Usable Access Control for the World Wide Web

Dirk Balfanz
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304
balfanz@parc.com

Abstract

While publishing content on the World Wide Web has moved within reach of the non-technical mainstream, controlling access *to published content still requires expertise in Web server configuration, public-key certification, and a variety of access control mechanisms. Lack of such expertise can result in unnecessary exposure of content published by non-experts, or can force cautious non-experts to leave their content off-line. Recent research has focused on making access control systems more flexible and powerful, but not on making them easier to use. In this paper, we propose a* usable *access control systems for the World Wide Web,* i.e., *a system that is easy to use both for content providers (who want to protect their content from unauthorized access) and (authorized) content consumers (who want hassle-free access to such protected content). Our system can be constructed with judicious use of conventional building blocks, such as access control lists and public-key certificates. We point out peculiarities in existing software that make it unnecessarily hard to achieve our goal of usable access control, and assess the security provided by our usable system.*

1. Introduction

On the World Wide Web, there are *content providers* and *content consumers*. Content providers *publish* content by making it available through Web servers. Content consumers view or otherwise consume content by pointing their Web browsers to the Web servers of the content providers. In the early days of the World Wide Web, almost everybody was, in addition to being a content consumer, also a content provider – if you were technical enough to connect to the Web, you were probably technical enough to know how to put up a home page. Then came a time when the balance shifted dramatically: On one side, there were a few portals such as Yahoo! springing up, which delivered vast amounts of content. On the other side, the proliferation of Web client software and online services such as AOL or MSN made millions of people content consumers. The result was a landscape in which we had relatively few significant content providers, and a large number of content consumers.

This picture is starting to change again. More and more users are becoming (small) content providers. Today, when you sign up with an ISP for Internet service, you usually get a few megabytes of "web space", which you can use to put up a web site. Content publishing software and services make it easier to publish content: Teenagers broadcast music they mix through Shoutcast channels. Users share songs or other files through peer-to-peer networks. Photo hobbyists put their photo collections online. In fact, some applications make it as easy as pushing a button to export nicely-formatted photo collections to a web-hosting service.

While it is getting easier and easier for small-time content providers to publish their content, it is not particularly easy to do so *securely*, *i.e.*, to allow content providers to easily specify who should have access to their content.

Large content providers can afford to manage databases of subscribed customers, request certificates from well-known certification authorities, and hire developers to put access control mechanisms in place. Small content providers (*i.e.*, individual users), however, often lack such resources and technical sophistication. They are left with three choices:

1. They do not put any access control on their content. If the content provider doesn't care who sees their content, this is clearly the correct and easy solution. Often, however, content providers wish to restrict access to their content, for example to protect their privacy. Another reason is that opening up copyrighted or otherwise protected material to unrestricted access makes those content providers liable under copyright or other laws.

2. Therefore, if content providers wish to restrict access to their content, they have to fight with whatever mechanism their content publishing software provides for access control. As we will see in Section 3, this may

present an unacceptable overhead both to the content publisher and content consumers.

3. Finally, content providers may therefore be forced not to publish their content at all, being dissatisfied with the choice between unrestricted access and having to master archaic access control mechanisms.

In this paper, we will investigate why controlling access to published content is not an easy thing to do – and point out some ways to rectify the situation. In Section 2, we will re-iterate the issue of *usable security*, and comment on related work. In Section 3 we will specifically look at some currently existing systems, and how they handle access control.

In Section 4 we will outline a content publishing system that is user-friendly, easy to use, and reasonably secure. Our system uses a custom-built Web server on the content provider's side, and off-the-shelf email and Web clients on the content consumer's side. Our Web server has a self-signed root certificate and issues client certificates to users. We construct simple access control lists based on known public keys of users. We take some care to minimize the overhead imposed to the user (*i.e.*, number of links to click on, number and nature of dialog boxes to deal with, *etc.*).

In Section 5 we will explain why it is difficult to implement such a system with the building blocks available today (such as commonly used Web browsers, commonly used authentication protocols, *etc.*). Finally, we draw some conclusions in Section 6.

2. Background

2.1. Usable Security

One of the reasons that security exploits happen is because users do not configure the security of their systems correctly. Security patches are being ignored, access controls lifted, security warnings about executable content such as macros dismissed, and protections turned off.

There are two main reasons for this behavior: One, the user doesn't understand what's going on. This is hardly the user's fault. For example, in its default setting, Internet Explorer will ask the user for a decision when a Web page tries to launch a "file or program in an IFRAME". We cannot expect the average user to understand the security implications of that decision (see [14] for more examples). The second reason is that even if the user understands what's going on, she may realize that the security mechanisms are in the way of whatever she wants to do. For example, security mechanisms often prevent the viewing of executable content delivered through email. If a user is curious what macro-enabled birthday card she received from her best friend, she will switch off the security mechanism in order to see the card. More examples can be found in [2].

The goal of *usable security* is to relieve the users from decisions they don't understand or that they don't want to make (most of them boil down to "you will either be insecure or you will be inconvenienced"). Security should be *implicit* and should *follow* what the user wants to do. In [13] Smetters and Grinter point out that SSH is a good example of usable security: it works just like `rlogin`, except it's more secure. The user specifies what she wants to do (log into another machine), and the security mechanism *follows*. In [8] Edwards *et al.* show how a collaborative groupware application can be outfitted with implicit security. In their application, users simply specify who should belong to their work group (which is something they need to do anyway), and this information is used by the system to enforce access control on system resources (access is restricted to members of that work group). Groove[1] is a commercial application that is similar in that respect – access to shared spaces is actually secured by SSL without the users necessarily being aware of the fact that access control mechanisms are in place.

Note that one usually pays a price for such convenience. In SSH, if users are not careful, the first connection to a new host is subject to a man-in-the-middle attack. Edwards *et al.*'s Casca application requires users to physically meet at some point, and Groove sends root certificates around by unsecured email. We believe, however, that the overall security provided by these applications is better than in conventional, "bullet-proof" approaches, since users are less likely to turn off security or mismanage their security settings.

For content publishing, implicit security means that we need to look for actions users have to perform anyway, and associate reasonable security mechanisms with them. For example, content providers are likely to announce the existence of their content to a circle of family and friends, for example by sending them an email. We can use this action to find out *who* should have access to the published content. Content providers never should have to decide *how* access control is enforced. On the other side, content consumers will probably click on a URL to visit the content provider's site. This should automatically cause the content consumer's Web client to use the necessary credentials to authenticate itself to the content provider's Web site, without any further user interaction. In Section 4.1 we will elaborate on the usability goals for secure content publishing.

2.2. Trust Management Systems

Trust Management Systems are a great way to express access control policies, and to enforce access control according to those policies. In Keynote [4] one can express arbitrary conditions for access. For example, one could say

1 `http://www.groove.net/`

that access to a certain resource is only allowed if the moon is full and the accessor is orange. It would be up to some other part of the system to determine the phase of the moon and the color of the accessor, but once those are fed to Keynote, it can decide whether to grant access. Keynote has some built-in capabilities to handle cryptographic keys, but it is not very comprehensive when it comes to delegation certificates or role-based access control. Binder [6] is a language that allows such statements to be made. In [3], Bauer *et al.* introduce an even more general access control language – they simply use high-order logic to express access control policies[2]. The idea to use logic-based languages to express policies was originally introduced in [1].

As much power as these trust management systems give the content provider to specify who is allowed to access what under which conditions, they all suffer from being too technical for the non-expert content provider. They may be justified as the enforcement mechanisms that runs "under the hood", but if users can't be bothered to remember a password [2], then we can hardly expect them to write down a Binder program to specify their access control policy.

Even the old-style, inflexible but supposedly easy-to-use, identity certificates provided by X.509 [11] prove to be too much of a hassle. To install a new certificate, for example, users may have to go through a number of dialog boxes, making decisions about certificate stores and key fingerprints (see Figure 3 later in the paper).

We see that in the past, a lot of effort in the area of trust management systems has gone into making them more flexible. Little concern has been given to the fact that *non-expert* users might want to control access to their content.

3. Controlling Access to Content – The State of the Art

In this section, we will give anecdotal evidence as to the state of the art of access control in publishing systems for the World Wide Web.

.MAC Apple's .MAC Web hosting service is tightly integrated with some applications available for the Macintosh. For example, in Apple's "iPhoto" digital photo management application, it requires just little more than pressing of a button to export a photo collection to the .MAC hosting service. There, the photos can be accessed from any Web client. If a user wants to protect access to some of her online photo collections, she can "password-protect" them. This involves picking a password, telling the .MAC service that password, and somehow distributing it to a set of authorized clients. This is both insecure and not very user-friendly. First, the password is likely to be weak.[3] Second, it is probably communicated insecurely. Third, it is trivial for such a password to be passed on to people the original content publisher did not want to grant access to.

Web Servers Web servers such as Apache, Tomcat, or IIS offer sophisticated access control mechanisms. Apache and Tomcat give examples of a clean separation between *authentication* and *authorization*. On one hand, site administrators can specify how users are authenticated, *e.g.,* through passwords or through client authentication using X.509 certificates. On the other hand, they can specify which of those users are authorized to access which resource on the system. While this separation makes a lot of sense, it does add complication to the setup process. None of these servers have tools to provide users with the necessary credentials (*e.g.,* passwords or certificates) to authenticate to the server.

IIS is somewhat of an exception, since it provides a mode in which authentication is "integrated" with a Windows Domain. All the site administrator needs to do is check the corresponding box to specify integrated Windows authentication. Then, authorization is delegated to the access control lists protecting resources on the file system (*i.e.,* a user can access a resource served out by a Web server if and only if she can also access that resource directly through the file system). Now, specifying authorization for Web access is as easy as right-clicking on a folder, selecting the "security" tab, and adding users to the access control list. The disadvantage of this system is that it only works within a Windows Domain (because the users in file access control lists have to be known to the system). Furthermore, it relies on passwords, which can be notoriously weak. If an IIS site administrator wants to switch to certificate-based authentication, she is back in the same boat as the Apache administrators – first, she needs to map certificates to known system users. Then, she needs to specify which users can access which resources. Again, this doesn't even address the problem of generating and distributing certificates to users.

FrontPage Tools such as FrontPage make it easy to create Web-servable content, and to export that content to Web servers. They provide easy-to-use shortcuts to create site directories, change the layout of all pages on a site, and more. As far as FrontPage is concerned, there is only one way to secure access to published content. The authentication method are passwords, and the authorization is provided by an access control list the content provider has to assemble. The good news here is that the content provider doesn't have to specify the passwords of the users – the server will use their Windows login passwords. The bad news is that only users already known to the local Windows Domain can be specified in the access control list.

2 This has the drawback that the content provider can no longer decide whether or not to grant access. Clients have to provide proofs, which the server then checks.

3 The target audience for iPhoto are hobby photographers, not IT professionals with security training.

To be fair, not all the systems compared in this section strive to be "user-friendly" content publishers for the small-time non-expert content provider. But those that do (.MAC and FrontPage) share with those that don't the property that content providers need to think about both authentication *and* authorization. Ideally, a content provider would only specify authorization information, and the system would take care of the authentication itself. The Windows-based systems allow this, but at the cost of restricting access to existing users of a Windows Domain, and also only for password-based authentication.

4. ESCAPE – Usable Security for Small Content Providers

In this section, we describe ESCAPE, an (e)asy and (s)ecure (c)ontent (a)uthorization and (p)ublishing (e)ngine. ESCAPE can be used by non-expert users to quickly share content through a Web server, and to specify access control for that content.

4.1. Goals of Usability and Security

Content providers usually go through a create-publish-announce cycle with their content: First, the content gets created (*e.g.*, a hobby photographer takes pictures). Then, the content is published somewhere (*e.g.*, the photographs are copied to a Web hosting service). Finally, the content provider will announce that her content is online (*e.g.*, send an email with the URL to friends and family). We can assume that the last step also adequately describes the content provider's intention in terms of access control for protected content. For example, if Alice publishes some content and then sends Bob, and only Bob, an email about this content, we will assume that no other than Bob is supposed to have access to that content.[4]

The goal, in terms of usability, is that for the content provider the create-publish-announce cycle for publishing protected content in a secure system should be identical to publishing unprotected content in an insecure system. In particular, the content provider should not be concerned with *authentication* mechanisms. Moreover, we believe that *authorization* information can be deduced from the content provider's actions, for example who gets the announcement and who doesn't. This follows the principle of *implicit security* outlined in Section 2.1.

On the clients' side, consuming protected content should also be identical to consuming unprotected content. There should be no remembering or typing of passwords, or complex management of certificates. While we can reach our usability goal for content providers, we will see that we are somewhat short of reaching this goal for content consumers – in part as a trade-off for better security. See Sections 4.2 and 4.3 for more details.

We know that a secure communication between content provider and content consumer is not possible without some a-priori shared trust information (*e.g.*, a shared secret or password, or public key). Therefore, our usability goals necessarily prohibit an unconditionally secure solution. Instead, we strive for a level of security similar to that provided by SSH. With SSH, the first time a client connects to a server, a man in the middle could hijack the connection and intercept all traffic from then on[5]. But given that the presence of a malicious man-in-the-middle during the first handshake is unlikely, the usability gained outweighs the security lost. In ESCAPE, we strive for a similar level of security – we accept a one-time setup that could potentially be subverted in exchange for a little sacrifice in security.

One way to set up such a system would be to use *capabilities*. For example, the URLs sent out to content consumers could include some hard-to-guess string, which would have to be presented as part of any request to access the content. This system could be made indistinguishable from one for unprotected content, but it has serious security issues: The capabilities can be intercepted as they are sent to the content consumers, and they can be trivially shared with other, unauthorized, individuals (*e.g.*, simply by forwarding the email announcement). We will present a system in which users use a private key to authenticate themselves to the content publishing server. This is a key they are unlikely to share with others, since it allows complete impersonation, not just access to certain content. Furthermore, in our system, no sensitive information is ever exchanged in the clear. We achieve this with off-the-shelf client software such as email readers and Internet browsers.

4.2. System Design Overview

The core of our system is the ESCAPE server. The ESCAPE server is a Web server that serves out content trough HTTPS. In our preliminary prototype implementation (see Section 4.3), the ESCAPE server only serves content that is available locally, and pre-formatted in HTML, but one could easily imagine a version of the server that accepts content upload, and auto-formats content that is not already pre-formatted (much like the .MAC content publishing service accepts a list of photos, and presents them through a polished Web site).

4 In contrast to this *protected* content, there is also *unprotected* content that is accessible by anyone regardless of who got the announcement.

5 That is, unless users actually compare hashes displayed by SSH over a secondary, secure channel.

The ESCAPE server has a key pair it uses to authenticate itself to clients, and to issue certificates for clients. Its public key can either be self-certified, or (at great expense) can be certified by a well-known certification authority. The ESCAPE server keeps an access control list for each directory that it serves out. Each access control list comprises the public keys of those clients allowed to access the directory in question.

To publish content, all the content provider has to do is put it on her computer (or, in the case where the ESCAPE server is running remotely, upload it to the ESCAPE server). She then uses the ESCAPE server to send out an email announcement about the newly published content: Using the GUI provided by ESCAPE, she navigates to the newly created content directory, picks a list of names from her address book, and presses a "Send Announcements" button. The ESCAPE server now does two things:

1. It adds pointers to every selected email recipient's address book entry to the access control list for the newly created content. The address book entry for an email recipient may or may not already contain his or her ESCAPE public key.
2. It sends out an email message to every selected email recipient, informing them about the availability of the newly created content. The email message will contain a URL that recipients can click on to access the content. The URL for a recipient whose email address is `bob@ibm.com` may look something like this: `https://alicescomputer.pacbell.com/holidayphotos?email=bob%40ibm.com`. It might be accompanied by a message inviting Bob to access Alice's newly published content.

Upon receipt of such an email, the recipient can click on the link in the message, and will be taken to the ESCAPE server of the content provider. If the recipient already has an ESCAPE certificate, it will be used to authenticate the client. The public key in the certificate will be used to verify authorization (by a simple lookup in the access control list for the URL in question). If the recipient does not have an ESCAPE certificate, upon visiting the content provider's ESCAPE server, he will be taken through a one-time setup procedure that will install an ESCAPE certificate with the recipient's Web browser. This certificate is issued by the ESCAPE server.[6] Note that this certificate doesn't actually certify any attributes of the client (such as name or email address). It is an "empty" certificate, containing only the signed public key of the client. The association between the public key and client identity is not made in the certificate, but rather inside the address book on the server. (This is because, as we will see later, any client can ask to get a certificate for any identity it wishes to assume, and we need to have a mechanism to quickly revoke wrongly issued certificates. As we will also see later, the decision to let clients acquire arbitrary certificates was in turn dictated by usability considerations.) The setup step is not necessary upon subsequent visits to the URL received in the email announcement, or any other URL on the provider's ESCAPE server. The client is directly served the requested content (if it's listed in the corresponding access control list).

Let us summarize the steps necessary for content providers and consumers to protect, and to access protected, content. The content provider needs to pick a list of users from her address book, which will receive a message about newly published content. In all likelihood, she would have done the same step in an insecure system as well. On the consumer's end, recipients of an email message can simply click on a single URL provided, and will have access to the content. If this is the first time they visit a URL on that particular provider's ESCAPE server, they will be taken through a quick online setup process, which is similar to what happens when people use SSH to log onto a server for the first time. Note that since we are using client certificates, some Web browsers may ask users to provide a password to unlock their private key whenever they visit an ESCAPE server. This is an unfortunate situation, which we will discuss in more detail in Section 5.

4.3. Implementation

It is not very hard to implement the system outlined in the previous section if one were to provide new email and Web clients, perhaps clients that could parse specially formatted email messages and automatically install certificates embedded in such messages. The trick is to implement it with existing client software, and in such a way that requires the least amount of user intervention, dialogs that need attention, *etc.* We will now present an implementation that works with the Outlook address book on the server side, and with Internet Explorer on Microsoft Windows on the content consumer side. The content consumer can use any email client she wishes.

The choice for Outlook on the server side was driven by its wide availability and easy-to-use scripting facilities. It's actually not the best choice in terms of usability for the content provider (see Section 5 for more details). The choice for IE on Windows was made because this appears to be the most user-friendly option for the client side (again, see Section 5 for more details, and a discussion of alternatives).

We implemented a prototype ESCAPE server in Java.

6 The sole purpose of appending the identity of the recipient to the URL sent to him is to be able to store the recipient's public key with the correct address book entry in the content provider's address book. Instead of an email address, any other unambiguous moniker for address book entries can be used.

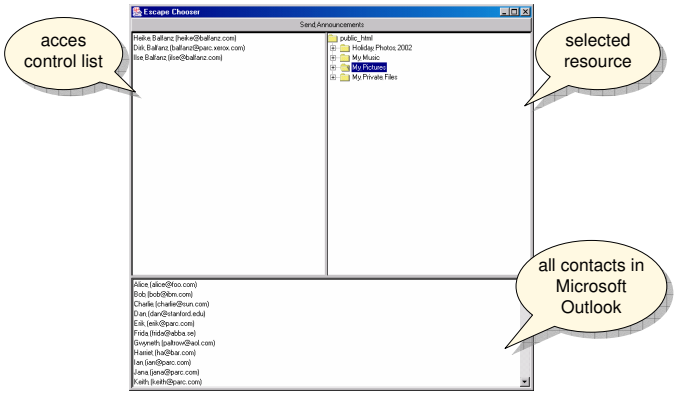

Figure 1. User interface of our prototype ESCAPE server

We used bridge2java [12] to access the Microsoft Outlook COM API from Java. We used the open-source PureTLS TLS implementation [5].

Figure 1 shows the user interface a content provider would see. The upper right pane shows the file system exported by the ESCAPE server. The user can browse to the directory that contains the new content, and select from the list of all known contacts (shown in the lower half of the window) those that should receive an announcement about the new content. Those chosen are simply dragged into the ACL (upper left) pane. Figure 1 shows that three contacts were selected to have access to the "My Pictures" folder. Pressing the "Send Announcements" button results in an email sent to those three users, containing an individualized URL that points back to the "My Pictures" folder, as explained in Section 4.2.

Figure 2 shows what happens when one of those recipients connects back to the ESCAPE servers. Let's assume that they have never connected to this ESCAPE server before, and therefore do not have an ESCAPE certificate for this server. When the client (remember that we're using Internet Explorer) connects, the server immediately starts an SSL handshake (without requiring client authentication). If the server uses a self-signed certificate, the user will see a dialog box that informs her that the server's certificate was issued by someone the user "has not chosen to trust". The user has to press the "continue" button, which will cause the SSL handshake to be finished, and the client to send its HTTP GET request. The server parses the GET request, decides that it is a request for an actual resource (as opposed to the posting of a certificate request), and proceeds with a renegotiation of the SSL connection, this time requiring client authentication. Since the client doesn't have a certificate issued by the ESCAPE server, it sends an empty certificate chain back to the server.[7] (We changed the PureTLS implementation to accept an empty certificate chain without raising an exception.) Since the client has not sent any certificates, it is not served the page it requested, instead it is served a page that informs the user that a one-time setup is needed. Upon pressing a button on that page, Internet Explorer will generate a keypair, and send a certificate request to the ESCAPE server. To the server, this looks like a new client connection. This time, however, the HTTP request turns out to be the posting of the certificate request. The server receives the request, and immediately issues a certificate. The client's public key is stored in the Outlook address book entry for the contact who's email address appeared in the original request URL (which is passed from page to page as HTML form data). The page served out to the client includes the client's certificate chain, and a message to the user that the setup is complete. It includes a link to the original URL the user can click. The certificate chain is automatically installed into the client's certificate store. This, however, results in two more dialog boxes the user has to deal with: First, IE informs the user that a "web site" is trying to install a certificate on her machine. After approving this, she is also informed that a new "trusted root" certificate is about to be installed. This is the self-signed certificate that the ESCAPE server sent along with the client certificate. Approving this install will eliminate warnings in the future. Note that if the content provider acquires a CA certificate from a well-known certification authority, this last dialog box would be eliminated.

Finally, when the user revisits the original URL (by clicking on the link on the "setup complete" page, or by clicking on the link she received in the email), the ESCAPE server will serve the actual content: The HTTP request will be parsed as a request for content; the SSL renegotiation will result in the client's sending of a verifiable certificate chain; the server will now check if the public key presented by the client is part of the access control list specified by the content provider; if so, it will serve the requested page.

Here are some observations about our implementation:

USER OVERHEAD Let us first look at the overhead imposed on content provider and consumer by this secure publishing engine. The content provider, in fact, may not even be aware of the fact that her content is protected by an access control mechanism. All she needs to do is use our provided tool to send out the announcement about newly created content. On the content consumer side, there are four dialog boxes that need to be dealt with during the first visit to the content provider's site. This can be reduced to two if the content provider chooses to purchase a CA certificate from a certification authority that's already known to Internet Explorer.

7 That is, if they are using TLS. See Section 5 for a discussion of the subtle differences between SSL and TLS.

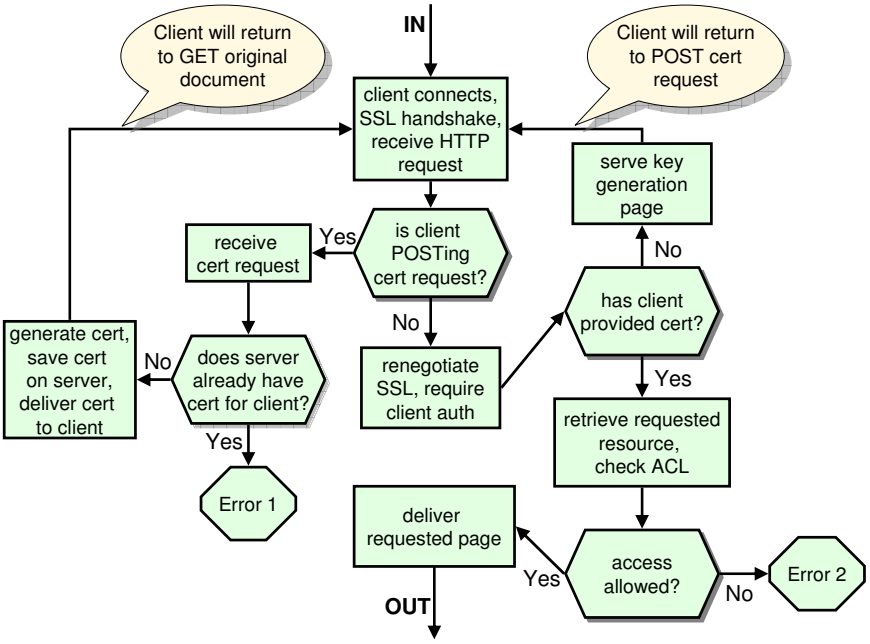

Figure 2. Diagram of internal workings of ESCAPE server

Subsequent visits do not require any special action on behalf of the content consumer at all[8], even though her private key is used to authenticate her to the ESCAPE server. This is because the keypair generation scripts on the "setup" page specify that the private key be stored without any additional protection. In this case, Windows' Data Protection system [10] encrypts the key using the user's login secret, and makes it readily available to processes owned by that user.

We believe that this is the minimum number of user interventions needed for setting up a certificate-based access control system with Internet Explorer. It would be interesting to see whether other browsers allow for a smaller number of dialog boxes, and what the security implications would be.

It turns out that there is one more inconvenience the content consumer has to endure – upon completion of the one-time setup, she has received all necessary certificates and should be able to revisit the original URL. Internet Explorer, however, caches the information that no certificates are available for client authentication (even though they now are), and the connection will fail (it will actually trigger "Error 1" discussed in more detail below). The content consumer has to restart Internet Explorer for the connection to start working.

8 This assumes that the "Don't prompt for client certificate selection when no certificates or only one certificate exists"-setting in Internet Explorer is set to "Enable".

AUTHORIZATION AND REVOCATION Figure 2 singles out two error conditions, marked as "Error 1" and "Error 2". While the system checks for other error conditions, these are especially interesting. Error 2 is simply the error raised when a client tries to access a resource it is not authorized to view (but it has a valid ESCAPE certificate). Error 1 has to do with revocation. As mentioned before, our system cannot be completely secure. For example, an announcement email sent to a recipient that is not set up with our server could be intercepted by a malicious man-in-the-middle. He could access the ESCAPE server before the legitimate user could, thus essentially assuming the legitimate user's identity (remember that we issues certificates immediately online to whoever asks for them first). When, however, the legitimate user later tries to access the ESCAPE server, she will trigger Error 1. The page served out under that condition says that legitimate users should contact the content provider. If and when the legitimate user does so, the content provider simply removes the public key for that user from her Outlook database (it's a Base64 string stored in "User Field 4" of the user's Outlook entry, which can simply be removed). Now, when the legitimate user visits the ESCAPE server again, *she* will be taken through the setup process, and *her* public key will be associated with her Outlook entry. This revokes the man-in-the-middle that illegitimately associated *his* public key with the legitimate user's Outlook entry.

The same mechanism can be used to revoke users permanently, for whatever reason.

4.4. ESCAPE's Security

We explained above how we carefully designed the system to minimize inconveniences for both content providers and content consumers. Let us now look at the level of security we get from our system.

LEAKING OF CONTENT Legitimate users could try and forward credentials that allow them to access content to illegitimate users. As we mentioned above, this would be trivial in a purely capability-based system, where URLs include a hard-to-guess string that would have to be present to access content. In our system each user has one private key per ESCAPE server, which is the credential that enables her to access *any* content on that server. Therefore, we feel that there is some disincentive to share this key with other people. There is never complete protection from content leakage. Authorized users can always download content they have access to, destroy any watermarks embedded in it, and then forward it to unauthorized users. We feel that the level of protection given by public key certificates, combined with the disincentive to share private keys, is adequate for the kind of content we would like to protect (*i.e.,* that provided by small-time publishers for a small set of consumers).

PROTECTION OF KEYS As mentioned before, clients' private keys are protected under the Microsoft Data Protection system [10]. This means that they are encrypted under a key derived from the user's login secret, and then stored on the file system. To use her private key, the user simply has to be logged on to the system. Other users that are logged on to the system, or users that gained access to the system without logging in (*e.g.,* by compromising a server process) won't be able to decrypt the private keys.[9] An attacker that can impersonate a legitimate user to her own computer, however, can easily decrypt those keys. Again, we feel that this provides adequate protection for the kind of content we are trying to protect, and offers desirable usability properties (no need for additional key protection passwords).

CONTENT PROTECTION The ESCAPE server only accepts incoming SSL connections. If the client does not present a certificate, it is given one, and the client's public key is associated, on the server's side, with an Outlook address book entry that's essentially of the client's choosing. We explained before how we can detect if a malicious client associates its public key with the wrong Outlook contact entry. If the client *does* have a certificate, normal SSL protection applies. We note again that a malicious user could gain access by obtaining a certificate before the legitimate user can, but that this access can easily be revoked.

9 At least this is true on Windows XP. On Windows 2000, administrators seem to be able to decrypt items encrypted under the Data Protection API.

5. Hurdles for Deployment

In this section, we will look at some of the remaining usability issues in ESCAPE, as well as shed some light on why other possible system designs seem less suitable in terms of usability. This will reveal that designing an easy-to-use secure system with off-the-shelf components is harder than one would expect.

CERTIFICATE DELIVERY One of the problems, in terms of security, with the system presented above is the way it delivers certificates. A more secure way would be to email certificates to the email address presented at certificate request time by the client's Web browser (as part of the URL). This would raise the bar significantly for an attacker who wanted to impersonate other principals to an ESCAPE server. We believe, however, that this would significantly affect the usability of the system. First of all, there would be no immediate access to the content when a client first receives the email announcing the content's existence. Upon connecting to the server, the client would have to wait for a second email delivering the certificate. Furthermore, the installation of a certificate that's sent per email is considerably more involved than installation of a certificate from a Web page. Scripting is switched off by default in most email clients, so one couldn't send an HTML page similar to the one presented to Internet Explorer for certificate pickup. Instead, one would have to send the certificate as an attachment in the email. Opening the attachment on the client side results in the dialog boxes shown in Figure 3. Some of the choices the user has to make are not obvious, and we believe that many non-expert users would fail to install the certificate successfully. Given that the window of opportunity for an attacker in our scheme lasts only until the legitimate user first contacts the ESCAPE server, we feel that the usability gained by our system offsets the security lost.

OUTLOOK SCRIPTING It turns out that simple things like accessing the Outlook address book already raise security issues that the user is asked to deal with. When starting the ESCAPE server (see Figure 1), Outlook pops up a dialog warning the user that "some application" is trying to access the email addresses of contacts in the Outlook database. The user has the choice of denying this access, or allowing it for up to 10 minutes. This caution is motivated by email worms, but a legitimately installed application such as the ESCAPE server should not trigger this warning dialog, which is quite confusing (the user is not even told which application is requesting the access). It would be interesting to see how other scriptable contact databases (*e.g.,* the Macintosh address book) handle this kind of situation.

CLIENT AUTHENTICATION We noted above that the ESCAPE server relies on the fact that a successful SSL handshake can be completed with a client that doesn't have a

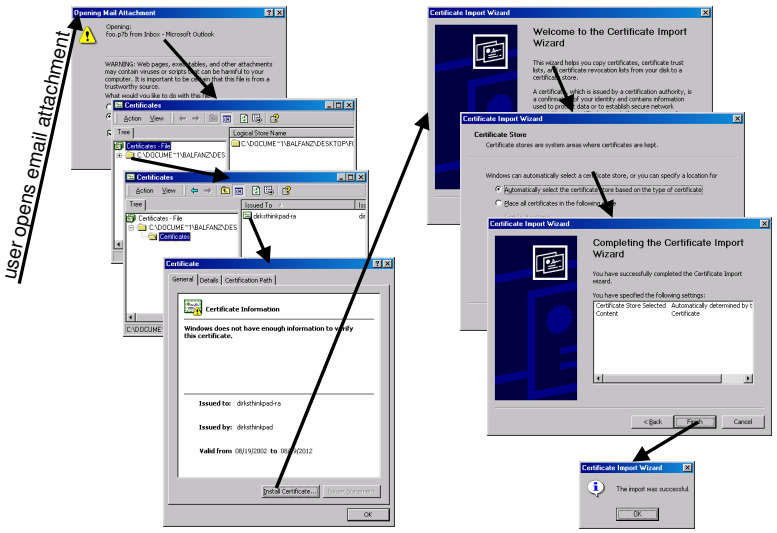

Figure 3. Installing a certificate delivered as an email attachment

certificate, even though client authentication is requested by the server. The SSL master secret doesn't need data from the client certificate, so theoretically, there is no need to abort a handshake if the client cannot present a certificate. Here is what the TLS RFC [7] says about this subject:

> 7.4.6. Client certificate
>
> When this message will be sent: This is the first message the client can send after receiving a server hello done message. This message is only sent if the server requests a certificate. If no suitable certificate is available, the client should send a certificate message containing no certificates. If client authentication is required by the server for the handshake to continue, it may respond with a fatal handshake failure alert. [...]

(The "it" in the last sentence refers to the server, not the client.) SSL v3.0, on the other hand, specified that a client, "if no suitable certificate is available," "should send a no_certificate alert instead" (see [9]). It turns out that Internet Explorer by default uses SSL v3.0, and implements it according to spec. Even though the no_certificate alert is meant to be only a warning, the PureTLS implementation we use aborts the handshake on receipt of such an alert. Once IE is set to set to use TLS, it also implements it according to spec, and the handshakes succeed in accordance with Figure 2. Mozilla appears to use TLS by default, and implements it correctly (*i.e.*, our handshakes succeed). Opera uses TLS by default, but appears to implement it incorrectly – we never successfully finished a handshake with Opera.

OTHER PLATFORMS We mentioned a few times above that the Data Protection System on Microsoft Windows removes the need for users to type passwords when Internet Explorer unlocks their private keys. Other browsers do not use this feature on Windows. This means that users have to type passwords every time they want to access an ESCAPE server. Likewise, other operating systems such as Linux do not provide a data protection service, so browsers *have* to use passwords to store private keys securely.

TRANSFERRING CERTIFICATES In our system, there can only be one public key per Outlook contact entry. This prevents illegitimate users from acquiring certificates for an identity once the legitimate user has received her certificate (see discussion of "Error 1" in Section 4.3). Unfortunately, the same mechanism prevents legitimate users from receiving a second certificate, say, for a second computer they own. Instead, they have to copy their keypair (and certificate) to each machine they want to use. Exporting and importing keypairs is an involved process. We are currently still investigating whether there can be a system that has similar usability and security properties as ESCAPE, yet allows users to easily set up multiple credentials on multiple machines they own.

MANAGING ACCESS CONTROL LISTS It is very easy to remove clients from an access control list. Using the ESCAPE graphical user interface, the content provider simply has to navigate to the published folder in question, and remove unwanted clients from the access control list. While this is certainly an easy task, it doesn't follow our mantra of "implicit security", *i.e.*, the only reason a content provider would want to do this is for security purposes.

It is unclear whether each and every security mechanism can be hidden underneath a non-security-related action, nor is it clear that this is desirable. After all, hiding security is

not our ultimate goal – encouraging people to *use* security is, and hiding it in appropriate places is one way to achieve this goal. Where security cannot be hidden (as seems to be the case here), exposing it in a user-friendly way is the right thing to do.

6. Conclusions

In this paper, we set out to design a system that would allow controlling access to content published on the World Wide Web. While this is a well-studied problem, our somewhat unusual goal was to achieve a high level of usability with off-the-shelf client software, while at the same time providing a reasonably secure system. Ideally, both content providers and content consumers would not have to do anything "extra" in a secure system (compared to an insecure one). Assuming a create-publish-announce cycle, we achieved this goal on the content provider's side. On the content consumer's side, we don't incur any extra cost except for a one-time setup process.

A truly user-friendly system relies on certain features found in off-the-shelf applications, some less surprising than others. For example, not only is the Data Protection system on Microsoft Windows (and the way IE uses it) useful for our goal, it also turns out that the correct implementation of a seemingly insignificant feature in TLS 1.0 is required to minimize the overhead imposed on users.

We implemented a content publishing server (ESCAPE) in Java that enables content consumers to access published content with common off-the-shelf client applications such as Internet Explorer. We strove to minimize the overhead to users (both on the publisher's and consumer's sides), bringing it as close as possible to that of a publishing system that doesn't have any built-in access control. Interesting future work would include a survey of client software not considered in this paper, and perhaps find even more opportunities to remove overhead imposed by security mechanisms.

ESCAPE is a publishing engine for small-time publishers because it doesn't scale to large numbers of content consumers. We don't believe this is a big disadvantage, since the content publishers that address small audiences will benefit most likely from usable security (*i.e.*, they are more likely to be non-expert users who can't afford to hire security IT personnel).

We encourage the community to design secure systems from ground up for usability, since unusable security mechanisms will not, as the adjective suggests, be used at all.

While our system addresses a very specific need (*i.e.*, it allows small-time publishers to protect access to their published online content), we believe that it can serve as a case study in usable security design. The design of our Easy and Secure Content Authorization and Publishing Engine also gives some insight into the caveats encountered when designing secure systems for usability, and into the kinds of trade-offs between usability and security that are likely to be encountered when building usable secure systems.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. D. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[2] A. Adams and M. A. Sasse. Users are not the enemy: Why users compromise computer computer security mechanisms and how to take remedial measures. *Communications of the ACM*, 42:40–46, December 1999.

[3] L. Bauer, M. A. Schneider, and E. W. Felten. A general and flexible access-control system for the web. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, CA, August 2002.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. *The KeyNote Trust-Management System Version 2*. IETF - Network Working Group, The Internet Society, September 1999. RFC 2704.

[5] Claymore Systems. PureTLS.

[6] J. DeTreville. Binder, a logic-based security language. In *2002 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.

[7] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. IETF - Network Working Group, The Internet Society, January 1999. RFC 2246.

[8] W. K. Edwards, M. W. Newman, J. Z. Sedivy, T. F. Smith, D. Balfanz, D. K. Smetters, H. C. Wong, and S. Izadi. Using speakeasy for ad hoc peer-to-peer collaboration. In *Proceedings of ACM 2002 Conference on Computer Supported Cooperative Work (CSCW 2002)*, New Orleans, LA, November 2002.

[9] A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL Protocol Version 3.0*. IETF - Transport Layer Security Working Group, The Internet Society, November 1996. Internet Draft (work in progress).

[10] W. Griffin, M. Heyman, D. Balenson, and D. Carman. Microsoft data protection. MSDN Online, October 2001.

[11] R. Housley, W. Ford, W. Polk, and D. Solo. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. IETF - Network Working Group, The Internet Society, January 1999. RFC 2459.

[12] IBM. bridge2java. `http://www.alphaworks.ibm.com/tech/bridge2java/`.

[13] D. K. Smetters and R. E. Grinter. Moving from the design of usable security technologies to the design of useful secure applications. In *New Security Paradigms Workshop '02*. ACM, 2002.

[14] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, Washington, DC, August 1999.