Impostor: A Single Sign-On System for Use from Untrusted Devices.

Andreas Pashalidis and Chris J. Mitchell
Information Security Group
Royal Holloway, University of London
Email: {A.Pashalidis,C.Mitchell}@rhul.ac.uk

***Abstract*— At present, network users have to manage a set of authentication credentials (usually a username/password pair) for every service with which they are registered. Single Sign-On (SSO) has been proposed as a solution to the usability, security and management implications of this situation. Under SSO, users need to manage only one set of authentication credentials in order to log into the services they subsequently use. This paper presents the design of an SSO system that is based on a trusted proxy, and that is suitable for use from an untrusted network access device. Unlike existing proxy-based SSO schemes, which require an infrastructure to be in place between the proxy and the service providers, the one presented here does not. An open-source implementation of the scheme, called 'Impostor', is also described. The prototype is implemented as an HTTP proxy, resulting in a system that works with common web browsers.**

I. Introduction

Single Sign-On (SSO) has been proposed as a solution to the issues that arise from the fact that network users need to manage a set of authentication credentials (usually a username/password pair) for every service with which they are registered. Under SSO, users have to manage only *one* set of authentication credentials in order to log into all SSO-enabled services. Several architectures for SSO have been developed, each with different properties and underlying infrastructures [1]. While some SSO systems are local, others rely on a trusted third party often called the 'SSO proxy'. In a proxy-based architecture the proxy typically authenticates users once at the beginning of a session (the 'primary' authentication), and automatically logs them into the services they subsequently use.

One advantage of proxy-based SSO systems is that they have the potential to enable user authentication to a variety of remote service providers from untrusted network access devices in a way that does not compromise the secrecy of long-term user authentication credentials. This applies even when the authentication methods used by remote service providers would normally not offer any protection to user credentials, e.g. the use of passwords. This paper presents the design of an SSO scheme with this feature. The paper is organised as follows. The next section defines what we mean by an 'untrusted' network access device in the context of this paper, and section III presents the objectives, architecture and properties of the scheme at a design level. Section IV presents 'Impostor', a concrete prototype implementation of the scheme in the form of an HTTP proxy. Section V discusses additional technical issues and sections VI and VII give an overview of related work and conclude the paper.

II. What is an untrusted network access device?

A network access device can compromise security in a number of ways; it can log and disclose all its internal state (including sent and received messages, cryptographic keys and passwords), it can spoof (both local and remote) interfaces, and it can simply deny service. While users do not expect a trustworthy device to engage in such malicious behaviour, the meaning of a network access device being 'untrusted' has to be precisely defined and assumptions about it made explicit in order to provide a clear understanding of the scheme's objectives.

In order to clarify matters, consider the following scenario, around which the scheme is built. Suppose a user needs to access some network service, say an e-mail account, from a network access device that he does not particularly trust, for example a public terminal at an airport or an Internet café. The user is typically willing to trust the terminal to act as the communication endpoint and input device during this particular online session, but for no longer. Now suppose that the e-mail provider authenticates users using long-term credentials, such as username/password pairs. Providing these long-term credentials to the public terminal, however, constitutes a serious risk since knowledge of these credentials enables the terminal (and its operator) to impersonate the user to the service provider, even after the session has ended. The primary purpose of the scheme described here is to avoid this risk.

Based on the above scenario, a network access device is deemed 'untrusted' if the following conditions hold.

- There exists a temporary trust relationship between the user and the network access device: the former trusts the latter only for the time period of a particular session, but for no longer. In particular, during that session, the device is trusted
  - not to spoof local and remote user interfaces, and
  - not to hijack communication sessions with individual services.
- The device is not trusted to protect any long-term secrets, i.e. secrets that remain valid *after* a particular online session has completed, that may be entered or otherwise made available to it.

Issues that arise if these conditions are relaxed are briefly discussed in section V-A. It is worth noting that the use of secure channels, such as those provided by the Secure Socket Layer (SSL) or the Transport Layer Security (TLS) protocols [2], does not help protect sensitive data from the untrusted device, although a secure channel would offer protection for exchanged data against third party interception and/or manipulation.

III. Description of the SSO scheme

This section describes the objectives, the architecture and certain key properties of the SSO scheme at a design level.

A. Objectives

Methods exist that enable users to authenticate themselves to remote entities without having to disclose long-term secrets to the access device they are using. Examples include challenge/response protocols and one-time passwords, see, for example, [3], and products based on such ideas are widely available. These authentication mechanisms are suitable for use from an untrusted device, but, unfortunately, are rarely used by providers of public network services. The majority of services authenticate users based on a username and password.

Based on these observations, the primary three objectives of the SSO scheme are as follows.

1) The scheme should enable users to authenticate themselves from an untrusted network access device (as defined in section II) to remote service providers whose login mechanisms involve long-term secrets (such as passwords), without, however, disclosing these long-term secrets to the access device.
2) The scheme should require users to manage only one set of authentication credentials, thereby providing SSO.
3) The scheme should be transparent to network service providers; they should not even need to be aware that it is in place.

B. Architecture

The architecture of the scheme is based on a trusted SSO proxy that keeps a copy of the user's long-term authentication credentials in a suitably protected credential database. All network traffic between the untrusted device and the remote service providers is physically routed though that proxy. The information flow that occurs in the event of a user trying to log into a network service is depicted in Figure 1. As shown in that figure, the user's login request to a remote network service provider is recognised and intercepted by the proxy in step 1. In step 2 the proxy authenticates the user using a suitable one-time authentication mechanism, typically including a challenge issued by the proxy (step 2.1) and a response from the user (step 2.2). Assuming successful user authentication, the proxy executes the authentication mechanism used by the network service on behalf of the user in step 3. This would typically involve employing the user's long-term secrets that are stored in the credential database. The secrets, however, never reach the untrusted network access device, as step 3 occurs directly between the proxy and the remote service provider. Finally, assuming that step 3 completes successfully, the service is provisioned in step 4 and the first objective listed in section III-A is met.

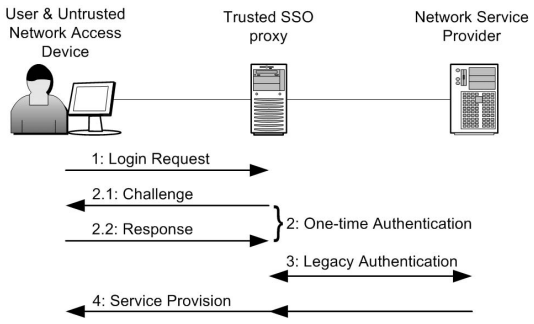

Fig. 1. Architecture

The second objective is met by differentiating between users (in step 2) based on a unique identifier which is decoupled from any particular network service; users have to memorise or otherwise have available only that one identifier and carry out the one-time authentication in order to use the system. Finally, the third objective is met by having the proxy replace the user's network address with its own when forwarding requests to network services. In this way service providers only 'see' the proxy and are therefore not aware of step 2 taking place. Within the various types of SSO scheme, this architecture falls into the category of proxy-based pseudo-SSO schemes [1].

C. Properties

As a proxy-based pseudo-SSO scheme, the system inherits all the properties of that particular class of SSO scheme. In particular, changes in a network service's proprietary authentication interface have to be reflected at the proxy. This means that maintenance costs may be relatively high, especially in a dynamically changing environment. The fact, however, that the proxy is transparent to service providers enables its immediate and cost-effective deployment.

As it is the case with any proxy-based SSO scheme, the proxy constitutes a single point of failure and has to be protected against service denial attacks. Furthermore, its credential database has to be properly protected against illegal access, as it contains long-term secrets that allow impersonation of users to all SSO-enabled network services. However, this is not as severe an assumption as it might seem; in practise the proxy might be implemented on the user's own 'home PC', which would already typically be trusted to store such secrets (e.g. in the form of cookies), and is not likely to be a major target for service denial attacks. In the corporate setting, on the other hand, using proxy chaining and load balancing techniques could offer an acceptable level of resistance against service denial attacks. Furthermore, the credential database could be stored in an encrypted form at a centralised server within a protected area of the intranet.

As a consequence of the proxy's need to intercept all traffic between the user and the rest of the network, no end-to-end

secure channels between the untrusted device and any service are allowed; if a secure channel is nevertheless required at the application level, the proxy has to set up two separate secure channels, one with the user and one with the service, such that interception can continue.

IV. The Impostor prototype

This section presents 'Impostor', a proof-of-concept implementation of the SSO architecture described in the previous section. Although the SSO architecture is generic in the sense that it could be used for several network services, such as the File Transfer Protocol [4] or the Simple Mail Transfer Protocol [5], Impostor is realised as an HTTP (web) proxy. Thus, the prototype offers SSO for websites only. It is intended to be started as a continually running process (also known as 'service' or 'dæmon') on a trusted 'SSO server' machine. The implementation was written in Java 1.4 and has been successfully tested with a variety of common web browsers and operating systems, both Unix-based and Windows-based. It is available as an open source project at `http://impostor.sf.net`.

The core of the software consists of an HTTP proxy, augmented with the SSO functionality described in the previous section; every incoming request to the proxy is analysed (step 1 in Figure 1) and, if it is recognised as a login request for a website for which the proxy has been equipped with user authentication credentials, the proxy presents the user with a (customisable) login web-page that contains a freshly generated challenge, and asks the user to provide his/her unique identifier and a response (step 2). If the identifier is valid and the response matches the challenge[1], the user's long-term credentials for the website in question are filled into the initial HTTP request from the user, which is then forwarded to the website (step 3). Otherwise, an error page is returned. If an incoming request is not recognised as a login request the proxy simply forwards it to the website.

If, at any stage, an SSL/TLS channel is required (via the HTTPS scheme), the proxy sets up two separate channels: one with the website and one with the user browser. The certificate the proxy uses to setup the browser-side SSL/TLS connection does not, of course, match the website's certificate, but, fortunately, this does not disrupt the service as browsers typically offer the option to accept 'unknown' certificates via a simple yes/no dialogue[2]. Moreover, users should expect this, as they will certainly know that Impostor is being used, and will also know the web page via which the proxy is operating.

Impostor offers an additional privacy protection service by passing every web page to a method that may filter its content, before sending it back to the browser. In this way it is possible to prevent personal information, such as a user's address or credit card details, ever reaching the untrusted device. It is worth emphasising that this filtering service is not disrupted in the presence of an SSL/TLS channel.

The software is designed in a modular fashion, such that it is easy to extend or replace individual components without affecting the functionality of unrelated components. In Java terminology, the core proxy dæmon classes form a *package* that 'talks', via well-defined APIs (i.e. Java interfaces), to the supporting components (i.e. classes) that are responsible for (1) recognising and 'filling in' HTTP requests for websites, (2) performing the one-time authentication mechanism, (3) managing the credential database, and (4) performing content filtering. In other words, the proxy dæmon is logically decoupled from these four components; it does not directly implement their functionality. Thus, separate *implementations* of these interfaces need to be provided in order to make Impostor work. The interfaces, along with their current implementations, are briefly introduced in the remainder of this section.

A. The RequestRecognizer interface

The RequestRecognizer interface provides access to the methods supporting the identification and processing of user login requests to websites. Every incoming HTTP request is passed to a method (called `init`) of this interface. If the implementation recognises it as a login request to a website (step 1 in Figure 1), the proxy is informed (via the `isRecognized` method) and invokes the SSO protocol, as explained above. The implementation of this interface must also provide a method that 'fills in' user credentials (i.e. usernames and passwords) into the request. The proxy calls this method (`fillInUsernameAndPassword`) after successful one-time user authentication (step 2) in order to perform the service-specific legacy authentication (step 3). If an incoming request is not recognised as a login request the proxy simply forwards it to the website.

The current implementation recognises and 'fills in' login requests for three web-based email providers, namely two well-known public providers and a university service.

B. The ChallengeResponseManager interface

The proxy uses an implementation of the ChallengeResponseManager interface in order to perform the one-time user authentication; the interface provides methods for issuing a new challenge (`getNewChallenge`), determining whether or not a given user identifier is valid (`isValidIdentifier`), and verifying a given response to a previously issued challenge (`verifyResponse`).

The challenge/response mechanism of the current implementation requires users to share a passphrase (at least eight characters) with the proxy server. The challenge consists of the proxy asking the user to provide three randomly chosen characters from his passphrase (e.g. the second, fifth and last). If the user fails to provide the correct response, the server keeps asking for the same set of characters until either the correct response is given, or a maximum number (e.g. five) of failures has been recorded. In the latter case, the user's account at the proxy is disabled. This challenge/response mechanism

[1]The proxy rejects replays of previously accepted responses and responses that are received after a specified timeout period.

[2]The situation is fortunate in the context of an untrusted network access device and the SSO scheme of this paper; in other scenarios it is certainly a weakness [6].

could be, in principle, easily replaced with any mechanism such as one of those mentioned in section III-A.

C. The UserManager interface

The UserManager interface defines the API used by the proxy dæmon to interact with the credential database management component. The implementation must provide the functionality of mapping proxy users to their website-specific authentication credentials. In particular, methods must be provided that retrieve the website-specific usernames (`getUsernameForIdentifier`) and passwords (`getPasswordForIdentifier`) for any valid proxy user identifier/SSO-enabled website combination.

The current implementation uses a simple, text-based credential database, but it could be replaced by arbitrarily complex installations.

D. The ContentFilter interface

The ContentFilter interface provides methods used to support the privacy protection service mentioned above. These methods filter HTTP headers (`filterHTTPHeaders`) and web pages (`filterWebPageLine`) before the proxy forwards them to the browser; the implementation looks for any kind of information that should not be sent to the untrusted network access device and substitutes it with a neutral (or empty) string.

The current implementation looks for a set of strings specified by the administrator. If any of these 'sensitive' strings is found, it is replaced by a string that is chosen randomly from a set of substitution strings for the particular sensitive one. This fairly basic (but nevertheless effective) filtering mechanism could be replaced with a more sophisticated content filtering technique.

V. Other issues

A. Relaxing the assumptions

As explained in section II, the untrusted network access device is assumed not to spoof user interfaces or to hijack communication sessions with individual services. This section briefly examines the implications of relaxing these assumptions.

If the untrusted device spoofs the user interface then there is no assurance that users are communicating with the entities they believe they are communicating with, even in the absence of external attacks. This is because authenticating a remote entity requires some kind of interface informing the user whether or not authentication was successful. If this interface is provided by a trusted device, such as a trusted smartcard reader, authentication of services (including the SSO proxy) could be supported in this untrusted environment. The value of such an authentication, however, is undermined in the presence of session hijacking. If the untrusted device hijacks the user's session (either on its own or colluding with another entity) it can effectively abuse the user's authentication status at any service. This renders any authentication mechanism (unilateral or mutual) between user and services useless, even if it results in a key that is supposed to cryptographically protect subsequent communications (as it is the case with SSL/TLS channels).

It is clear that, if the untrusted device spoofs its interfaces or hijacks the communication sessions with individual services, the user's entire online session is compromised. A network access device that engages in such malicious behaviour is not addressed by the SSO scheme described here. It is nevertheless worth noting that, even in the presence of interface spoofing and session hijacking, users of the scheme described here do not have to disclose long-term secrets to the untrusted device; as long as those secrets are not disclosed by other sources, only the current session is compromised.

B. Some technicalities

This section documents some technical issues that arose during the implementation of the SSO proxy.

1) Setting up Impostor: As explained in section IV above, the Impostor prototype is independent of its supporting components. Therefore, 'setting up' the system mainly involves setting up the components responsible for the one-time user authentication mechanism and the credential database management, i.e. populating them with the user's Impostor and website-specific authentication credentials. It is also necessary to generate the asymmetric keypair (and a corresponding public key certificate) that Impostor will use when setting up browser-side SSL/TLS connections. Finally, the Impostor login webpage and error page have to be designed according to some simple guidelines that allow the proxy to dynamically insert challenge values (and other details) at runtime.

2) HTTP Authentication: In [7] the "HTTP Authentication" method is specified that is often used by websites and web proxies to authenticate users. Although there is provision for a challenge/response mechanism with the one-time property (called 'Digest Access Authentication'), it involves users typing their usernames and passwords into the access device. Thus, during user-to-proxy authentication (step 2 in Figure 1), a standard web form is used to acquire the user's identifier and response.

In order to authenticate users to websites (step 3), on the other hand, the SSO proxy can perform the HTTP Authentication protocol as long as the RequestRecognizer component supports it. (In fact, the university web-based email service mentioned in section IV-A uses HTTP Authentication.)

3) The use of cookies: The web proxy does not, by default, interfere with the cookies that individual services may store in the browser. It is possible, however, to filter out cookies using the content filtering facility described above. The proxy itself does not save cookies in the browser.

4) Persistence of connections: The proxy does not currently support persistent connections; every incoming HTTP request is processed independently of others. This has a number of side effects: on the one hand, the user has to re-authenticate every time the proxy is asked to provide long-term credentials; this reduces the exposure to session hijacking. On the other hand, the performance is degraded in certain circumstances.

5) Real world applicability: From a practical point of view, the main drawback of the Impostor prototype is that users are required to configure their browsers to use the proxy, i.e. change the settings of the untrusted device's browser. Some public terminals impose restrictions on this. Another issue may arise with firewalls between the untrusted device and the proxy. Running the proxy on a port that is usually not blocked by firewalls (such as port 80 or 22) may help circumvent that problem.

VI. Related Work

There exists a variety of SSO architectures; a detailed comparison can be found in [1]. The scheme described here falls into the category of a proxy-based pseudo-SSO scheme.

Web-based SSO schemes that are based on a trusted SSO proxy, such as the set of the Liberty Alliance [8] specifications and Microsoft Passport [9], are probably the most closely related schemes to the Impostor prototype. Liberty in particular is neutral to any specific authentication method. This means that, if a suitable mechanism is selected, use from an untrusted network access device can be supported. However, as pointed out in [1], Liberty and Passport are true SSO schemes. This means that explicit relationships between the SSO proxy and the service providers need to be established and supported by a (potentially costly) common security infrastructure (such as a Public Key Infrastructure) that spans the SSO proxy, all participating websites and eventually even the end-users. In contrast, as the scheme described here is transparent to service providers, it does not require explicit relationships or any common security infrastructure between the proxy and service providers; the solution is far more flexible and easily deployed. Furthermore, it can be implemented on a small or large scale; the proxy can be operated by any organisation, including individual users, and not just by organisations with well-established relationships with service providers.

Local SSO schemes, such as Novell's SecureLogin[3], Passlogix' V-GO[4] and Protocom's SecureLogin[5] are SSO solutions that can be deployed transparently to service providers. The fundamental difference between these systems and the prototype described here is that these systems are not suitable for use from an untrusted network access device; the local device necessarily gets access to the user's service-specific authentication credentials.

Another area of related work is that of content filtering proxies. The majority of existing content filtering proxies focus on the protection of minors from inappropriate content and the restriction of employees in a business environment. Although the purpose of the content filtering functionality of the system presented here is different, the concept is similar.

However, one ought to keep in mind that a malicious device can always 'switch off' the proxy without notification.

VII. Conclusion

We presented the design of a proxy-based pseudo-SSO scheme and described a prototype implementation of it. The scheme essentially overlays a single, one-time authentication method over legacy authentication mechanisms of individual service providers, thereby providing SSO. The prototype, which works 'in the real world' as an HTTP proxy in a manner completely transparent to websites, is easily extensible and does not specify any particular one-time authentication method or database component; it may be deployed by individual users or in a corporate setting.

Looking into the future of ubiquitous computing, where personal devices with limited (or no) user interfaces might need to connect to various service providers as users roam, pseudo-SSO schemes such as the one presented in this paper might prove useful; all the limited devices are required to perform is one authentication mechanism with the trusted proxy. The proxy, as a middleware service, then executes predefined authentication mechanisms with each individual service provider, not only circumventing the (costly) need for system-wide agreements and infrastructures, but also transparently adapting to a dynamically changing environment.

Acknowledgment

The first author is supported by the State Scholarship Foundation of Greece.

References

[1] A. Pashalidis and C. J. Mitchell, "A taxonomy of single sign-on systems," in *Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003, Proceedings*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and J. Seberry, Eds., vol. 2727. Springer-Verlag, July 2003, pp. 249–264.

[2] E. Rescorla, *SSL and TLS*. Reading, Massachusetts: Addison-Wesley, 2001.

[3] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.

[4] J. Postel and J. Reynolds, *RFC 959: File Transfer Protocol*, Internet Engineering Task Force, October 1985.

[5] *RFC 2821: Simple Mail Transfer Protocol*, Internet Engineering Task Force, April 2001.

[6] A. Pashalidis, "A cautionary note on automatic proxy configuration," in *IASTED International Conference on Communication, Network, and Information Security, CNIS 2003, New York, USA, December 10-12, 2003, Proceedings*, M. Hamza, Ed. ACTA Press, December 2003, pp. 153–158.

[7] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*, Internet Engineering Task Force, June 1999.

[8] *Liberty ID-FF Architecture Overview DRAFT Version 1.2-03*, Liberty Alliance, April 2003.

[9] *Microsoft .NET Passport Review Guide*, Microsoft, November 2002.

[3] www.novell.com/products/securelogin

[4] www.passlogix.com/sso

[5] www.protocom.cc