers commonly use cookies for this purpose. Given that many users block cookies as a privacy precaution, however, Flash objects and other forms of shared client-server state often serve as substitutes.

Each of these approaches has pros and cons. Challenge questions are sometimes more memorable than passwords, but are vulnerable to phishing and attacks involving mining of public databases [7]. Cookies and their alternatives are largely resistant to basic phishing attacks. As they are accessible, however, to any server in the domain that set them (or an attacker than can hijack or simulate a domain), they are vulnerable to pharming attacks. Furthermore, cookies identify browsers rather than users; they cannot help authenticate roaming users, i.e., users that change browsers.

Short of an infrastructure of trusted computing devices, there is no panacea in sight for the problems of on-line identity theft. Rather, the data-security community continues to develop and deploy an ever-changing palette of authentication techniques.

Our contribution

In this paper, we propose a new form of user authentication that we call an *active cookie*. Our active-cookie scheme repurposes ordinary cookies (and other browser-cached objects) within a new, dynamic authentication protocol. We use the term active cookie to refer to an ordinary cookie when it is configured for our scheme. Our active-cookie protocol can provide special security properties that ordinary cookie deployment cannot. For example, an active cookie can help defend against *pharming* attacks, while basic use of a cookie—even one containing cryptographic authentication data—essentially offers no resistance to such attacks.

We envision active cookies as particularly helpful in strengthening server authentication of clients alongside invocation of SSL (Secure Sockets Layer) sessions. Active cookies provide security properties different from and complementary to those of SSL. Unlike SSL, an active cookie helps a server ensure that there is no active man in the middle of a browser session. In principle, SSL provides some protection against man-in-the-middle attacks: When a certificate does not match the domain name that a user is visiting, a browser typically displays a warning. Many users, however, ignore such warnings, so that the effective protection is limited.[1]

Active cookies rely on two essential ideas:

1. **IP-tagged cookies:** Cookies and other browser content are usually *tagged* with the domain name of the servers that are to access them. For example, a cookie might be tagged in a browser with the domain name www.ironcladbank.com. In this case, the cookie is made available to the server *only* when the browser connects to an IP address that the browser is told corresponds to this domain. Our scheme relies instead on *a cookie tagged with an IP address* that is associated with a valid authenticating server. This form of tagging is important. It means that an attacker capable of spoofing a domain name—e.g., mounting a pharmer attack that causes the domain name www.ironcladbank.com to be directed to a hostile server—does not have the privileges required to access the cookie. In other words, to protect our active cookie scheme against hostile access, we rely on the fact that spoofing an IP address is often not a practical threat today (while spoofing a domain name is).
2. **A new form of IP tracing:** Our goal in preventing attacks like pharming is larger than mere browser authentication: We want to ensure that there is no attacker in the loop during an authentication session. To achieve this goal, we employ a new and special form of IP tracing. Standard IP-tracing methods rely on geographical location, ownership, or session-to-session consistency of IP addresses to help authenticate client machines. In contrast, active cookies are agnostic to such features of IP addresses. Our techniques aim simply to ensure consistency in IP-address presentation by a client during a particular authentication session. This aspect of active cookies is one of their salient practical benefits. Active cookies are unaffected by the IP-address changes that clients frequently undergo in computing environments today.
   Briefly stated, our active-cookie scheme performs an IP-address check as follows. A user establishes a normal connection to the desired server $\mathcal{S}$ over a normal HTTP channel, and with a normal DNS lookup. We call this normal communication channel a *soft* channel. (It is vulnerable to domain-name spoofing attacks and the like.) The server $\mathcal{S}$ then redirects the user's browser to a fixed IP address $IP_{\mathcal{S}}$, i.e., a predetermined and therefore trustworthy IP address for the server. We call this new channel a *hard* channel. The server $\mathcal{S}$ checks that the IP addresses of the client for both the soft channel and the hard channel appear to be consistent. This check helps assure that there are no malicious entities in the loop. At this point, the server $\mathcal{S}$ can verify the presence of a valid active cookie on the hard channel. The client and server can subsequently invoke SSL on the soft channel if desired, or invoke SSL in a different browser window.

Because our active-cookie scheme relies on a combination of server-initiated redirection and a conventional cookie, it requires *no explicit user involvement.* Indeed, the user need not even be aware of the presence of an active cookie in her browser.

Active cookies can help defend against a range of phishing and man-in-the-middle attacks, including DNS (Domain Name Server) corruption. Thus, they promise to strengthen existing Web systems and can complement many SSL deployments. Active cookies do not provide security as strong as purpose-built cryptographic protocols for clients, such as client-side SSL certificates. Our active cookie protocol *can be defeated by an adversary that controls communication traffic in the neighborhood of the client or server.* We mean here an adversary that has seized active control of

[1] Sometimes, attackers can even suppress certificate-mismatch warnings. It is generally difficult for a pharmer to obtain the private key corresponding to a legitimate SSL certificate for a domain under attack. Pharmers have been known to obtain fraudulent certificates, however [19]. And in principle a pharmer can dupe a user into installing an invalid certificate in her browser.

network routers and can therefore modify packets, e.g., change their destination IP addresses and cause them to be re-routed. In contrast, a client-side SSL certificate or other broad-based cryptographic system for mutual authentication would defend against such attack. The main benefit of active cookies is that they can strengthen ordinary client-server systems with *minimal overhead*. Active cookies involve almost no change in user experience or client performance, and they are easy for servers to administer.

Organization

In section 2, we briefly review the security literature and existing practices related to active cookies. We present our basic active-cookie protocol in section 3, along with an informal security analysis in which we consider the effects of phishing, pharming, and malware attacks. We describe our experimental implementation and surrounding issues in section 4, and explore variant schemes in section 5. We conclude in section 6. We discuss basic countermeasures to malware in appendix A, and some extensions of our ideas in appendix B.

2 Related Work

Given the complex and evolving nature of phishing, pharming, and other forms of on-line identity theft, researchers have proposed a range of distinctly different solutions.

2.1 Filtering

One approach is to limit the degree to which users are exposed to malicious content. Phishing/spam filters aim to cull malicious e-mail. Some are bundled with anti-virus software, others associated with traditional spam filters. E-mail filters rely on centrally produced signature files, peer-to-peer lists (whether white, grey, or black), or locally maintained Bayesian filters. All are vulnerable in differing degrees to attacks in which a phisher probes the system for holes in its filter. A different approach seeks to authenticate e-mail messages, as many phishing attacks rely on falsified sender information in e-mail. Examples of e-mail authentication efforts include the commercial Sender ID [3] and DomainKeys [1] projects, as well as academic efforts like that of [4].

A different approach is to filter and validate content at the browsing stage by means of toolbars or plug-ins that alert users to Web sites of concern. Such toolbars can classify sites according to a centralized list or based on the local behavior of the user. Good examples are the widely deployed Google toolbar and eBay toolbar, as well as the embryonic Trustbar [2]. Toolbars are not an alternative to spam filtering, but a complement.

2.2 Server-to-user authentication

Rather than relying on user-installed tools or new infrastructure, an increasingly popular practice is for servers to authenticate themselves directly to users. Many financial institutions now assign pictures to individual accounts (e.g., animals, landscapes, automobiles) and ask users to create accompanying text descriptions. These *visual keys*, as we call them, are displayed to assure a user that she has logged onto a valid site rather than the site of a phisher, and that it is safe for her to enter her password. Of course, visual keys present something of a Catch-22. It is not safe to display a user's visual key until she has been authenticated; otherwise a phisher can learn the key. On the other hand, the user must see the key *before* she authenticates herself in order to ensure she is not being phished. To resolve this problem, Web sites display visual keys only after successfully achieving some primary form of user or computer authentication, e.g., recognizing a cookie [14], but before the user enters her password. Active cookies can strengthen initial authentication for visual keys by extending their protection to scenarios involving pharming.

Mobile phones are also sometimes pressed into service for server-to-user authentication; financial institutions send SMS (Short Message Service) messages to confirm the completion of transactions. Of course, mobile phones can also authenticate users directly to servers via caller ID.

2.3 Stronger user authentication

A still different tactic for protecting user identities is to furnish users with stronger forms of authentication, i.e., to render identities more difficult to capture when users are exposed to malicious content. Passwords constituting by far the most prevalent form of on-line authenticator, Ross et al. [18] have created a browser plug-in called PwdHash. This simple but elegant application hashes the password entered by a user with the domain name of the site to which the password is being transmitted. PwdHash thus prevents situations like a phisher harvesting a password at 1roncladbank.com, from reusing it at the legitimate site Ironcladbank.com. PwdHash has the benefit of requiring no server-side modifications. It does have some drawbacks, however. First, it requires users to install plug-ins; the system becomes awkward when users employ browsers without PwdHash installed, so portability is limited. Additionally, PwdHash does not defend against pharming and other attacks [17], and provides improved but still limited resistance to guessing attacks against low-entropy passwords. Several researchers have also proposed complementary approaches to thwart password guessing, primarily involving expensive hash functions that raise the cost of brute-force attack [8, 16].

Challenge questions, as mentioned above, are a popular form of user authentication. They are similar to passwords, except that they rely on biographical secrets. On registering with a given site, a user is asked to furnish answers to questions such as "What high school did you attend?" or "What is the name of your favorite pet?" These questions may be posed to the user as a backup authentication method, i.e., to help in password recovery, or as a supplementary authentication method in cases where hardware authentication or client-machine authentication is insufficient. Challenge questions carry not only the drawbacks associated with passwords, i.e., guessability and vulnerability to disclosure, but are in some cases also subject to lookup in public databases. The challenge question "What is your mother's maiden name" is vulnerable in this sense, as recently demonstrated by Griffith and Jakobsson [7].

Given the vulnerability of passwords, PINs, and challenge questions to mining and guessing by attackers and to injudicious disclosure by users, a range of alternative and complementary user-authentication mechanisms has emerged on the Internet.

One of the most common alternatives is cookies. These are data objects cached in browsers and designed to provide persistent state in client-server interactions. Cookies are only accessible to the site that has stored them (first-party cookies) or to functionally affiliated sites (third-party cookies). "Secure" cookies are only released via HTTPS. Cookies require no user intervention or client-side modification, and are very attractive as a basic form of authentication. Cookies, however, have a couple of limitations. First, they are vulnerable to pharming attacks. That is, an attacker that can cause a browser to believe that it is visiting the site ironcladbank.com when it has in fact been directed to a server belonging to the attacker, can harvest the cookies set for ironcladbank.com. Second, many users today block cookies as a privacy measure (although more often third-party cookies than first-party ones).[2]

[2] In principle, browsers could be designed so that secure cookies are only released when the certificate name matches the domain name. This practice would make cookies much more potent tools against pharming. In order to compromise

As a result, some Web sites appeal to other forms of cacheable browser state to authenticate users. Shared Flash objects, for instance, can function as authenticators in much the same way as cookies, as can HTTP cache-control headers [15]. Cached data objects that can serve to identify (or track) users are sometimes refered to as *cache cookies*. Similarly, SSL session keys can help authenticate users. They reside on browsers as a form of persistent state, and thus permit resumption of a secure session without a fresh public-key operation. Jackson et al. [10] provide an overview of cache cookies and related authentication techniques (but regard them in the light of tracking methods that can potentially threaten user privacy).

As in our work, Juels, Jakobsson, and Jagatic [12] describe how cryptographic enhancements to cookies and cache-cookies can enhance resistance to man-in-the-middle attacks. They also demonstrate how browser history files, i.e., browser records of recently visited sites, Temporary Internet Files, and JavaScript objects can serve as a form of shared state for user authentication. Their ideas do not by themselves protect against general man-in-the-middle attacks as effectively as ours; their scheme relies on secret file-names that an adversary can progressively extract from an authenticating server.

IP tracing provides another alternative to passwords. The IP address of a given computer typically changes from session to session, thanks primarily to physical displacement of laptops and to the prevalence of Network Address Translation (NAT), a system for sharing IP addresses among multiple devices. Nonetheless, IP addresses can provide some information about devices, such as their associated subnet, ownership, and geographical vicinity. Some sites thus employ IP address information as a contributing factor in authentication decisions.

Hardware authentication tokens, e.g., RSA SecurID [9], once the preserve of corporate users, are increasingly entering the hands of consumers. These tokens generate one-time passcodes or execute challenge-response protocols. Hence, they do not suffer from the main security limitations of passwords. Hardware tokens, however, do not defend against simple man-in-the-middle attacks. A man in the middle can capture and forward a one-time passcode, and can act as a relay for challenge-response messages.

Active cookies represent yet another strengthened form of user authentication, but one, as we explain, that can help defeat man-in-the-middle and related attacks. Like all of the other approached to protecting user identities that we have enumerated, active cookies have their limitations. They are just one of the panoply of overlapping and complementary techniques required for successful defense against identity theft.

3 Our Active-Cookie Protocol

3.1 Technical Overview

The key concept behind our active-cookie protocol is the careful combination of an IP-tagged cookie with IP tracing. We let $AC_{\mathcal{P}}$ denote an IP-tagged cookie for our scheme (which, again, we refer to as an active cookie). A server $\mathcal{S}$ plants a user-specific, active cookie $AC_{\mathcal{P}}$ in the browser of a user $\mathcal{P}$. $AC_{\mathcal{P}}$ includes a secret key $x_{\mathcal{P}}$ specific to user $\mathcal{P}$. (As with conventional cookies, different servers can act independently: Each can place its own active cookie.)

We emphasize that users do not need to install any program or take any explicit action in order to reap the security benefits of active cookies. This is important for many reasons, including ease of

a secure cookie, then, a pharmer would have to compromise the PKI infrastructure by duping a user into installing a bad certificate in her browser, compromising a valid certificate, or obtain a certificate fraudulently.) Presumably browsers do not include this restriction on cookies because of common irregularities in certificate issuance in the industry.

deployment, security assurance, and platform independence. Active cookies only require that users accept first-party cookies. Many users reject third-party cookies, but rejection of first-party cookies is somewhat rarer. Moreover, as we shall explain, in a variant of our scheme, $AC_{\mathcal{P}}$ can be a Temporary Internet File or other cached object that carries a secret key. (It is even possible for $AC_{\mathcal{P}}$ to comprise multiple cached objects.)

An essential security feature of the active cookie $AC_{\mathcal{P}}$ is the way it is planted in the browser. In our scheme *$AC_{\mathcal{P}}$ is tagged with an IP address $IP_{\mathcal{S}}$, and not a domain name.* Thus only a server associated with $IP_{\mathcal{S}}$ can gain access to $AC_{\mathcal{P}}$. An attacker that spoofs a domain name, e.g., a pharmer that poisons a DNS table, does not gain the privileges required to access $AC_{\mathcal{P}}$ in the user's browser. As we explain below, $AC_{\mathcal{P}}$ is only accessible through what we call a *hard* channel.

While $AC_{\mathcal{P}}$ authenticates the browser of user $\mathcal{P}$, the full objective of $\mathcal{S}$ is larger: To ensure a trustworthy session with $\mathcal{P}$, i.e., that there is no attacker in the loop. Toward this end, the full active cookie protocol operates receives and transmits information over *two different channels*:

- **The soft channel:** The *soft* channel, which we denote by $Soft$, is the ordinary browser-to-server connection. This channel is soft in the sense that it is subject to diversion or hijacking by an adversary: In a phishing attack, the user is mislead via a soft channel to an incorrect URL; in a pharming attack, the user's browser contacts a correct domain name, but is diverted to a malicious server. (Note that although "soft" in one sense, the soft channel may be SSL-protected.)
- **The hard channel:** The *hard* channel, which we denote by $Hard$, is a browser connection to a fixed IP address $IP_{\mathcal{S}}$ belonging to the server $\mathcal{S}$. Because the hard channel corresponds to a server IP address, and not a server domain name, it is more trustworthy than a soft channel. An adversary can always direct a client to connect to $IP_{\mathcal{S}}$, i.e., onto the hard channel. But barring IP-spoofing, cookies and other content tagged with $IP_{\mathcal{S}}$ will only be released to the valid server $\mathcal{S}$.

When a client asserting user identity $\mathcal{P}$ contacts the server $\mathcal{S}$, we regard the channel thus established as the soft channel $Soft$. The client establishes this channel as a normal browsing operation, i.e., there is no active cookie here. To engage the active-cookie protocol, the server $\mathcal{S}$ first determines the IP address $IP_{\mathcal{P}}^{soft}$ associated with the soft channel. Then $\mathcal{S}$ redirects the browser of user $\mathcal{P}$ onto a hard channel, i.e., to $IP_{\mathcal{S}}$. (To achieve session consistency, the server assigns a unique session identifier SID to the soft-channel session, and tags the browser redirection with SID.)

When $\mathcal{S}$ receives the browser connection on the hard channel, it performs two checks:

1. $\mathcal{S}$ checks that there is no attacker in the loop. To do so, the server $\mathcal{S}$ determines the IP address $IP_{\mathcal{P}}^{hard}$ from which the client is transmitting on the hard channel $Hard$. $\mathcal{S}$ then verifies that $IP_{\mathcal{P}}^{soft} = IP_{\mathcal{P}}^{hard}$. The consistency between IP addresses on the soft and hard channels helps ensure that there is no attacker in the loop.
2. $\mathcal{S}$ authenticates the active cookie $AC_{\mathcal{P}}$. That is, it accepts $AC_{\mathcal{P}}$ from the client browser and checks that it contains the correct key $x_{\mathcal{P}}$ for user $\mathcal{P}$.

Our use of IP tracing, it should be emphasized, is one of the novelties of our solution. Existing systems generally rely on the operational characteristics of IP addresses, e.g., their registered network locations, as a means of tracking the patterns of use of a client computer over time. In contrast, active cookies are *agnostic* to the individual characteristics of a client's IP address and to changes in this IP address between sessions. Rather, as we have explained, active cookies seek to ensure a special form of IP-address consistency at session initiation. They verify equality between the IP address employed by a client on the ordinary, soft channel (which is affected by pharming) and its IP address as it appears on the hard channel. (In their use of out-of-band messaging—the band in this case being the ordinary browser session—active cookies are conceptually akin to authentication via SMS messaging.)

Of course, like an ordinary cookie, an active cookie is only of value as an authenticator when securely planted on a browser, i.e., securely initialized. Other forms of authentication are needed to establish a server-to-client relationship at the time an active cookie is set.

3.2 Adversarial powers

Our active-cookie protocol is designed to defend against an adversary that can cause a client to connect to the wrong server on the soft channel. The adversary can direct a client to an undesired URL via a phishing attack, e.g., to 1roncladBank.com rather than Ironcladblank.com. The adversary can alternatively mount a pharming attack, that is, cause a client attempting to contact a desired, correct URL to be routed to an incorrect IP address. In other words, we assume that the adversary can cause the client to connect on the soft channel to a hostile server. And the adversary can always, of course, cause the client to connect on the hard channel to the valid server $\mathcal{S}$.

On the other hand, we assume an adversary that cannot spoof IP addresses. First, we assume that the adversary does not actively corrupt routing systems. In other words, we assume that messages are routed correctly to their destination IP addresses – *this is distinct from causing an incorrect resolution of domains to IP addresses, as is done in a pharming attack.* Consequently, $\mathcal{S}$ can correctly determine the IP address on the soft channel. (While the originating IP addresses on packets may be corrupted by the adversary, we assume that the soft channel is a two-way connection such that IP addresses on both sides may be correctly determined.) Second, we assume an adversary that does not share an IP address with the valid client $\mathcal{P}$; for example, we assume that the adversary does not reside with the client behind a firewall with a single outward-facing IP address.

Finally, we make three assumptions fundamental to correct functioning of active cookies:

1. *Initialization:* The active cookie $AC_{\mathcal{P}}$ is securely initialized, i.e., there is no adversarial compromise of the initialization process.
2. *Access control:* Recall that $AC_{\mathcal{P}}$ is tagged with the originating IP address $IP_{\mathcal{S}}$, and therefore only accessible by a server that presents this IP address to the client. As a consequence of our assumption about correct routing, therefore, the adversary cannot access $AC_{\mathcal{P}}$.
3. *Corruption:* The adversary cannot modify the software executed by $\mathcal{P}$ or $\mathcal{S}$, or of any router on the normal path between these nodes. However, the adversary *can* modify the contents of the routing tables of such nodes, as well as of other nodes.

We detail the powers of the adversary further in our security analysis in section 3.5 below.

3.3 Active-cookie protocol: Details

We sketch our active-cookie protocol in Figure 1. Here, again, $\mathcal{S}$ denotes the authenticating server. SID is a unique session identifier. We let a single arrow, e.g., $\longrightarrow$, denote transmission over the soft channel, and a double arrow, e.g., $\Longrightarrow$ denote transmission over the hard channel, i.e., to IP address $IP_{\mathcal{S}}$. We let $\overset{k}{\to} D$ denote storage of a record in database D under database key k, and similarly let $\overset{k}{\leftarrow} D$ denote the associated process of retrieval. We assume a database X of secret keys referenced by the identities of users, i.e., by the value $\mathcal{P}$. (Alternatively, keys can be derived from a master secret.) Any failed database lookup operation or redundant message flow from the client results automatically in rejection of the authentication attempt.

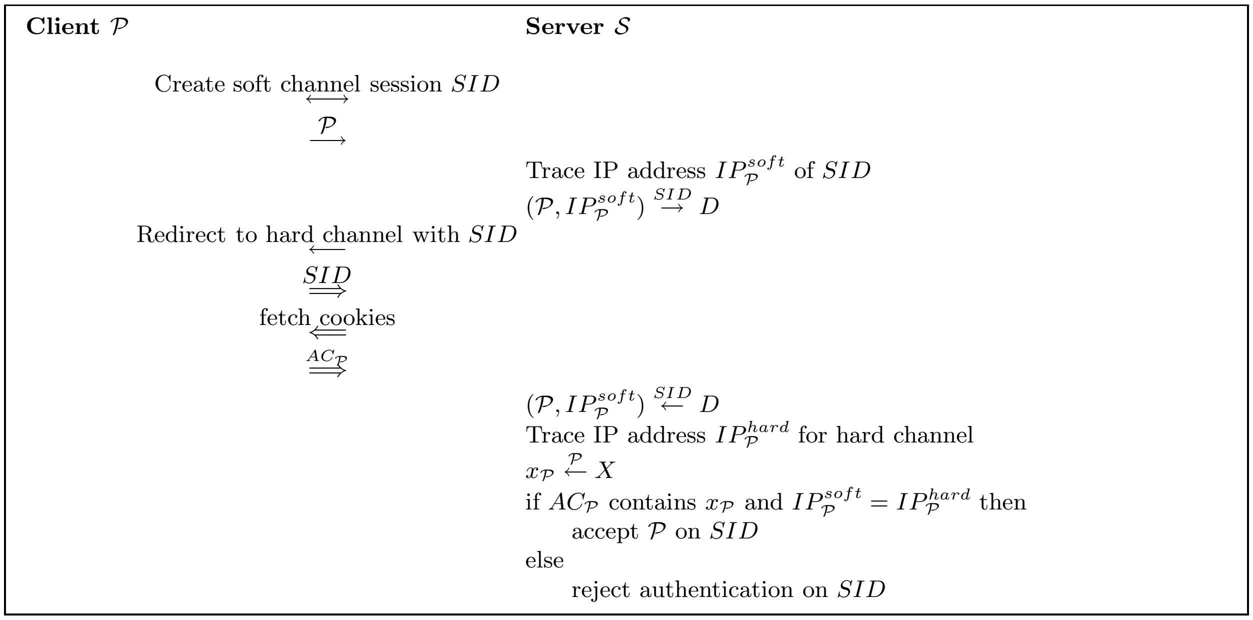
Client $\mathcal{P}$
Server $\mathcal{S}$
Create soft channel session SID $\longleftrightarrow$
$\xrightarrow{\mathcal{P}}$
Trace IP address $IP_{\mathcal{P}}^{soft}$ of SID
$(\mathcal{P}, IP_{\mathcal{P}}^{soft}) \overset{SID}{\rightarrow} D$
Redirect to hard channel with SID $\longleftarrow$
$\overset{SID}{\Longrightarrow}$
fetch cookies $\Longleftarrow$
$\overset{AC_{\mathcal{P}}}{\Longrightarrow}$
$(\mathcal{P}, IP_{\mathcal{P}}^{soft}) \overset{SID}{\leftarrow} D$
Trace IP address $IP_{\mathcal{P}}^{hard}$ for hard channel
$x_{\mathcal{P}} \overset{\mathcal{P}}{\leftarrow} X$
if $AC_{\mathcal{P}}$ contains $x_{\mathcal{P}}$ and $IP_{\mathcal{P}}^{soft} = IP_{\mathcal{P}}^{hard}$ then
accept $\mathcal{P}$ on SID
else
reject authentication on SID

Fig. 1. Active-Cookie Protocol

Remark: It is important to ensure the robustness of our protocol against denial-of-service attacks. Thus, entries in the database D should expire after an appropriate period of time T'—a prudent security measure in any case to ensure again exploitation of stale authentication sessions. As an alternative approach, $\mathcal{S}$ can dispense with D entirely, and instead apply a timestamp and message authentication code (MAC) to the tuples it sends to the client. The client can return these MACed values in its own message flows. $\mathcal{S}$ rejects any such tuple if it bears an incorrect MAC or if its timestamp antedates the current time by more than T'. (Roughly this approach is proposed in [11].)

3.4 Using SSL

SSL provides data integrity on traffic between endpoints (and, in principle, authentication of the server through its certificate). Active cookies, in contrast, aim to provide secure identification of endpoints, and particularly the client. While SSL and active cookies serve two different security goals, they are complementary.

Both the soft channel and hard channel may be instantiated as SSL connections. If the hard channel is SSL protected, then $AC_{\mathcal{P}}$ must be a *secure cookie*, that is, a standard type of cookie designated for release only in an HTTPS request. Browsers do not release ordinary HTTP cookies over SSL. Temporary Internet Files may be cached and retrieved over SSL, but do not persist between sessions.

Even in the presence of SSL, an active adversary on the hard channel can compromise our active cookie protocol by rerouting traffic in a given session, that is, by changing $IP_{\mathcal{P}}^{hard}$ in the view of $\mathcal{S}$. Nonetheless, use of SSL on the hard channel can provide stronger security in two senses: (1) It protects against capture of $AC_{\mathcal{P}}$, and thus later, offline impersonation of a valid client by an active adversary, and (2) It provides full protection against a passive, i.e., strictly eavesdropping, adversary on the hard channel.

Our expectation is that many deployers will choose to sacrifice the added security of SSL on the hard channel. Hard-channel SSL deployment requires management of a certificate bound to the IP address on the hard channel, in addition to any certificates associated with the soft channel. Indeed, load-balancing requirements might lead to use of multiple IP addresses on the hard channel, such that SSL deployment on the hard channel would impose a requirement for multiple extra certificates.

3.5 Security analysis

Initialization: As a precondition for secure deployment, of course, we require secure initialization of $AC_{\mathcal{P}}$ on the hard channel. An active cookie $AC_{\mathcal{P}}$ can only provide strong authentication if it has been securely planeted by $\mathcal{S}$ in the browser of the intended user $\mathcal{P}$. This means two things:

- $AC_{\mathcal{P}}$ has been planted in the browser of $\mathcal{P}$: In other words $\mathcal{S}$ has established a session with the correct client. At the time of initialization of an active cookie, that is, when $\mathcal{S}$ cannot leverage an existing active cookie, it can use alternative forms of authentication to achieve an adequate level of assurance. For example, $\mathcal{S}$ might use SMS messaging or challenge questions to authenticate the user. These tools are inconvenient for the user, and therefore not desirable for frequent use, but are suitable for what we hope to be the relatively rare initialization of active cookies. Once installed, an active cookie is transparent to the user.
- $\mathcal{A}$ has not eavesdropped on $AC_{\mathcal{P}}$ during initialization: With knowledge of $AC_{\mathcal{P}}$, of course $\mathcal{A}$ can impersonate $\mathcal{P}$. If $AC_{\mathcal{P}}$ is a secure cookie, then it must be planted over SSL, and is therefore not subject to eavesdropping. Without SSL, an eavesdropping adversary can potentially learn $AC_{\mathcal{P}}$. As we explain below, however, an adversary $\mathcal{A}$ that can compromise the hard channel can defeat the active-cookie protocol even after secure initialization.

We assume as a precondition for our analysis that the active cookie $AC_{\mathcal{P}}$ has been securely initialized in the browser of $\mathcal{P}$.

Details about the adversarial model: The goal of an adversary in defeating the active-cookie system is to impersonate the browser of $\mathcal{P}$. Suppose that $\mathcal{S}$ is associated with domain name XYZ.com. The most powerful type of adversary our techniques can defend against is one that mounts a pharming attack, that is, a man-in-the-middle attack in which $\mathcal{A}$ reroutes traffic on the soft channel to a malicious node. In particular, $\mathcal{A}$ causes the user to believe that she is accessing site XYZ.com; even the browser believes it is contacting the IP address for XYZ.com, when in fact the browser has connected with a malicious site.[3] Such an $\mathcal{A}$ may interact with the client $\mathcal{P}$ while trying to impersonate the client to $\mathcal{S}$.

As our system relies on IP tracing, it does not defend against a pharming adversary such that:

(1) $\mathcal{A}$ shares an IP address with the client $\mathcal{P}$.

For example, our active-cookie system does not protect against an adversary that resides behind a firewall with $\mathcal{P}$ on a network that performs Network Address Translation (NAT) and maps all clients to the same outward-facing IP address.[4] This is a basic limitation in our security model.

Even if $\mathcal{A}$ does not share an IP address with $\mathcal{P}$, if $\mathcal{A}$ can corrupt determination by $\mathcal{S}$ of the IP address $IP_{\mathcal{P}}^{soft}$ for the soft channel established by a client, then $\mathcal{A}$ can defeat our active-cookie system. $\mathcal{A}$ can simply make it *appear* that $IP_{\mathcal{P}}^{soft} = IP_{\mathcal{P}}^{hard}$, even when this is not the case. In other words, $\mathcal{A}$ can defeat our system if:

[3] In an SSL session, $\mathcal{A}$ may not present a valid certificate, but if $\mathcal{A}$ simulates an SSL session, e.g., displays a lock icon somewhere in the browser, this may be sufficient to dupe the user. Even if $\mathcal{A}$ does launch an SSL session with the wrong certificate, users may disregard warnings generated by their browsers.

[4] Our protocol can be extended to trace ports as well as IP addresses, an extension that we do not treat here.

(2) $\mathcal{A}$ actively corrupts network routing systems, causing $\mathcal{S}$ to determine an incorrect IP address $IP_{\mathcal{P}}^{soft}$ on the soft channel.

We also note, of course, that $\mathcal{A}$ can defeat the active-cookie system—and almost any other authentication system—if:

(3) $\mathcal{A}$ corrupts $\mathcal{P}$ or $\mathcal{S}$, i.e., $\mathcal{A}$ gains control of the client or server.

Let us assume in our remaining analysis, then, that $\mathcal{S}$ can correctly determine a unique IP address $IP_{\mathcal{P}}^{soft}$ for a client's soft channel. In other words, let us exclude cases (1), (2), and (3).

Attack analysis: Suppose that $\mathcal{A}$ mounts a man-in-the-middle pharming attack against $\mathcal{P}$. Then $\mathcal{A}$ obtains a session identifier SID in a session that it initiates with $\mathcal{S}$. In order to mount a successful impersonation attack, then, $\mathcal{A}$ must accomplish two things: (A) Cause a session with identifier SID to be invoked over a hard channel with $\mathcal{S}$, along with release of an active cookie $AC_{\mathcal{P}}$ with valid key $x_{\mathcal{P}}$; and (B) Cause $\mathcal{S}$ to determine that $IP_{\mathcal{P}}^{soft} = IP_{\mathcal{P}}^{hard}$ for session SID. And $\mathcal{P}$ will only release $AC_{\mathcal{P}}$ over the hard channel to a server with IP address $IP_{\mathcal{S}}$.

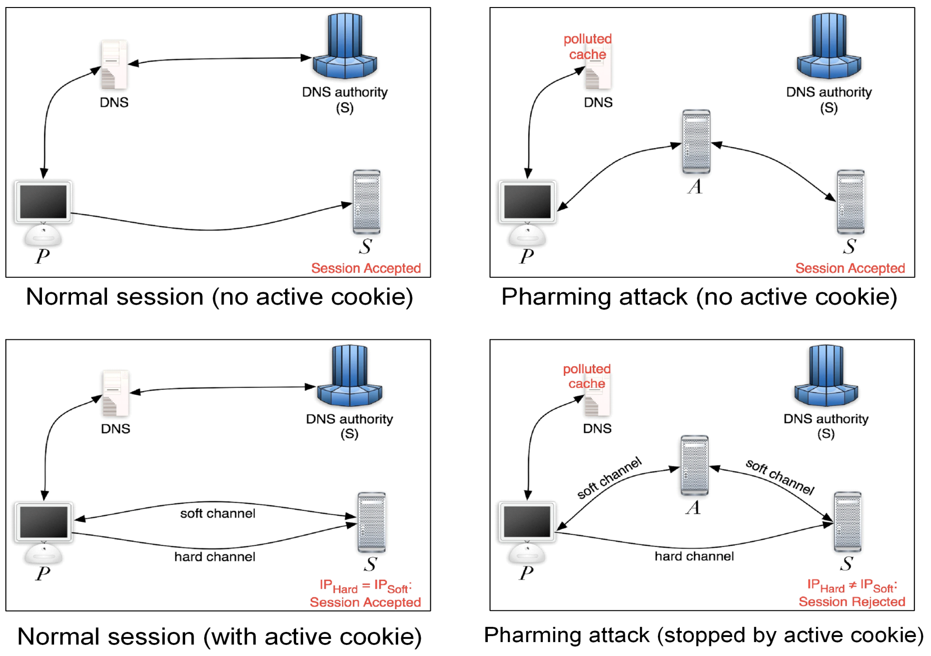
DNS
DNS authority (S)
P
S
Session Accepted
Normal session (no active cookie)
polluted cache
DNS
DNS authority (S)
A
P
S
Session Accepted
Pharming attack (no active cookie)
DNS
DNS authority (S)
soft channel
hard channel
P
S
IP_Hard = IP_Soft: Session Accepted
Normal session (with active cookie)
polluted cache
DNS
DNS authority (S)
soft channel
soft channel
A
hard channel
P
S
IP_Hard ≠ IP_Soft: Session Rejected
Pharming attack (stopped by active cookie)

Fig. 2. Scenarios illustrating effect of active cookies against a pharming attack

Therefore, the only way that $\mathcal{A}$ can mount a successful impersonation attack is through successful compromise of the hard channel $Hard$ between $\mathcal{P}$ and address $IP_{\mathcal{S}}$. If $\mathcal{A}$ can eavesdrop on the hard channel or spoof $IP_{\mathcal{S}}$, then $\mathcal{A}$ can learn the active cookie $AC_{\mathcal{P}}$ and use it in a later session to impersonate the user. If $\mathcal{A}$ can actively modify traffic on the hard channel, then $\mathcal{A}$ can additionally hijack the current session by redirecting traffic and spoofing $IP_{\mathcal{P}}^{hard}$ so that it matches some $IP_{\mathcal{P}}^{soft}$ corresponding to a client controlled by the attacker.

Therefore, $\mathcal{A}$ can only impersonate $\mathcal{P}$ successfully if:

(4) $\mathcal{A}$ compromises the hard channel.

In summary, then, given secure initialization of $AC_{\mathcal{P}}$, an attacker $\mathcal{A}$ can only successfully impersonate $\mathcal{P}$ successfully by achieving one of the following conditions: (1) $\mathcal{A}$ shares an IP address with the client $\mathcal{P}$; (2) $\mathcal{A}$ actively corrupts network routing systems, causing $\mathcal{S}$ to determine an incorrect IP address $IP_{\mathcal{P}}^{soft}$ on the soft channel; (3) $\mathcal{A}$ corrupts $\mathcal{P}$ or $\mathcal{S}$, e.g., $\mathcal{A}$ plants malware on the client or server; or (4) $\mathcal{A}$ compromises the hard channel.

It is important to note that conditions (1) and (2) do not lead to compromise of $AC_{\mathcal{P}}$ itself. The same is true for condition (4) when $AC_{\mathcal{P}}$ is a secure cookie and therefore only subject to release over SSL. In all of these cases, the adversary can only compromise the active-cookie system on a temporary basis. In contrast, condition (3) effectively means total compromise of the system (with problems beyond the scope of our work). If $AC_{\mathcal{P}}$ is not a secure cookie, then condition (4) allows a pharming adversary to compromise $AC_{\mathcal{P}}$ directly and thus impersonate $\mathcal{P}$ in later sessions.

Figure 2 above depicts four session types, namely a normal session and a session under attack from a pharmer, each with and without active cookies.

We shall present a more formal security model and analysis in the full version of the paper.

4 Implementation

We developed our prototype implementation of the active-cookie protocol on industry-standard platforms. We wrote PHP version 5 scripts to run on Apache Web server version 2.2.0. The Apache server and PHP module were compiled and installed in Gentoo Linux R9 with kernel 2.6.15 using the GNU compiler suite version 3.3.6.

We performed testing of our prototype mainly in Firefox versions 1.0.4 and 1.5 on Windows XP and Mac OS 10.4.4 respectively. We chose the Firefox browser because of its strict adherence to the W3C HTML specifications, as well as its fairly widespread use. Tests also indicate that our prototype functions in Microsoft Internet Explorer 6.0.

Our software consists of standard HTML and some basic PHP scripting on the server to provide rapid prototype development and access to HTTP headers.

Setting an Active Cookie: In order to place an active cookie in the browser of a visitor, our prototype guides clients through a very quick setup phase. This is required *only* when a visitor does not yet possess an active cookie for our site in her browser. The visitor begins by identifying herself to the server (by providing $\mathcal{P}$ as described in Figure 1). The server generates a session id SID and new 128-bit secret key, then stores the pair $(\mathcal{P}, x_{\mathcal{P}})$ in a list of active keys. The server records SID as a pair with $\mathcal{P}$ to inform the hard channel that $\mathcal{P}$ is undergoing setup, and then forwards the client to the hard channel. Finally, over the hard channel, the server sets the cookie value $x_{\mathcal{P}}$ on the client computer. This cookie value is accessible only to the server when Web pages are accessed on the hard channel.

When the cookie is initially served to a client, it is delivered with an expiration time set far in the future to help ensure its retention for a long time (arbitrarily one year, in our implementation).

Verification: Once the cookie is set in a client's browser, the server that set the cookie can verify the validity of the soft channel between them. This occurs automatically in our prototype when a client who already has an active cookie accesses the login area of the site.

On receiving the user's a login request, the server generates a fresh SID. The server records this SID along with the client's IP as registered on the soft channel. The server then redirects the client to the hard channel. To pass the SID onto the hard channel, the redirect assumes the form of a HTTP "Location:" header of the form: "http://[IP address]/hc_verify.php?SID=xxxxxxx."

Verification occurs on the hard channel. The server records the SID and IP address presented by the client on the hard channel and receives the active cookie $AC_{\mathcal{P}}$.

On receiving the cookie, the server verifies for session SID that: (1) The secret value in $AC_{\mathcal{P}}$ matches the secret key $x_{\mathcal{P}}$ that the server has on record and (2) The recorded IP addresses presented by client on the hard channel and soft channel are identical. If both conditions are met, the server returns a page on the soft channel indicating that the channel is secure, and that the client may proceed with login.

When the server determines the soft channel to be insecure—i.e., when the key is wrong or the soft and hard client IPs differ — the user is forwarded to a warning page via the hard channel (instead of back to the soft channel). This way they are presented with a warning that cannot be intercepted and removed by an attacker who is on the soft channel. Additionally, the server can record the channel as invalid (based on SID) and disallow any critical transactions.

Performance: As noted above, our system requires no client-side changes. No client plug-ins are required; our system requires special-purpose software only on the server side.

We tested our active cookie prototype to determine the overhead that it imposes on both a client and the server. We observed no significant slowdown. Execution of the active-cookie protocol for authentication is very fast: it takes less than a second. Of course, this time will vary depending on the normal Internet connection speed of the client and server, the computing speed of the client, and network congestion. We believe that since the protocol involves two rounds of communication between client and server, the variance in execution time will be minimal, and never prohibitive.[5]

For a client with an established active cookie (the secret cookie value is set), the experience in visiting our prototype server is little different from that of visiting a site without this authentication. The only noticeable feature of our system is that the browser is rapidly redirected twice—to and from the hard channel—before the login prompt is displayed. This procedure is no different than the multiple redirections that take place after a user enters her credentials into an ordinary (if sophisticated) Web application like Google's Gmail service. Redirection occurs in less than a second, and then the login prompt appears.

The impact of active cookies on a server is minimal. Our implementation uses a small amount of disk space for the secret keys distributed to clients. The computational load is considerably less than that imposed by SSL. (We are in the process of obtaining precise measurements.) We believe, therefore, that active cookies will impose only marginal overhead on secure server systems.

5 Extensions and Variants

We envisage a few extensions that could make active cookies more attractive in deployment:

Load-balancing: In many Web environments, a domain comprises multiple servers in the interest of load balancing. It may be desirable to direct a client to any of a number of servers $\mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_n$ on the hard channel during the process of authentication, rather than a single, monolithic server $\mathcal{S}$. Thus, initialization of a client $\mathcal{P}$ may involve caching a separate active cookie $AC_{\mathcal{P}}^{(i)}$ for each of the n servers—with either a shared or distinct secret $x_{\mathcal{P}}^{(i)}$.

Temporary Internet Files: Rather than caching $AC_{\mathcal{P}}$ on the client as an ordinary cookie, it is possible to store it as a Temporary Internet File (TIF)—in particular, an HTML file.

[5] We are in the process of measuring client-side and server-side overhead times, and plan to include a plot in the final version of this paper.

We can let $AC_{\mathcal{P}}$ be a Web page specific to user $\mathcal{P}$ that contains a reference to an object with secret URL $x_{\mathcal{P}}$; this URL is associated with the IP address $IP_{\mathcal{S}}$. (For example, suppose that the hard-channel IP address is 255.255.255.255. The resource referenced in $AC_{\mathcal{P}}$ might be a GIF with the URL http://255.255.255.255/$x_{\mathcal{P}}.gif$.) The server $\mathcal{S}$ sets $AC_{\mathcal{P}}$ on the client over the hard channel, but does not cache the object with name $x_{\mathcal{P}}$. When the Web page is invoked, the client requests $x_{\mathcal{P}}$. As a result, the client transmits the secret value $x_{\mathcal{P}}$ to $\mathcal{S}$.

Of course, an adversary can invoke $AC_{\mathcal{P}}$ by referring to it in its own content served to the client $\mathcal{P}$. Contemporary browsers enforce same-origin policies around TIFs, however. Consequently, barring browser-security flaws, adversarial code, such as JavaScript, should be unable to reference the internal contents of $AC_{\mathcal{P}}$ and extract $x_{\mathcal{P}}$.

An attractive feature of active cookies deployed as TIFs is that they work even in the face of complete cookie blocking or purging by the client. Furthermore, they do not require the use of executables like JavaScript or Java (which are unavailable on some 10% of clients [5]). On the other hand, TIFs are themselves subject to manual purging by users looking to improve browser performance. Additionally, browsers automatically purge selected TIFs when their TIF caches are full. A server can enhance the persistence of a TIF in a given browser by accessing it repeatedly, as browser policies favor purging of less frequently used objects.

Another form of TIF we can employ is a cache cookie—in this case a sequence of active cookies with secret names as proposed in [12]. These provide a narrow, additional form of security. They protect not just against domain-spoofing but also against IP- spoofing attacks—but only for offline attackers, i.e., those that are not harvesting the secret names via man-in-the-middle attacks.

Executable active cookies: It can be desirable for an active cookie to involve an executable function, that is, for $\mathcal{P}$ not to release $AC_{\mathcal{P}}$ directly, but to apply a cryptographic function f (or even a multi-round protocol) to $x_{\mathcal{P}}$.

For example, an active cookie might generate a one-time passcode compatible with the RSA SecurID system [9], with $x_{\mathcal{P}}$ serving as the token seed. (We could even dispense with the IP-tracing elements of our protocol, simplifying deployment at the expense of protection against pharming attacks.) Or an active cookie might perform a public-key-based challenge-response operation with $x_{\mathcal{P}}$ as the private key, thus simulating the operation of a smartcard.

Incorporating executable code into an active cookie is not difficult. The server $\mathcal{S}$ serves the code for f over the hard channel. This code can take essentially any desired form, JavaScript being perhaps the most practical. The executable can act upon a a secret value $x_{\mathcal{P}}$ residing in any desired type of separate, static object, whether it be a TIF or a cookie. If the executable f refers to the object via a URL associated with the hard channel (and thus a URL that is IP, rather than domain-based), and the object is cached, then the secret $x_{\mathcal{P}}$ will be locally accessible. It is important that the executable itself contain no secrets, as sandboxed programs do not enjoy the same access restrictions as static cached objects. For example, JavaScript functions are subject to enumeration and referencing in a global name space. Thus, an adversary can access even a piece of JavaScript cached over the hard channel.

$\mathcal{S}$ may deliver fresh executable code for f in every authentication session. Alternatively, if desired, the executable code for f can be set to reside in the client cache along with $x_{\mathcal{P}}$ for later invocation by $\mathcal{S}$.[6]

Remarks:

[6] While as secure in principle as dynamic serving of code, caching of f could be more dangerous in practice should there exist holes in the same-origin policies of browsers.

– Social engineering is a danger with TIF-based secrets, although probably a minor one. An adversary can potentially dupe a user into revealing $AC_{\mathcal{P}}$. For example, if $AC_{\mathcal{P}}$ is a GIF, the adversary could reference it on the soft channel and cause it to be displayed to the user. By persuading the user to drag and drop the image into e-mail, the adversary could potentially capture it. Such tricks are more or less difficult to mount, depending on the nature of the TIF. (A large transparent GIF, for example, might help thwart drag-and-drop attacks.)
– In the face of cookie blocking and TIF purging, the most robust approach to active-cookie deployment is to construct $AC_{\mathcal{P}}$ as a composite of different objects. $AC_{\mathcal{P}}$ might, for instance, consist of an ordinary cookie containing $x_{\mathcal{P}}$ *in addition to* a Web page $W_{\mathcal{P}}$ referencing a URL containing $x_{\mathcal{P}}$, a cache cookie in the browser history, etc. If some of the objects containing $x_{\mathcal{P}}$ are purged, $\mathcal{S}$ can still retrieve the others. (To provide some insulation against compromise of a particular cached-object type, it may be desirable to associate distinct keys with different objects.)

5.1 Simplified active-cookie variants

The essence of our active-cookie protocol is redirection of browsers onto a trustworthy hard channel. Stripping our system down to the bare essentials yields two variants:

Simple swapping: A simplified variant of our protocol is for $\mathcal{S}$ to swap the hard channel in for the soft, i.e., to perform the redirect and drop the soft channel. The advantage of this approach is conceptual simplicity: It obviates the need for the equality check on IP addresses, and still allows for $AC_{\mathcal{P}}$ to be either an executable or a cookie. By maintaining an outer browser frame corresponding to the original domain name and maintaining an inner frame for the redirected connection, it is possible to make this variant invisible to the user. While perhaps attractive in some situations, this variant has a drawback. In order to execute SSL on a hard channel $IP_{\mathcal{S}}$, a server must have a certificate specific to $IP_{\mathcal{S}}$. Such a certificate would mean added overhead, and might look suspicious to users. Moreover, in a load-balancing environment in which there are multiple hard-channel IP addresses, it would be necessary to support multiple certificates.

Bookmark variant: The most problematic situation is where users clear both cookies and TIF caches. In a *bookmark* variant, the server asks each user at the end of the setup phase to add to her bookmarks a pointer to a secret URL—one that is hard-coded, random, and unique. In a situation where verification fails due to the absence of an active cookie, the user can be asked to follow the bookmark, allowing the server to both verify the identity of the client (by means of the unique bookmark) and of the absence of an attacker on the communication channel (given the use of the hardcoded channel). While this approach requires user involvement and is not as convenient or secure as our previously described protocol, it offers a worst-case fall-back mechanism.

6 Conclusion

We have proposed active cookies as a lightweight, highly flexible technique for authenticating browsers. Active cookies have certain limitations. They do not, for instance, offer security against strong attacks like active corruption of routers on the client-server path, as do more holistic cryptographic solutions. But active cookies can help protect against virulent attacks like pharming that defeat even hardware authentication tokens. Their outstanding feature is that they create no real change in user experience and are easy to administer on the server side.

Active cookies have a salient limitation: They authenticate the browser of a user, *not* the user herself. A user that changes browsers or purges browser caches will lose the benefit of previously planted active cookies. We view active cookies as playing a role like that of ordinary cookies in user authentication. To account for movement of users among different platforms, servers are increasingly adopting authentication methods that distinguish between home environments and those used by users while roaming. Active cookies are just one tool in a growing arsenal that needs both to accommodate the varying habits of users and to respond to the evolution of online attacks.

An important area for future research is the ways in which active cookies can harmonize with existing authentication tools. We believe that active cookies can work on mobile platforms, such as cell phones, and are investigating such deployments. We are also performing research on a range of subtle extensions and refinements to our security model and assumptions.

Trusted computing platforms and other potentially monolithic solutions to the problem of authentication are unlikely to emerge soon. As systems rely increasingly on a palette of tools for authentication, we believe that active cookies will have a valuable role to play.

Acknowledgments

The authors thank Ben Adida and David Leach for their comments and critiques of this work. We also thank Chris Karlof and Umesh Shankar for pointing out a security vulnerability in an earlier version of this work and suggesting the idea of dynamically-served executable active cookies.

References

1. Domainkeys: Proving and protecting email sender identity. Referenced 2006 at antispam.yahoo.com/domainkeys.
2. Trustbar. Referenced 2006 at trustbar.mozdev.org.
3. Sender ID technology: Information for IT professionals, Updated February 17, 2005. Referenced 2006 at www.microsoft.com/mscorp/safety/technologies/senderid/technology.mspx.
4. B. Adida, D. Chau, S. Hohenberger, and R. Rivest. Lightweight signatures for email, 2005. Presented at DIMACS Workshop on Theft in E-Commerce. Referenced 2006 at http://theory.lcs.mit.edu/∼rivest/publications.html.
5. The counter.com (R). Global statistics 2005, 2006. Internet Usage Statistics. Referenced 2006 at http://www.thecounter.com/stats/2005/December/javas.php.
6. Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. In *Intl. Workshop on Theory and Practice in Public Key Cryptography (PKC)*, page 130144. Springer-Verlag, 2003. LNCS no. 2567.
7. V. Griffith and M. Jakobsson. Messin' with Texas: Deriving mother's maiden names using public records. In J. Ioannidis, A. D. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security (ACNS)*, page 91103. Springer-Verlag, 2005. LNCS no. 3531.
8. J.A. Halderman, B. Waters, and E.W. Felten. A convenient method for securely managing passwords. In *14th Intl. World Wide Web Conference*, pages 471–479, 2005. Referenced 2006 at www.cs.princeton.edu/∼jhalderm/papers/www2005.pdf.
9. RSA Security Inc. SecurID product description, 2006. Referenced 2006 at http://rsasecurity.com/node.asp?id=1156.
10. C. Jackson, A. Bortz, D. Boneh, and J. Mitchell. Web privacy attacks on a unified same-origin browser. In *15th Intl. World Wide Web Conference*, 2006. To appear.
11. A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In S. Kent, editor, *Network and Distributed System Security Symposium*, pages 151–165, 1999.
12. A. Juels, M. Jakobsson, and T. Jagatic. Cache cookies for browser authentication (extended abstract), 2006. To appear.
13. D. V. Klein. Foiling the cracker: A survey of and improvements to, password security. In *UNIX Security II: USENIX Workshop Proceedings*, pages 5–14, Berkeley, CA, 1990.
14. Bank of America SiteKey description, 2006. Referenced 2006 at http://www.bankofamerica.com/privacy/passmark/.
15. M. Pool. Meantime: non-consensual HTTP user tracking using caches, 2000. Blog entry. Referenced 2006 at http://sourcefrog.net/projects/meantime.
16. N. Provos and D. Mazières. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, page 8191, 1999.
17. A. Raskin. Simulated browser attacks. In M. Jakobsson and S. Myers, editors, *Phishing and Anti-Phishing.* Wiley, 2006. To appear.

18. B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. Mitchell. Stronger password authentication using browser extensions. In P. McDaniel, editor, *USENIX Security*, pages 17–32, 2005.
19. J. Vijayan. Microsoft warms of fraudulent digital certificates, 22 March 2001. Referenced 2006 at http://www.computerworld.com/softwaretopics/software/story/0,10801,58857,00.html.
20. T. Wu. A real-world analysis of kerberos password security. In *Network and Distributed System Security Symposium*, 1999.

A Playing hide-and-go-seek with malware

Malware is a malicious executable that compromises a client machine. It may take the form of a virus, a worm, or a Trojan horse. In principle, malware can access the full state of the client and monitor input to the client from the user by, e.g., monitoring keystrokes. For this reason, there is perhaps no system for user authentication that is generally robust against malware without substantial reliance on some external device, e.g., a hardware authentication token.

In practice, however, the functionality of malware is often limited. We briefly consider several types of constraints and describe countermeasures that we might deploy in active cookies.

Time-limited malware: A piece of malware often infects clients over just a limited window of time. Once a virus becomes widespread, for example, virus checkers are typically updated to recognize and eliminate it. In consequence, we can harden an active cookie by means of periodic modification to client state. For example, suppose that the key $x_{\mathcal{P}}$ is periodically updated after successful client authentication. In particular, suppose that rather than a single static key $x_{\mathcal{P}}$, the active cookie for $\mathcal{P}$ holds a key $x_{\mathcal{P}}^{(i)}$ that valid only during authentication epoch i. In other words, after epoch i, the active cookie is refreshed with a new key $x_{\mathcal{P}}^{(i+1)}$. (An epoch might be defined according to calendar time, number of successful user authentications, etc.)

Suppose then that the attacker successfully compromises $AC_{\mathcal{P}}$ and the password of the user during epoch i. If the user successfully updates the active cookie and receives a new key $x_{\mathcal{P}}^{(i+1)}$ after the window of infection and prior to key update by the attacker, then the attacker, who possesses the now invalid key $x_{\mathcal{P}}^{(i)}$, loses the possibility of impersonating the user. Alternatively, the attacker might successfully update her own active cookie $AC_{\mathcal{P}}$, and obtain a new key $x_{\mathcal{P}}^{(i+1)}$. In this case, the active cookie of the valid user would contain a stale key $x_{\mathcal{P}}^{(i)}$. The attacker might be able to impersonate the user. But if the user attempts to authenticate, the server can detect her use of the stale key $x_{\mathcal{P}}^{(i)}$—a suspicious event that would alert the server administrator to a compromise of the active cookie for $\mathcal{P}$.

Prior to infection, it is also possible to cache secret data on the client. Juels, Jakobsson, and Jagatic [12], for example, show how cache cookies can be hidden in a browser cache so that they are only addressable by a server with prior knowledge of their location. We can use a similar principle to hide active-cookie state within cache-cookies. Of course, we cannot always prevent a piece of malware from addressing these secret cache-cookies, since it may have access to the full state of the client. But we can hide cache-cookies in such a way that malware cannot easily determine where they reside, and would therefore have to transmit a large fraction of the state of the client in order to exploit them outside the window of infection.

As a simple example, consider the following enhancement to our active cookie protocol. A server might cache a sequence of TIF-based active cookies $AC_{\mathcal{P}}^{(1)}, \ldots, AC_{\mathcal{P}}^{(m)}$. In epoch i, after successful authentication via our basic active-cookie protocol and after the user has entered the correct password, the server invokes the active-cookie protocol on $AC_{\mathcal{P}}^{(i)}$. In this case, a piece of malware can only locate the active cookie for a future epoch i by trawling the cache of the client. Provided that the TIFs for the active cookies and their associated URLs are created without obvious distinguishing characteristics, it may be possible to prevent their compromise by malware. Even an adversary that compromises a subset of active cookies and the password of the user cannot exploit them to achieve successful authentication in authentication epochs beyond the window of infection. This concept is similar in spirit to the notion of key-insulated cryptography as in, e.g., [6].

Operationally limited malware: A piece of malware may be designed to perform only a limited number of operations. Malware may be constrained by limitations in its infection vector or by a strategy to avoid detection by monitoring software. Alternatively, it may perform very general client monitoring, without mounting a focused attack against a particular application or Web site. For example, a piece of malware might act exclusively as a keystroke logger and cookie collector, i.e., might seek to seize static data used for authentication. In this case, the malware will fail to bypass the protection afforded by the active cookie.

B Broader Application of Our Techniques

Our active-cookie techniques can even be useful not just with browser caches, but also for authentication in plug-in or standalone applications. In principle, such applications can access SSL keys or generate their own secure channels, and can therefore rely on purely cryptographic protocols to defend against man-in-the-middle attacks, rather than techniques that hinge on identification of IP addresses. Deployment of our IP-address-based techniques as part of a standard executable (rather than a cached and/or sandboxed one) may still prove attractive in certain situations. Although not as strong as fully integrated cryptographic solutions, our techniques have the advantage of enforcing a clean separation between protocol layers. In other words, they can enhance the security of an SSL session while respecting encapsulation of the cryptographic module that executes SSL, a feature that may be helpful in, e.g., achieving FIPS 140 compliance.

Even in the case in which standalone software executes cryptographic protocols that are robust against man-in-the-middle attacks, our general schema of enabling sites to cache secrets or keys on browsers may prove valuable. A Web browser (or an extension like a toolbar) can be configured with a key-store whose entries remote sites may invoke for the purpose of user authentication. This serves essentially as a form of "light" client-side certificate, and can support either symmetric or asymmetric approaches, and either client-to-server or mutual authentication. In order to protect against denial-of-service attacks it is necessary to restrict the number of keys that a given server can cache. There are a few ways of accomplishing this. One is to assign a quota of keys to each domain that a browser connects with—or to every distinct, validated server-side certificate presented to a browser via SSL. Re-keying privileges must also be restricted, either in the same way as initialization and/or on the basis of server knowledge of a given key to be updated. Many possible policies around use of a given key are possible. It can be open for use by any server, to a set of delegated servers, or to the initializing server alone.