Journal of Computer Security 19 (2011) 869–893
DOI 10.3233/JCS-2011-0424
IOS Press

WebCallerID: Leveraging cellular networks for Web authentication

Francis Hsu [a], Hao Chen [a,*] and Sridhar Machiraju [b]
[a] *University of California, Davis, CA, USA*
[b] *Google Inc., Mountain View, CA, USA*

Web authentication that is both secure and usable remains a challenge. Passwords are vulnerable to phishing attacks, while physical tokens face deployment obstacles. We propose to leverage the authentication infrastructure of cellular networks to enhance Web authentication. We design WebCallerID, a Web authentication scheme that uses cell phones as physical tokens and uses cellular networks as trusted identity providers. Since WebCallerID requires no user participation during authentication, it prevents security mistakes by users. WebCallerID also prevents rogue websites from replaying authentication assertions or stealing users' identities. We have implemented a prototype of WebCallerID using the OpenID framework. The prototype shows that WebCallerID seamlessly integrates into OpenID-capable Web authentication while avoiding phishing problems in OpenID and simplifying user participation.

Keywords: Authentication, cellular networks, mobile authentication, OpenID, phishing, single sign on, usable security, Web authentication

1. Introduction

Identity is a necessary component of most security-sensitive computer interactions. We assert and verify identity through *authentication*, where one party determines the identity of the other by verifying some evidence supporting the identity assertion.

The simplest, oldest form of authentication is conducted with a shared secret, often a password. We continue to use password-based authentication even for modern systems like the Web, because it is familiar to most users, even though it suffers from well-known usability problems. Since passwords are vulnerable to brute-force attacks, authentication systems require users to create strong passwords that are difficult for automated systems to guess. These strong passwords, however, are long and difficult to memorize. When a user submits passwords to a remote server, the web browser simply submits the plain text password directly to the remote side (but at least encrypted in transport by TLS). Such an authentication process is vulnerable to reply attacks. If this password is captured at either the browser or the Web

*Corresponding author: Hao Chen, Department of Computer Science, University of California, Davis, CA 95616-8562, USA. Tel.: +1 530 754 5375; Fax: +1 530 752 4767; E-mail: hchen@cs.ucdavis.edu.

server, it may be reused. Password-based Web authentication systems are especially vulnerable to replay attacks because web users often reuse the same password across different websites. If an attacker can capture a user's password at one website, he may use the password to log into the user's accounts at many other websites. To steal a user's password, the attacker can either exploit vulnerabilities at poorly administrated websites or launch a phishing attack on the user. Although techniques are available to help users identify rogue websites, studies have found that ordinary users fail miserably at such tasks [10].

Saltzer and Schroeder's security design principle of Separation of Privilege [33] suggests that critical systems protected by multiple mechanisms are more robust. As a result, *multi-factor authentication* was proposed to make subverting an authentication process more difficult since the attacker must obtain or forge more than one piece of evidence. These factors are frequently categorized into "something you know", "something you have" and "something you are". Passwords are an example in the first category.

Physical tokens can provide an additional factor in the "something you have" category. They may function individually as the single factor in an authentication process, but are frequently paired with passwords so that loss or theft of the token is not sufficient to compromise the authentication process. These tokens carry a secret shared with the authenticating site, but only show the user a string to be sent to the remote website. This string is derived from the secret and other sources such as a clock [32]. Physical tokens, however, have not been widely used for Web authentication, mainly due to deployment barriers. First, this approach requires extra hardware and incurs extra cost. Second, this approach requires the creation of *token authorities*. If each website issues its own token, a user would have to carry a different token for each website. On the other hand, if websites share a single token authority, the authority has to register users and verify their identities, a substantial undertaking. Finally, integrating physical tokens seamlessly into Web authentication may be a challenge. If the token communicates directly with the browser during Web authentication, it may require additional device drivers and browser plugins. In practice, currently users have to relay challenges and responses between the browser and the token. This step is vulnerable to some of the same weaknesses of passwords. Again, a phishing attack could capture user's token response with the user's password and then use them in a man-in-the middle attack. In this case, phishing attacks can still be successful because they exploit the human element in the authentication process.

Phishing vulnerabilities suggest that removing humans from the authentication process could reduce vulnerabilities of authentication systems. In fact, there is a large infrastructure that authenticates billions of users every day without requiring their active participation – the cellular networks. Cellular networks authenticate their users through the hardware in their cell phones.[1] Thus, cell phones serve as physical tokens

[1]As cell phones are highly personalized devices, cellular networks attribute all the services used on a cell phone to its owner, just as banks attribute all the transactions initiated by an ATM card to the card's owner. Techniques exist to prevent the use of stolen cell phones, such as lock codes.

to their networks and avoid all the problems of user involvement in the authentication process.

In this paper, we propose using cell phones for Web authentication. This approach leverages the authentication infrastructure inside cellular networks to enhance Web authentication. Compared to other approaches based on traditional physical tokens, our approach overcomes many deployment barriers. First, cell phones can be used for Web authentication without incurring additional costs or burden to most people. This is because most people who use the Web also own cell phones and carry the phones with them wherever they go. Moreover, due to financial and privacy reasons, they tend to diligently safeguard their cell phones. Second, cellular networks have already verified the identities of most users (exceptions include some pre-paid users). Most countries have a few, well-established cellular networks. If a website trusts these networks, it can leverage user authentication provided by these providers. For example, SMS (text messaging) has been used for authorizing payment by financial institutions, such as PayPal [28]. Finally, as more cell phones support data services, we can leverage cellular data networks for Web authentication seamlessly, i.e., without the need for additional hardware (e.g., physical tokens) or software (e.g., client authentication programs or browser plugins), or for users to relay information to and from browsers (e.g., manually copying authentication codes from an SMS into a browser). Users can run commodity browsers either on their cell phones, or on PCs that have network connections to their cell phones (Section 4.4).

As physical authentication tokens, cell phones are not only easy to deploy but can also provide a unique piece of information: the locations of their users. In many jurisdictions, cellular networks are required to identify the location of phones for emergency response. Location information has been effectively used for detecting financial fraud, such as ATM and credit card transactions. As a result, a card-present transaction often incurs a lower cost than a card-absent transaction. Most Web transactions are treated as card-absent, because the location of the browser cannot be verified (e.g., due to the use of proxies). In contrast, if a website leverages cellular networks for authentication, it can identify the location of the user.

1.1. Overview

We design WebCallerID, a scheme that leverages cellular networks for secure Web authentication. WebCallerID involves the user, the relying party (the website that the user wishes to authenticate to), and the cellular network, as shown in Fig. 1.

As a proof of concept, we implemented WebCallerID in the OpenID framework [26]. OpenID is a single sign on (SSO) identity system, where the user authenticates himself to a third-party instead of authenticating himself to a destination website. This third-party is known as an *identity provider*, and verifies the user's identity. The identity provider then handles the subsequent authentication to other sites. OpenID purposely leaves out how a user authenticates to his identity provider. In practice, many OpenID identity providers use password authentication, which not

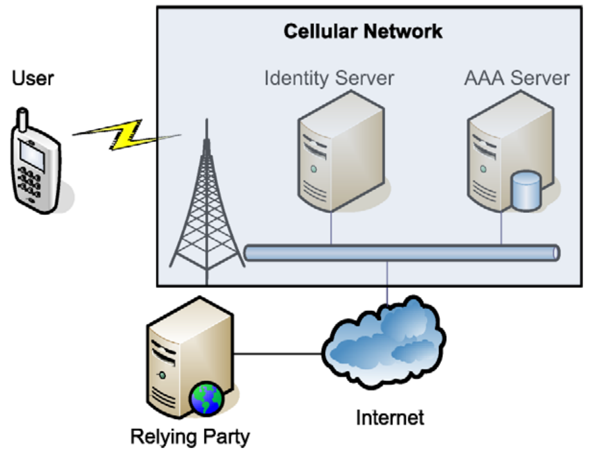


Fig. 1. Overview of the WebCallerID authentication scheme. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2011-0424.)

only inherits all the vulnerabilities of password authentication, but also could make phishing easier [42]. We propose to use the cellular network as an identity provider by setting up an OpenID server within the cellular network, which authenticates mobile devices using existing authentication mechanisms in cellular networks. Additionally, in current OpenID systems, the user has to specify his identity provider. By contrast, the user of our scheme need not specify his identity provider, because the relying party (the website) can automatically detect the user's cellular network and his identity provider (Section 4.2).

Setup. The cellular network runs an OpenID identity server, which queries the existing AAA (Authentication, Authorization and Accounting) server for user profiles. The relying party accepts OpenID authentication. The user chooses an identity at his cellular provider for our scheme. An obvious choice is the user's phone number, but other choices can also be used to protect the user's privacy better (Section 5.4). Then, the user registers his identity at the relying parties that he plans to log in.

Authentication. The user can visit a relying party from a commodity browser running on one of the following devices:

- A cell phone with data services.
- A PC with cellular data services (e.g., via cellular data card).
- A PC that can initiate a connection with a cell phone (e.g., via Bluetooth, USB or infrared). If the user prefers, the PC may choose to use the cellular network just for handling authentication, and to use a non-cellular Internet connection for traffic to the rest of the Web. (Section 4.4).
- A PC not connected to a cell phone. In this case the user assists the authentication process by acting as the "connection" and copying a single string from the cell phone to the browser (Section 4.4).

After the user clicks an authentication button on the relying party's website, the website authenticates the user via his cellular network automatically.

1.2. Goals

WebCallerID has the following design goals:

- WebCallerID should be as secure as the authentication in cellular networks. In other words, if an attacker can break WebCallerID, he would have been able to break the cellular authentication. This implies that we do not consider such threats as an attacker stealing a mobile device, or controlling software on a mobile device.
- WebCallerID should not require users to participate in authentication. If users do not participate, they cannot make security mistakes. Another benefit is that users would get security for free – they do not have to perform any *security tasks*.
- WebCallerID should prevent phishing attacks both for authentication and privacy. A phishing website should not be able to replay a user's authentication credentials, nor to gain additional knowledge about the user's identity from the credentials.

2. OpenID

OpenID is an identity system where an *OpenID identity provider* (OP) acts as a trusted third party to authenticate a *user* to other *relying parties* (RPs). Using this identity system, a user no longer has to maintain different passwords for different relying parties, thereby avoiding security and usability problems with maintaining multiple passwords. The relying party only needs to be "OpenID enabled", but no modification is needed to the user's system software or Web browser. The identity provider runs an OpenID server, which communicates with the relying party and the browser to execute the authentication protocol. The popularity of OpenID is growing with major web companies like Google and Yahoo planning offerings of OpenID identity services for their users.

The OpenID authentication protocol follows these steps:

1. The user submits an *OpenID identifier*, a URI, at the login page of a relying party. This URI normally takes the form of a URL (e.g., http://openid.example.com/userid), though other forms are allowed in the protocol specification.
2. Based on the URI, the relying party discovers the user's OpenID server. Then, the relying party constructs an *authentication request* for the identity and returns this to the user's browser.
3. The browser forwards the authentication request to the OpenID server.

4. The OpenID server authenticates the user, signs a response, and returns the response (*authentication assertion*) to the user's browser.
5. The browser forwards the authentication assertion to the relying party.
6. The relying party verifies the assertion, by either forwarding it to the OpenID server, or by using a shared secret that the relying party has established beforehand with the OpenID server.

2.1. Security concerns

OpenID has become popular and trusted at many blogs, wikis, and social networking sites. However, in its current form, it is unsuitable for high-value websites, such as financial institutions, due to security and privacy concerns.

Phishing. Since relying parties no longer need to handle password authentication of users, it may seem that they are no longer susceptible to phishing attacks. However, the problem is only offloaded to the OpenID server. The OpenID specification purposely leaves out how the user authenticates to the OpenID server. Since popular OpenID systems authenticate users using passwords, they are still vulnerable to phishing attacks. An attacker could lure a victim user to authenticate to a rogue relying website, which then, by following the OpenID protocol, redirects the user to a rogue OpenID server. A phishing attack on OpenID server would be more serious than a phishing attack on individual websites, because the captured OpenID credentials would allow the attacker to log into all its relying websites.

Privacy. In popular OpenID systems, a user presents a single identity in each authentication with a relying party. This raises privacy concerns because multiple cooperating relying parties would be able to link a users activities to the same presented identity. Moreover, OpenID offers a mode of authentication, where the request is made without user interaction with the OpenID server. This authentication type is known as an immediate request. If this authentication succeeds, the relying party can learn the user's OpenID identity automatically when a users visits the relying party website. Simply visiting a website can harm the user's privacy.

A *directed identity* is an identity chosen by the OpenID server for an authentication request by a relying party. Instead of authenticating an identity specified by the relying party, the OpenID server selects an identity for the assertion response. In this authentication process, a relying party only needs to know of the OpenID server to construct an authentication request. A user does not need to give the relying party any personal identifiers at the start. After the authentication request is received, the OpenID server then allows the user to select and present different identities to relying parties. Presenting unique identities specific to each relying party can break attempts to track an identity across sites, however, this authentication process is supported in some but not all popular OpenID systems. Moreover, this burdens the user with managing multiple identities for different websites, which offsets some benefits of the OpenID system.

3. Design

WebCallerID uses the cellular network infrastructure as the trusted intermediary. To use this infrastructure, a user needs to visit a website using his cellular data connection. This can be done directly with a browser on a cell phone or from a PC using the cellular data service for Internet access. After the user clicks the login button, the website communicates with the identity server in the user's cellular network to authenticate the user. The authentication is transparent to the user – the user need not enter any information for the authentication. For compatibility with phones that cannot connect directly to a PC for data service, we also provide a scheme that allows the user to serve as the connection by transferring information from the phone into the browser.

3.1. Authentication protocol

Entities. Our authentication scheme involves a user and his cell phone, websites (relying parties) that the user wishes to authenticate to, and an identity server inside the user's cellular network.

Setup. The user registers his *directed identity* at each relying party. Section 3.2 describes how directed identities are computed. We assume that all messages in the protocol are encrypted and authenticated.

Authentication. When the user visits the relying party, authentication happens in the following steps:

1. The relying party sends a nonce n to the user's browser.
2. The browser discovers the identity server in the cellular network and forwards n to the identity server. The details of this process are discussed in Section 4.2.
3. The identity server queries the cellular network for the user's identity and computes the user's directed identity *ID* for this relying party. Then, it signs a tuple $t = (\mathit{ID}, n, \mathit{RP})$ using its private key, where *RP* is the identity of the relying party. The signed t is called an *authentication assertion*. The identity server sends the authentication assertion to the browser, which then forwards it to the relying party.
4. The relying party verifies n, *RP*, and the signature in the authentication assertion. If the verification succeeds, the relying party has authenticated the identity *ID*.

3.2. Identity

The identity server in the cellular network provides the user's identity to relying parties. Since OpenID was proposed to unify identity management, popular OpenID servers provide only a single identity of each user to all relying parties. This causes privacy concerns. Colluding relying parties could track a user's activities. Moreover, if a user is tricked into authenticating to a rogue relying party, even though the relying party cannot replay the user's authentication assertion, the relying party gets to know the user's identity, which could compromise the user's privacy. The *directed identity* feature of OpenID allows a user to select different identities for different relying parties. However, maintaining multiple identities would be burdensome for both the user and the identity provider.

In our scheme, the user selects a *master identity*, and the identity server computes a unique directed identity for each relying party. It uses the same approach as Pwd-Hash [30]. Let the user's master identity be ID_U, the relying party's identity (e.g., its domain name) be *RP*, and *PRF* be a pseudo random function. Then the directed identity of ID_U at *RP* is $PRF_{ID_U}(RP)$, where ID_U is used as the key to *PRF*, and *RP* is used as the input to *PRF*.

3.3. Security and privacy

Our scheme provides several security and privacy benefits:

1. The user need not remember or submit any passwords. Instead, the cell phone serves as the user's authentication token. Since our scheme uses no password, it avoids all password-related problems.
2. A relying party cannot replay its received authentication assertion to log into another relying party. The authentication assertion contains the relying party's identity signed by the identity server. As long as the relying party cannot forge its identity, it cannot use its authentication assertion to log into other relying parties. For example, the identity could be the relying party's domain name, or could come from credentials proving the relying party's identity, such as a signed public key certificate. We will describe how our scheme defeats both passive and active replay attacks in detail in Section 5.2.
3. A rogue relying party cannot discover the user's identity for other relying parties, since the user's identity at each relying party is the output of a *PRF* on the relying party's identity.

Thanks to the first benefit, Web authentication becomes transparent to the user. Since the user does not participate in the authentication (other than possessing his cell phone), he cannot mistakenly divulge his authentication credentials. Moreover, the user does not have to worry about phishing attacks – he can freely log into any website without security or privacy compromise during the authentication. This is because the website cannot reuse his authentication assertion (due to the second benefit), and cannot discover either his master identity or his directed identity at another website (due to the third benefit).[2]

[2]However, our scheme cannot protect the user from disclosing confidential information *after* login.

4. Implementation

As a proof of concept, we implemented WebCallerID based on the OpenID framework. WebCallerID involves three components: relying parties, a cell phone, and an OpenID server inside a cellular network. The OpenID server queries user profiles from the AAA (Authentication, Authorization and Accounting) server, which handles the authentication of the cellular device [8]. Sections 4.2–4.4 describe these three components in details.

4.1. Protocol

Figure 2 shows the messages in the protocol for authenticating a user U to a relying party *RP*:

1. The user U visits the login page of the relying party *RP* to initiate the authentication process.
2. The relying party *RP* obtains the identifier of the user's identity server *IS* via the process described below in Section 4.2.

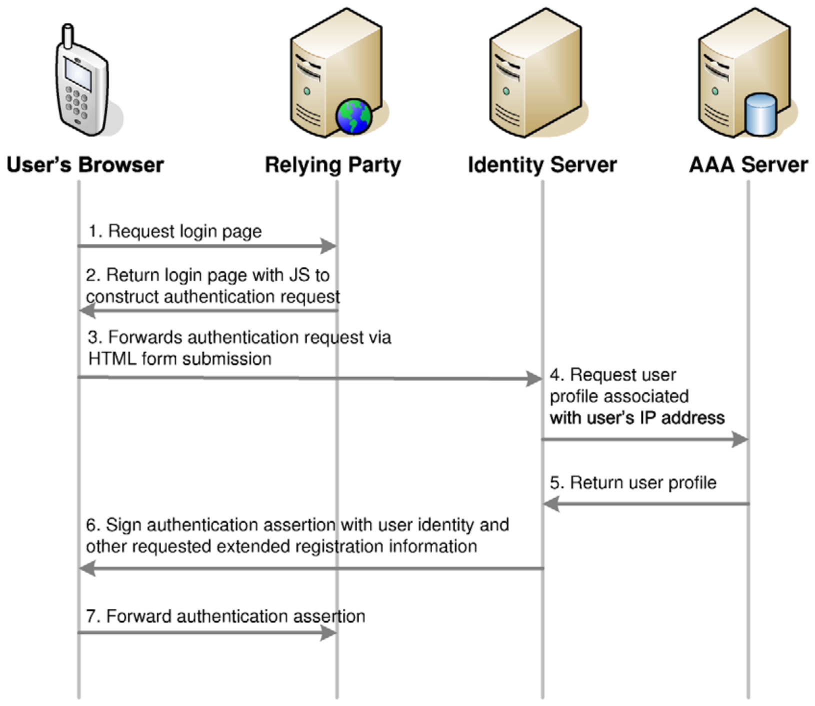

User's Browser
Relying Party
Identity Server
AAA Server
1. Request login page
2. Return login page with JS to construct authentication request
3. Forwards authentication request via HTML form submission
4. Request user profile associated with user's IP address
5. Return user profile
6. Sign authentication assertion with user identity and other requested extended registration information
7. Forward authentication assertion

Fig. 2. Messages during authentication. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2011-0424.)

3. The relying party *RP* redirects the user's browser to the identity server *IS* with an OpenID authentication request. The authentication request is an *OpenID immediate request* and contains a *return_to* parameter with an https URL pointing back to the relying party *RP* and a nonce.
4. Upon receiving an authentication request from the IP address IP_U, the identity server *IS* queries the cellular network's AAA server for the user that is currently assigned IP_U.
5. The AAA server returns the user's identity ID_U and some other information in his profile.
6. The identity server *IS* constructs an authentication assertion containing a directed identity $ID_{directed}$ computed from *RP* and ID_U (Section 4.3), a response nonce, other profile information, and a signature over the tuple of these values. Then, the identity server *IS* returns this authentication assertion to the user's browser to be forwarded to the relying party *RP*.
7. The user's browser forwards the authentication assertion to the relying party *RP*. *RP* checks the authentication assertion by verifying if the parameters match those of the authentication request, the response nonce has not been seen, and the signature is valid.

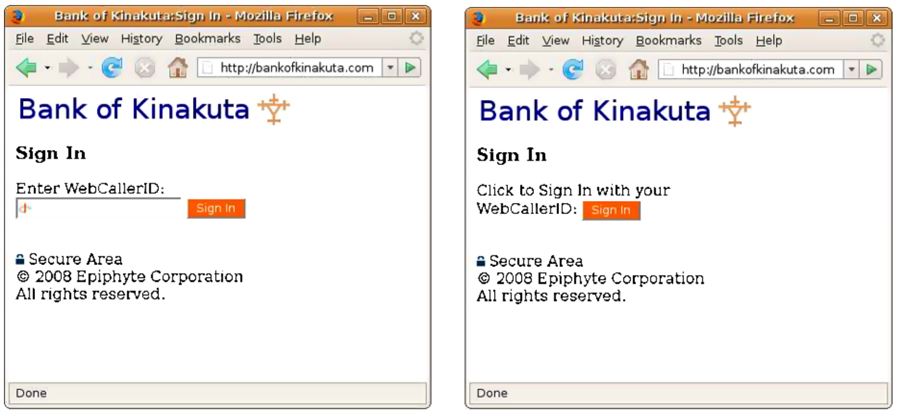
4.2. *Relying parties*

A relying party is a website with an authentication component that accepts and verifies OpenID identities. The relying party may completely use this component as the sole piece of authenticating information or use it in conjunction with other authentication systems in a multi-factor authentication scheme.

We have implemented two types of relying party authentication components. The first is a standard OpenID authentication component, which requires the user to enter his identifier. The other requires no extra user interaction and simply consists of a JavaScript module that uses the cellular network connection to complete authentication when started. If it detects no cellular network connection, it can automatically fall back to the first scheme.

Since the first relying party authentication component has the same interface for entering an identifier as in OpenID, it is compatible with existing OpenID infrastructure. We use this component for cellular users who do not have browser access to the internet via their cell phone. On the login page of the relying party, the user enters his identifier, which is a URI. Based on this identifier, the relying party discovers the OpenID server inside the cellular network by following standard OpenID protocols. Figure 3(a) shows the screenshot of a login page.

The above approach, however, is not transparent to the user, as he needs to enter his identifier. To remove this requirement, we implemented a second type of relying party authentication, which requires no user input. The login page of such a relying party simply ties authentication to the *login* button (Fig. 3(b)). After the user clicks the button to login (possibly after entering the information necessary for another


Bank of Kinakuta:Sign In - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://bankofkinakuta.com

Bank of Kinakuta

Sign In

Enter WebCallerID:

Sign In

Secure Area
© 2008 Epiphyte Corporation
All rights reserved.

Done

(a)

Bank of Kinakuta:Sign In - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://bankofkinakuta.com

Bank of Kinakuta

Sign In

Click to Sign In with your
WebCallerID: Sign In

Secure Area
© 2008 Epiphyte Corporation
All rights reserved.

Done

(b)

Fig. 3. Two types of login pages of relying parties. The login page using WebCallerID authenticates the user automatically. Depending on the web application, this authentication may be sufficient to imply authorization. In the case of security sensitive applications, additional authentication factors can be used to verify user intent and authorization. (a) A login page that requires the user to enter his identity. (b) A login page that automatically acquires the user identity from a WebCallerID server. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2011-0424.)

authentication system), the page runs a JavaScript module. This module contains a list of known OpenID servers inside the cellular networks that the relying party trusts. The module tries to initiate an OpenID discovery process with these OpenID servers until one of them succeeds.[3] The OpenID server inside the cellular network returns the user's identity to the relying party upon successful authentication.

The above JavaScript module has 20 lines of code. To probe an OpenID server, the module creates an *iframe*, and redirects the iframe to the OpenID server with an authentication request that has a *return_to* parameter specifying the relying party. If the authentication is successful, the OpenID server returns the authentication assertion to the browser with a redirect to the *return_to* parameter provided. The module then can allow the relying party's Web application to proceed with the authenticated user session.

4.3. OpenID server

We created our OpenID identity server on the test network of a major US cellular provider. Our identity server is based on the OpenID 2.0 Draft specification and is composed of less than 800 lines of Python code using the Python OpenID Library [29]. The OpenID identity server authenticates users in the cellular network

[3] As an optimization, the module could use the IP address of the cell phone to narrow its list of OpenID servers, or could use a standard name that resolves to a local identity server in all cellular networks. We did not implement this optimization.

to relying parties. It services authentication requests from users as well as optional association requests from relying parties.

The identity server receives authentication requests from the user's cell phone via the cellular network. The identity server fulfills these OpenID authentication requests by leveraging the cellular network's authentication infrastructure, the AAA (Authentication, Authorization and Accounting) server, that is handling the authentication of the cellular device. The AAA server authenticates clients requesting access to the network and maintains session state for the duration of the client's use of the network. This state includes the MIN (mobile identification number) and the assigned IP address for the client's session.

Our identity server queries a Bridgewater Systems AAA Service Controller v.8.2.1 server inside the cellular network to obtain the user profile currently associated with the source IP address of the authentication request. It makes one LDAP query of the AAA server per authentication request. Based on the user profile, the identity server derives the user's identity at the relying party, creates an authentication assertion, and sends the assertion back to the relying party.

Since the identity server derives the user's identity from the IP address of his cell phone, we must ensure that these IP addresses cannot be spoofed. Fortunately, to obtain fraudulent authentication assertions, an attacker would have to not only spoof the source IP of the authentication request, but also be able to receive the authentication assertion at the spoofed IP address. Such spoofing is infeasible because of the encrypted PPP sessions between the cell phone and the cellular switch, which is often referred to as the Packet Data Serving Node (PDSN).

To provide a directed identity to the relying party (i.e., to provide different identities to different relying parties), for each user the server maintains a master identifier, ID_U, that is not shared with any relying parties. Instead, the server constructs a directed identity for each relying party RP as $ID_{directed} = PRF_{ID_U}(RP)$, where PRF is a pseudo random function. By constructing a directed identity on the fly, the server need not maintain records per relying party per user. The server uses the domain name and, if available, the organizational name from the SSL certificate of the relying party for RP. We leverage the existing SSL certificate infrastructure to authenticate the identity of the relying party to prevent attacks by taking over expiring domains.

Optionally, the server may send extended registration information obtained from AAA's user profile to relying parties. Such information may include the name, verified billing address, mobile phone number and email address of the user. The server can also obtain the phone's location from the cellular network's Mobile Positioning Center (MPC) server. Combined with a mobility profile, a history of the user's previous locations, constructed by the cellular network, the identity server can even identify anomalous authentication attempts and make this known to the relying party. This location information could be processed in a manner similar to card-present credit card transactions to identify authentication requests made when the phone is lost or stolen. The relying party may use this additional information as part of its authentication or authorization decisions, and regulate access to services based on their values. For privacy concerns, the server should only release such information to the websites that the user has consented to during the registration phase. Note that in this case, the server would need to maintain a list of authorized websites per user.

4.4. Client software

Since OpenID requires no special capabilities in the user's browser or extra software on the client machine, our implementation entirely relies on the browser to complete the authentication. We require the browser to have access to the cellular network or the user to act as an intermediary between the browser and his cellular phone to complete an authentication transaction. We discuss how to satisfy this requirement in various scenarios below:

Direct access to the cellular network. When the browser runs on a cell phone or on a PC with a cellular card, it can directly access the cellular network.

Bridged access to the cellular network. When the browser runs on a PC without a cellular card, the user can create a connection between the PC and his cell phone (e.g., via Bluetooth or infrared). This way, the PC uses the cell phone as a wireless modem.

Multihomed access to the cellular network. When the user wishes to access the Internet via a non-cellular network (due to bandwidth, cost, or other considerations), he can use the cellular network just to handle authentication requests. He can achieve this by configuring the client device (where the browser runs) in a multihomed setup with connections to both the non-cellular Internet provider and the cellular network (Fig. 4). For example, a user may primarily use a WiFi connection on a laptop computer, but can also tether a cellular phone for Internet access via the cellular network. Such a multihomed user may wish to use the cellular network for authentication but prefer to use the non-cellular network for subsequent Web sessions. To achieve this, we set up the routing table on the laptop to send traffic to the identity server through the cellular network and to use the non-cellular network for all other traffic. Such a change could be automatically made by driver software for the phone.

No access to the cellular network from the browser. If the user is unable to connect the phone to the computer, the browser running on the computer would have no access to the cellular network. However, if the user can act as an intermediary between the phone and browser, we can modify the authentication protocol in Fig. 2 as follows. The protocol starts when the user enters his identifier on the login page of the relying party. After the relying party constructs the authentication request and sends it to the identity server (Step 3), the identity server on the cellular network authenticates the user via a separate channel on the cellular phone and the URL bar of the browser, instead of via the AAA server (Steps 4 and 5) as normally done. This separate channel can be messages sent over a secure short message service

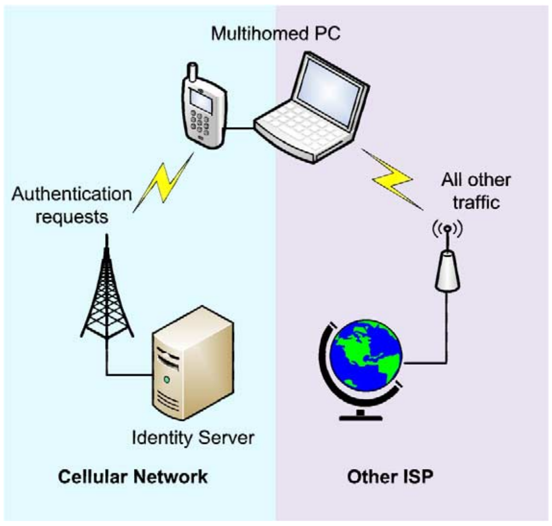


Fig. 4. Multihomed setup for authentication. The browser runs on the PC, which connects to both a cellular network and a non-cellular network. We configure the PC's routing table to send traffic to the OpenID server through the cellular network, and to send all the other traffic through the non-cellular network. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2011-0424.)

(SMS), multimedia message service (MMS), or other messaging systems that guarantee that senders are unforgeable and that only the intended recipient can read the message.

The identity server responds to the authentication request by constructing a unique secure (https) URL associated with the authentication transaction and sends it to the user via the messaging channel. The message instructs the user to navigate his browser to this URL. This action authenticates the user to the identity server and allows the identity server to respond with the authentication assertion.

The identity server authenticates the user in this manner since the secure URL is a nonce that the user would only know if he possessed the cell phone. The cellular network protects the security of the nonce from the identity server to the message on the phone via the secure messaging system, and the browser protects it with SSL/TLS when the user enters the secure URL in the URL bar.

There exist similar SMS authentication schemes that send the user a code via SMS, which the user then enters into a login page [31]. However, those schemes are vulnerable to the following man-in-the-middle attack. If the user visits the login page of a phishing website, the phishing website could request the real website to send a code via SMS to the user. When the user enters the code into the login page controlled by the phishing website, the phishing website could immediately reuse the code to log into the real website. By contrast, in our scheme, even if the user visited the login page of a phishing website, the phishing website cannot capture the secure

URL sent to the user via SMS, because the user enters the secure URL directly into the URL bar, which is not under the control of the visited website.[4]

5. Security evaluation

5.1. Security assumptions

The security of WebCallerID relies on the following:

- *Secure authentication inside cellular networks*. We assume that cellular networks can authenticate their phones securely. In other words, we assume that it would be difficult for an attacker to impersonate another user without possessing the user's cell phone. Moreover, we assume that no cell phone can spoof its IP address *and* receive packets at the spoofed address from the OpenID server inside the cellular network. Such spoofing is infeasible because of the encrypted PPP sessions between the cell phone and the PPP-gateway. Moreover, since the MAC address of each cell phone is assigned by a base station and is not spoofable, the PPP-gateway can look up the allocated IP address of the MAC address and perform ingress-filtering.
- *Uncompromised cell phones*. We assume that cell phones are under the control of their users, and their software has not been compromised. Once a user chooses to use his cell phone for Web authentication, the cell phone becomes a physical access token to his accounts. Users already have experience in protecting physical access tokens, such as bank cards, and such experience will help them protect their cell phones. In fact, many cell phone users already take steps to protect their phones from unauthorized use, due to the risk of personal information disclosure and the cost of unauthorized use. Some phones use passwords or biometrics for controlling access to restrictive functions (such as blocking outgoing numbers) [2,13]. We would apply similar techniques for restricting access to the Web authentication function.
  Malware could be a potential problem. We do not specifically address this problem in this work, but it is a well-researched area. As cellular phones become more powerful, we could apply techniques from malware defense on PCs. Virtual machines can securely separate trusted programs from untrusted ones. For example, pocket hypervisors [7] can provide virtualization services for mobile computing platforms like cellular phones. A secure operating system running on such a platform can run the browser in one virtual machine, run other user

[4]Typosquatting could pose a risk. If the user mistypes the domain name the URL, an attacker squatting at the mistyped domain name could capture the URL, replace the mistyped domain name with the correct domain name, and use it to log into the real website. However, this attack works only if the user mistypes the domain name *and* accurately types the rest of the secure URL (because the rest of the URL serves as a nonce to be verified by the identity server).

applications in separate virtual machines, and allow only the browser's virtual machine to contact the identity server in the cellular network. Other work on securing cell phones from malware can be found in [4,6,15,41].

- *Secure channel between the relying party and the browser*. We assume that an authentication assertion can be read only by the relying party. If the attacker is passive, we can satisfy this assumption by using end-to-end encryption, such as SSL/TLS. However, if the attacker is active, we would need mechanisms, such as PKI, for preventing man-in-the-middle attacks.

5.2. Security benefits

Compared to password-based authentication, WebCallerID provides the following benefits:

- Users no longer participate in security decisions. They do not have to remember passwords. This not only simplifies users' tasks but also avoids bad passwords. Since users do not have to submit credentials, they cannot make security mistakes, such as falling victim to phishing attacks.
- Users no longer have to worry about rogue relying parties. Phishing websites with login pages indistinguishable from the real relying party cannot trick users into incorrectly authenticating to them. Users do not need to correctly identify a site by its URL. In WebCallerID, a user's credentials are created by his OpenID provider and are tied both to a specific relying party and a nonce selected by the relying party. A user may send his credentials to a malicious relying party; however:
  - If the malicious relying party simply creates a rogue site to collect authentication credentials from visitors in an offline phishing attack, it cannot replay credentials to authenticate to a target relying party because the latter would expect the credentials to contain a new nonce.
  - If the malicious relying party engages in a man-in-the-middle attack, e.g., by tricking the user into visiting a deceptive URL, it still cannot replay the credentials to authenticate to a target relying party, because the credentials include the domain name of the target relying party.
  - If the malicious relying party is a man-in-the-middle and can compromise routing and DNS, we may defend against replay attacks as follows. The relying party includes a fingerprint of its public key in its authentication request (Message 2 in Fig. 2). The relying party also gives this public key to the user for establishing a session key (the user picks a session key, encrypts it with the public key, and sends the ciphertext to the relying party) for encrypting messages between the user and relying party. The OpenID server will include this fingerprint in the authentication credentials (Message 6 in Fig. 2), which the user's browser relays to the relying party (Message 7 in Fig. 2). The relying party then checks the credentials for the fingerprint of its public key.

If a malicious relying party attempting a man-in-the middle attack sends his public key to the user instead, the OpenID server will create credentials including the fingerprint of the malicious party's public key, which will be rejected by the target relying party. On the other hand, if the malicious relying party sends the target relying party's public key to the user, the malicious relying party will be unable to eavesdrop on or modify subsequent messages between the user and the target relying party because the messages are encrypted by a session key that has been picked by the user and encrypted by the public key.

Currently, SSL/TLS with properly signed certificates from a PKI is used to prevent these kinds of man-in-the-middle attacks. However, it still relies on the user's ability to identify the correct URL of the site and judge that a certificate is valid. The browser still provides a mechanism for the user to bypass this security by accepting invalid certificates. By contrast, our proposed scheme frees the user from having to do any work to verify the identity of the relying party or to understand certificate warnings, and prevents the user from accepting invalid certificates.

Compared to physical token-based authentication, WebCallerID provides the benefits of easy deployment, as cell phones are becoming more ubiquitous. Moreover, WebCallerID can be integrated into existing Web authentication infrastructure as long as the user possesses a data capable cell phone.

We implemented WebCallerID in the OpenID framework. WebCallerID solves a vexing weakness in typical OpenID systems, i.e., how to authenticate to OpenID providers securely? We leverage the cell phones as physical authentication tokens, and leverage existing authentication in cellular networks to provide OpenID authentication. Another advantage of WebCallerID is that relying parties can probe our OpenID providers automatically. This is in contrast to typical OpenID systems in which the user has to provide such information.

5.3. Multi-factor authentication

Web applications with greater security requirements, such as online banking, rely on a second authentication factor along with password authentication. Many of these systems face shortcomings because they are vulnerable to the same man-in-the-middle attacks as passwords [34,35]. WebCallerID can function as one component of a multi-factor authentication system, but is not vulnerable to these man-in-the-middle attacks. It does not preclude other schemes, and is in fact complimentary to other authentication schemes. It introduces a "something you have" component to an authentication process that currently in practice mostly relies on "something you know". Furthermore, adding this authentication scheme presents no additional work for users in the authentication process if the user can pass data packets via his mobile device. With cellular data services becoming more ubiquitous, WebCallerID provides a more secure alternative to password authentication, and a more usable alternative to physical token-based authentication.

5.3.1. *Strengthening WebCallerID with other factors*

The threat model of WebCallerID (Section 5.1) assumes that the mobile device running WebCallerID is accessible only to its legitimate owner. In other words, WebCellerID assumes that the owner delegates her authority to her mobile device. If the owner worries that her mobile device may fall outside her control at times, she could use other authentication factors to protect her mobile device and WebCallerID (This is analogous to using passphrases for protecting private keys in the public key authentication method in SSH). These factors could include text or graphical passwords, biometrics, etc. The authentication process using these factors could happen either locally on the mobile device or remotely at the identity server. In the former case, the device would require necessary software to authenticate the user and to mediate access to the identity server. In the latter case, the device would need to collect user credentials and forward them to the identity server. These factors would strengthen WebCallerID by protecting it against unauthorized access to mobile devices.

5.4. *Privacy*

WebCallerID protects the user's identity by sending his directed identity to relying parties (Section 3.2). Directed identities are unique to each relying party, and it would be infeasible for one relying party to derive the user's directed identity at another relying party or the user's master identity, barring a brute force or a dictionary attack on the master identity. Therefore, users should choose master identities that withstand these attacks. For example, users' phone numbers are a bad choice for their master identities, because they have insufficient entropy. Alternatively, since users need not remember their master identities, our OpenID server could assign strong master identities to users.

Even though a rogue website cannot steal the user's master identity, the website could compromise the user's privacy if the user enters his confidential information later in the Web session. This is out of the scope of this paper.

Generally, using a third-party identity service instead of authenticating to each website directly collects a user's authentication records at one location, the identity server. As such, the user gives up the privacy of his authentication records at the identity server. However, when the user accesses the Internet via a cellular network, the cellular network already knows the websites that the user visits, so the user gives up no additional privacy when he uses the identity server provided by the cellular network.

5.5. *Deployment*

Web users. WebCallerID, like OpenID, requires no special capabilities in the user's browser or extra software on the client machine. It only requires that the user can access a cellular network, either directly from his browser or by serving as an intermediary between the browser and his cell phone (Section 4.4).

Relying parties. Relying parties, who are consumers of authentication assertions, need to trust the cellular networks that run OpenID identity servers. Given the moderate number of major cellular providers, we think that establishing this trust is administratively feasible.

Cellular networks. By contrast, cellular networks do *not* need to trust relying parties. As described in Section 5.2, an authentication assertion provided by the cellular network for one relying party is not reusable at another relying party. Technically speaking, a cellular network need no prior knowledge about a relying party before providing authentication assertions for the relying party.[5] There are apparent incentives for cellular networks to provide this authentication service, such as increased revenue from more data usage and more subscribers. On the other hand, cellular networks also need to assess risks associated with this service, such as legal liability and inadvertent leak of users' or networks' private information to relying parties.

6. Performance evaluation

We evaluated the performance of WebCallerID. On the server, we measured the throughput of authentication responses; on the client, we measured the duration of authentication sessions.

Since our goal is to demonstrate the feasibility of WebCallerID rather than developing a production system, we compared the performance of WebCallerID to that of popular Web authentication schemes running on the same stock desktop PC in our lab. In a production system where WebCallerID is running on powerful servers, we expect its performance to be far better.

6.1. Setup

Our test machine ran an Ubuntu 9.04 64-bit system, with an Intel Core 2 Duo 1.86 GHz CPU, 3 GB DDR3 RAM, and 1 GBps ethernet. We tested three authentication systems on this machine:

- WebCallerID identity server.
- Apache (2.2.11) HTTP Server's basic password authentication.
- Django Web framework (1.0.2)'s user password authentication, served through Apache's mod_python (3.3.1).

The Apache HTTP Server [1] is the most popular web server [24] deployed today. It provides a basic password authentication mechanism where authentication credentials are provided to the server in an HTTP request. The server then verifies the credentials against entries in a password file.

[5]Due to business and legal concerns, the cellular network might wish to enter into an agreement with each relying party.

Django [11] is a popular web application framework for the Python language with tens of thousands of users [17]. As part of the framework, it provides a user authentication component. Web applications developed with Django allow users to submit credentials to authenticate themselves to the web application. Credentials are then checked by the Django authentication module against user records in a database.

A user authentication for WebCallerID identity server consists of two HTTP requests for OpenID discovery and one HTTP request for the authentication request. Password authentication for Apache and Django requires only a single HTTP request to submit the authentication credentials to the server. For each of these authentication systems, we preloaded 10,000 user profiles and measured its throughput separately.

6.2. Authentication server

We tested the performance of the WebCallerID identity server as it responded to authentication requests from relying parties. Using Apache JMeter [21], we simulated a workload of 10,000 client requests, with 1, 10 and 100 concurrent users on a 1 GBps local network.

Table 1 compares the authentication throughput of the WebCallerID identity server to that of the Apache basic password authentication and that of the Django Web framework's user password authentication. Among the three servers, Apache has the best throughput, which is 5.3 times the throughput of WebCallerID. This is mainly because the WebCallerID identity server serves three HTTP requests during one authentication session, while Apache serves only one HTTP request. Moreover, Apache is written in C while we implemented the WebCallerID identity server in python. Since Django is also written in python, comparing WebCallerID to Django is more illustrating. The Django authentication process also only takes one HTTP request, but the additional overhead of the extra work done in the framework slows it down.[6] Table 1 shows that the throughput of WebCallerID is 2.3 times that of Django.

Table 1

Comparison of authentication server throughput

	Apache	Django	WebCallerID
HTTP requests/authentication	1	1	3
Authentications/second	405	33	77

[6]We investigated the sources for Django's slow performance. We extracted the authentication operation of Django and WebCallerID and tested them in isolation. For one authentication operation within the server, Django spends 0.08 ms in the python MySQLdb query for user credentials, while WebCallerID spends 0.14 ms in the python-LDAP query for user credentials. However, Django's python MySQLdb query accounts for only 9.6% of its execution time (Django spends the rest of its execution time in its object-relational mapper, etc.), while WebCallerID's python-LDAP query accounts for 83% of its execution time. Therefore, overall one Django user authentication takes about 5 times longer than one WebCallerID user authentication.

6.3. Client

We wrote a client script to measure the duration of one WebCallerID authentication. We chose not to measure this duration directly in the browser because it is more precisely to time our script and we wish to exclude other factors, such as parsing and rendering, in the browser that may affect the timing measurement.

Figure 5 shows the duration of one authentication session with 10 concurrent users. Figure 6 shows the average duration of one authentication session with 1, 10 and 100 concurrent users. Again, although the duration of the Apache basic authentication is the shortest, the duration of the WebCallerID authentication is 2.49 times shorter than the Django user password authentication. The average duration of a

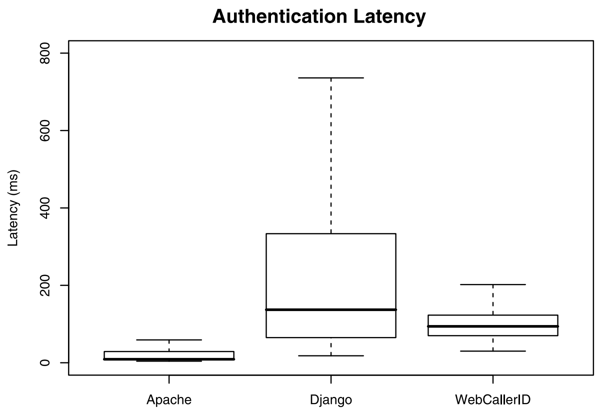


Fig. 5. Duration of client authentication with 10 concurrent users.

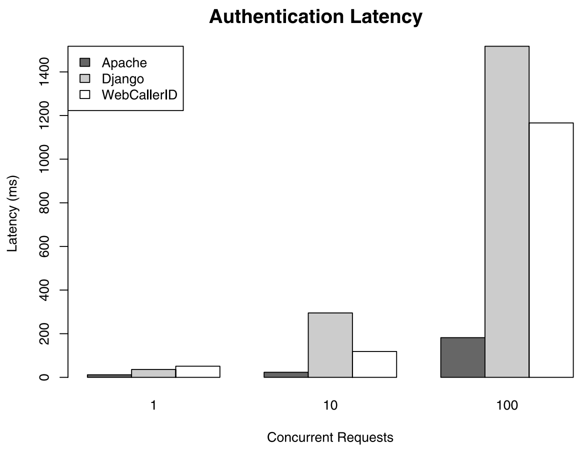


Fig. 6. Average duration of client authentication with 1, 10 and 100 concurrent users.

WebCallerID authentication is 50, 118 and 1166 ms for 1, 10 and 100 concurrent users, respectively, which imposes a small delay in the authentication process. By comparison, it takes a typical users at least several seconds to enter a password during password-based authentication.

7. Related work

The ideas behind Kerberos [25] are found in many of today's Web authentication services, such as OpenID [26] and CardSpace [5]. A Kerberos server acts as a trusted party to verify a user's identity and relays this claimed identity to other parties with a secure protocol. We choose the OpenID framework to implement a prototype of our authentication scheme, because of the advantage that this framework requires no modification to a user's system software or Web browser. We could implement our scheme in other identity systems, such as Microsoft's CardSpace. A disadvantage of CardSpace is that it requires an operating system software client currently only available on the Windows OS. Our implementation of directed identity borrows its idea from PwdHash [30].

Authentication based on physical tokens overcomes some intrinsic problems with password authentication. RSA SecurID is a typical example [32]. However, physical tokens often face deployment obstacles, due to the cost and inconvenience of carrying the tokens. By contrast, cell phones are becoming ubiquitous. Moreover, most physical tokens either require additional software (device drivers or special applications) or require the user to relay data between them and the browser. By contrast, our scheme requires no extra software on the user's machine and no participation from the user during authentication when his browser can access a cellular data network.

When physical tokens are unavailable, researchers have proposed alternatives to standard password authentication, such as graphical passwords [9,20,39], secrets derived from "personal entropy" [12], and human-executable computations [18]. They authenticate users with a challenge, whose response the user can compute. While these solutions break simple phishing attacks that steal passwords for replay, they are still vulnerable to man-in-the-middle attacks. Also unlike our system, they require users to participate actively in the authentication. Experience shows that it is difficult to design secure schemes that require humans to participate in a challenge response protocol [14].

Several systems use a mobile phone to aid authentication. One class of solutions is designed to protect against keyloggers on untrustworthy clients, such as an Internet kiosk [19,23,40]. They require a separate trusted system to act as a secure proxy. The SMS channel of the mobile phone serves as an out-of-band traffic channel via which the user can obtain or submit secret information. These systems defend against a different threat from our authentication scheme and require a user to participate in security tasks with the cell phone for authentication.

Another class uses an external trusted system to provide mutual authentication of the user and the website. These systems use the mobile phone to store secrets and verify the identity of the remote site [16,22,27]. Storing the user's secrets on the mobile phone prevents the users from inadvertently disclosing them in a phishing attack, but still allows for user-friendly secure authentication when the phone can communicate with the browser. However, it requires browser modifications and additional software.

Wangensteen et al. and Van Thanh et al. use secrets stored on the SIM card itself as authentication credentials, and also leverage the authentication of the SIM for a single sign on system [36–38]. Because they rely on the SIM as part of the trusted platform, their scheme does not work on networks that do not use SIM cards. Also, it either needs additional Java software to run directly on the mobile device or requires the user to participate in the authentication process via SMS. Deploying additional software to a heterogeneous set of devices can be difficult for a service provider. By contrast, our approach requires no additional software beyond a basic browser, which is becoming standard on modern cell phones. Additionally, our approach does not require the user to participate in the authentication process when the browser can access the cellular network directly, which is also becoming common.

As cell phones become more ubiquitous and powerful, banks have introduced banking services using cell phones. For example, PayPal Mobile [28] allows users to send payments using SMS. However, due to the inherent limitations of SMS, it cannot support the rich, interactive functions that we enjoy in Web applications. SMS has also been used as a secondary factor for strengthening Web authentication, but it requires substantial user involvement. Banks have also deployed Internet banking services for cell phones, which users access via either a browser or a special application [3]. However, these services use the cellular network simply as an Internet service provider, and fail to leverage the built-in authentication in the cellular network for enhancing their security.

8. Conclusion

We designed WebCallerID, a Web authentication scheme that leverages the authentication infrastructure inside cellular networks. WebCallerID uses cell phones as physical tokens and uses cellular networks as trusted identity providers. Using this scheme, a website authenticates a user with the help of an identity server inside the cellular network. Since the authentication procedure is transparent to the user and requires no user involvement, it prevents users from making security mistakes. Moreover, WebCallerID prevents rogue websites from replaying authentication assertions or stealing user's identities. Our prototype implementation shows that WebCallerID not only seamlessly integrates into OpenID-capable authentication, but also prevents phishing problems in OpenID and simplifies user participation in OpenID authentication.

Acknowledgments

This work was partially supported by the National Science Foundation through grants CNS 0644450 and 1018964. We wish to thank Jean Bolot, Mukund Seshadri and Hui Zang for their insightful suggestions.

References

[1] Apache http server project, http://httpd.apache.org/.

[2] Apple iPhone user guide, http://manuals.info.apple.com/en/iPhone_User_Guide.pdf.

[3] Bank of America mobile banking, http://www.bankofamerica.com/onlinebanking/index.cfm?template=mobile_banking.

[4] A. Bose, X. Hu, K.G. Shin and T. Park, Behavioral detection of malware on mobile handsets, in: *MobiSys'08: Proceeding of the 6th International Conference on Mobile Systems, Applications, and Services*, New York, NY, USA, 2008, ACM, pp. 225–238.

[5] D. Chappell, CardSpace, 2006, available at: http://msdn2.microsoft.com/en-us/library/aa480189.aspx.

[6] J. Cheng, S.H. Wong, H. Yang and S. Lu, Smartsiren: virus detection and alert for smartphones, in: *MobiSys'07: Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, New York, NY, USA, 2007, ACM, pp. 258–271.

[7] L. Cox and P. Chen, Pocket hypervisors: opportunities and challenges, in: *Eighth IEEE Workshop on Mobile Computing Systems and Applications, 2007, HotMobile 2007*, March 8–9, 2007, pp. 46–50.

[8] C. de Laat, G. Gross, L. Gommans, J. Volbrecht and D. Spence, Generic AAA Architecture, August 2000, RFC 2903.

[9] R. Dhamija and A. Perrig, Déjà Vu: a user study using images for authentication, in: *Proceedings of the 9th USENIX Security Symposium*, 2000.

[10] R. Dhamija, J.D. Tygar and M. Hearst, Why phishing works, in: *CHI'06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2006, ACM, pp. 581–590.

[11] Django, http://www.djangoproject.com/.

[12] C. Ellison, C. Hall, R. Milbert and B. Schneier, Protecting secret keys with personal entropy, *Future Generation Computer Systems* **16**(4) (2000), 311–318.

[13] Fujitsu f902i, http://www.fmworld.net/product/phone/f902i/.

[14] P. Golle and D. Wagner, Cryptanalysis of a cognitive authentication scheme (extended abstract), in: *SP'07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007, pp. 66–70.

[15] C. Guo, H.J. Wang and W. Zhu, Smart-phone attacks and defenses, in: *Proceedings of the Third Workshop on Hot Topics in Networks: HotNets III*, November 2004.

[16] S. Hallsteinsen, I. Jorstad and D.V. Thanh, Using the mobile phone as a security token for unified authentication, in: *ICSNC'07: Proceedings of the Second International Conference on Systems and Networks Communications*, IEEE Computer Society, 2007, p. 68.

[17] A. Holovaty and J. Kaplan-Moss, *The Django Book*, 2009.

[18] N.J. Hopper and M. Blum, Secure human identification protocols, in: *7th International Conference on the Theory and Application of Cryptology and Information Security*, Lecture Notes in Computer Science, Vol. 2248, 2001, pp. 52–66.

[19] R.C. Jammalamadaka, T.W. van der Horst, S. Mehrotra, K.E. Seamons and N. Venkasubramanian, Delegate: a proxy based architecture for secure website access from an untrusted machine, in: *ACSAC'06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, 2006, pp. 57–66.

[20] I. Jermyn, A. Mayer, F. Monrose, M.K. Reiter and A.D. Rubin, The design and analysis of graphical passwords, in: *Proceedings of the 8th USENIX Security Symposium*, 1999.

[21] JMeter, http://jakarta.apache.org/jmeter/.

[22] M. Mannan and P.C. van Oorschot, Using a personal device to strengthen password authentication from an untrusted computer, in: *Financial Cryptography and Data Security (FC'07)*, 2007.

[23] R. McMillan, Mobile phones help secure online banking, 2007, available at: http://www.pcworld.com/article/id,137057-c,onlinesecurity/article.html.

[24] Netcraft web server survey, http://news.netcraft.com/archives/web_server_survey.html.

[25] B.C. Neuman and T. Ts'o, Kerberos: an authentication service for computer networks, *IEEE Communications Magazine* **32**(9) (1994), 33–38.

[26] OpenID, http://openid.net.

[27] B. Parno, C. Kuo and A. Perrig, Phoolproof phishing prevention, in: *Financial Cryptography*, 2006, pp. 1–19.

[28] PayPal Mobile, https://www.paypal.com/us/cgi-bin/webscr?cmd=xpt/cps/mobile/MobileOverview-outside.

[29] Python OpenID library, http://openidenabled.com/python-openid/.

[30] B. Ross, C. Jackson, N. Miyake, D. Boneh and J.C. Mitchell, Stronger password authentication using browser extensions, in: *Proceedings of the 14th Usenix Security Symposium*, 2005.

[31] RSA mobile, http://www.rsa.com/press_release.aspx?id=1370.

[32] RSA SecurID, http://www.rsa.com/node.aspx?id=1156.

[33] J. Saltzer and M. Schroeder, The protection of information in computer systems, *Proceedings of the IEEE* **63**(9) (1975), 1278–1308.

[34] S.E. Schechter, R. Dhamija, A. Ozment and I. Fischer, The emperor's new security indicators, in: *SP'07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 2007, pp. 51–65.

[35] B. Schneier, Two-factor authentication: too little, too late, *Commun. ACM* **48**(4) (2005), 136.

[36] D. van Thanh, T. Jonvik, B. Feng, D. van Thuan and I. Jorstad, Simple strong authentication for internet applications using mobile phones, in: *Global Telecommunications Conference, 2008, IEEE GLOBECOM 2008*, November 30–December 4, 2008, pp. 1–5.

[37] A. Wangensteen, L. Lunde, I. Jørstad and T. van Do, Secured enterprise access with strong SIM authentication, in: *EDOC*, 2006, pp. 463–466.

[38] A. Wangensteen, L. Lunde, I. Jorstad and D. van Thanh, A generic authentication system based on SIM, in: *ICISP'06: Proceedings of the International Conference on Internet Surveillance and Protection*, IEEE Computer Society, 2006, p. 24.

[39] D. Weinshall, Cognitive authentication schemes safe against spyware (short paper), in: *SP'06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, 2006, pp. 295–300.

[40] M. Wu, S. Garfinkel and R. Miller, Secure web authentication with mobile phones, in: *DIMACS Workshop on Usable Privacy and Security Software*, 2004.

[41] L. Xie, X. Zhang, A. Chaugule, T. Jaeger and S. Zhu, Designing system-level defenses against cellphone malware, in: *SRDS'09: Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, Washington, DC, USA, 2009, IEEE Computer Society, pp. 83–90.

[42] K.-P. Yee, Phishing and OpenID: bookmarks to the rescue?, available at: http://usablesecurity.com/2007/01/20/phishing-and-openid/.