Polybius: Secure Web Single-Sign-On for Legacy Applications

Pascal Gienger Marcel Waldvogel
University of Konstanz
Konstanz, Germany
pascal.gienger@uni-konstanz.de, marcel.waldvogel@uni-konstanz.de

Abstract: Web-based interfaces to applications in all domains of university life are surging. Given the diverse demands in and the histories of universities, combined with the rapid IT industry developments, all attempts at a sole all-encompassing platform for single-sign-on (SSO) will remain futile. In this paper, we present an architecture for a meta-SSO, which is able to seamlessly integrate with a wide variety of existing local sign-in and SSO mechanisms. It is therefore an excellent candidate for a university-wide all-purpose SSO system. Among the highlights are: No passwords are ever stored on disk, neither in the browser nor in the gateway; its basics have been implemented in a simple, yet versatile Apache module; and it can help reducing the impact of security problems anywhere in the system. It could even form the basis for secure inter-university collaborations and mutual outsourcing.

1 Introduction

The processes in teaching, learning, research, and administration at a university are manifold. As a result, the applications written for their support date back to various decades, using dozens of different interfaces, programming environments, and authentication/authorization concepts. Despite efforts into identity management, the integration in practice is limited to a subset of applications, as it falls short of user demands and security requirements. Outside a small group of "compatible" applications, there remains a chaos of usernames, passwords, and conventions, combined with many manual and error-prone processes. Even those few applications are sometimes combined by hard-to-use SSO systems, turning ordinary users away in despair.

In this paper, we present a generic framework, *Polybius*,[1] which allows the creation of an SSO mechanism which provides single-log-out and can sit on top of existing SSO or local sign-in systems. It avoids the common pitfalls of storing passwords in cleartext anywhere. We also have implemented the system in a light-weight Apache module, providing an interface to three previously unconnected systems, none of which require any modifications to become part of the SSO infrastructure.

[1]Polybius was a Greek historian who also reported on the secure way the Romans used to distribute their ephemeral passwords `http://en.wikipedia.org/wiki/Password#History_of_passwords` (visited 2011-01-06)

The processes at a university are not only manifold, as already explained, but also tightly interwoven. However, the various applications stemming from different decades, technologies, and manufacturers, rarely provide this integration directly.

For example, in the context of a research project, acquisition, funding planning, people hiring, project planning, e-collaboration, and travel require several distinct applications, requiring different tools and multiple entry of the same information. Also, lectures require time and room scheduling, assigning assistants, group assignments, preparation and publishing of texts, notes and slides, as well as grading and entering grades, which again involve a multitude of systems.

Therefore, important goals of application management at universities include:

1. to integrate these applications from a user's point of view, e.g., portals, common authentication infrastructure, . . . ; and
2. to allow applications to interoperate, i.e., have data generated by one application be used to .

The aim of both types of integration is to improve the efficiency of processes while at the same time allowing modularity to prevent single-vendor lock-ins and therefore provide competitive multi-vendor integration or migrations.

Conventional designs have kept the SSO mechanism out of the actual user↔system data flow as much as possible. However, the time is ripe to rethink traditional design rationale: The secure design of Polybius, advances in computer performance, coupled with ease of redundancy and graceful degradation make Polybius less of a single point of failure than conventional SSOs in many cases. We discuss this in detail in Section 3 and show applications with and without password synchronization in Sections 4 and 5, respectively.

2 Related work

Many different approaches to the SSO problem have been developed, most notably Kerberos [NYHR05], Shibboleth [Sco05], and CAS [Jas].

Kerberos is very widespread, frequently deployed as part of Microsoft Active Directory. Applications do check issued Kerberos tickets which are requested in authentication phase from the Key Distribution Center, KDC ('Domain Controller' in Active Directory parlance). The concept has been slightly modified by Microsoft to be able to send Kerberos tickets through by HTTP to allow their Microsoft Internet Explorer to use the same authentication structure for web applications [JZB06].

In an effort to offer an SSO infrastructure to a broader audience and heterogeneous environments, other schemes did evolve. Shibboleth is widely used in the European academic community whereas CAS is often found in North America. Shibboleth is well known for its cross-domain authentication, which we will not cover here.

All these methods share disadvantages when it comes to legacy web applications: They will not work out of the box. Instead, you will have to rewrite each and every application to make use of these structures. Applications have to be “kerberized” to use Kerberos, “shibboletised” to use Shibboleth, and so on.

First, any such modification requires access to the application source code to modify the login process. Second, it may even require modification of the application logic, as different information is available as part of the login process. Third, it may not even work at all, if the backend insists on password-based authentication, as is common when accessing legacy services such as terminal emulations or non-Web protocols including many IMAP or LDAP servers.

3 Polybius design

To avoid these problems, our approach is to keep the passwords as part of the process. Storing or transmitting passwords is generally frowned upon, as they increase the vulnerability of the system: An attacker may gain access to the password store or to the user’s browser. To avoid attacks on data at rest or data in flight, Polybius employs a form of secret sharing between the browser and the SSO gateway: Neither of them has enough information to get at the password, but on every request, the browser delivers the information which allows the gateway to recover the password for this request; which the gateway forgets immediately after using it for the backend service.

To accomplish this, user credentials (e.g. user name(s) and password) are stored in a session database encrypted using AES in CBC mode, using a 256 bit encryption key and a 128 bit initialization vector (IV). These two values together with a session-id (to reference the appropriate record in the database) form the unique session “key” needed by the SSO server to reconstruct the password when needed. This session key is stored as an HTTP cookie in the user’s browser (Figure 1).

This method leaves the session database useless for an attacker - without the keys he is not able to extract the credentials. When capturing a cookie from a browser the attacker will be able to take over the session (as it is the problem with every cookie based session mechanism) but even if he gets the session database he would only be able to decrypt the session record tied to his session key. Compared to other methods (storing passwords in cleartext during a session) this is far better.

The latter can be minimized in impact by expanding the scheme to random temporary session passwords, created especially for this session only (see below).

Polybius runs as an Apache module and makes use of Apache’s reverse proxy features. Every web application running under Polybius SSO are accessed by defined prefixes. It’s the prefix which makes Apache forward (reverse proxy) the request to the specified web application. Apache’s proxy modules are capable of rewriting HTTP path responses, cookie domains and cookie paths.

Applications may still be used without a Single-Sign-On. Every application remains a standalone installation.

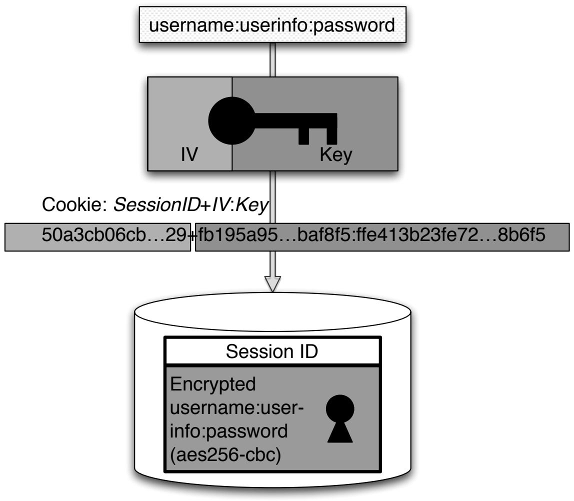
username:userinfo:password
IV
Key
Cookie: *SessionID+IV:Key*
50a3cb06cb…29+fb195a95…baf8f5:ffe413b23fe72…8b6f5
Session ID
Encrypted username:user-info:password (aes256-cbc)

Figure 1: User credentials are stored encrypted in the session database.

4 Polybius with synchronized passwords

In this case, synchronized passwords are needed, so the same password has to be set in all authentication databases used by the applications to be accessed by Polybius[2].

4.1 Basic HTTP authentication

The interaction with legacy web applications using http basic authentication[3] is shown in Figure 2.

- The user enters his username and his password on a HTML Form which is part of Polybius' infrastructure.
- The password has been checked against the authentication database. If it is wrong, it stops the procedure indicating an error.
- Random 256 bit AES key, 128 bit IV and 128 bit Session-ID are generated.
- The user name(s) and the password are encrypted with this Key and this IV using CBC mode of AES.
- This encrypted data is stored in the session database under the Session-ID as database key/index.

[2]The use of some kind of identity/user management software represents one possible scenario

[3]`WWW-Authenticate:`-Header followed by a `Authorization: Basic XXXX` reply, where `XXXX` is the base64-encoded form of `username:password`

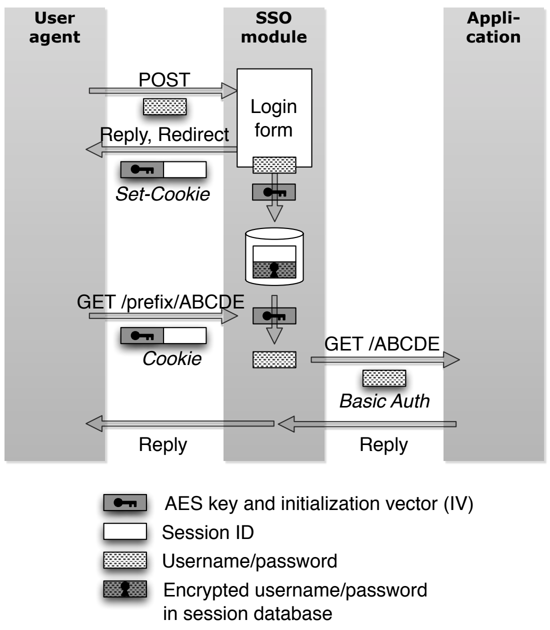

Figure 2: Cookie flow diagram for the basic http auth scenario

- A Set-Cookie-Statement is sent in the reply to the user's browser consisting of the session id, the AES key and the IV.

The user is now *logged on*. The legacy application is defined as an apache proxy target under a specific prefix. So whenever a GET/POST request comes in regarding an URL sharing this prefix the following happens:

- The user browser sends the SSO cookie along with his request. The Polybius apache module gets this cookie, and deletes it from the request header. The application "behind" the Apache proxy will not see it.
- The SSO module retrieves the session data stored in the database under the Session ID given in the browser cookie. It then decrypts it using the AES key and IV also being present in the cookie.
- The SSO module now constructs an `Authentication: Basic XXXX` header line in the request using the username and the password decrypted from the session database.
- Apache's proxy module now takes this http request and forwards it to the web application.
- The response is passed through to the user's browser.

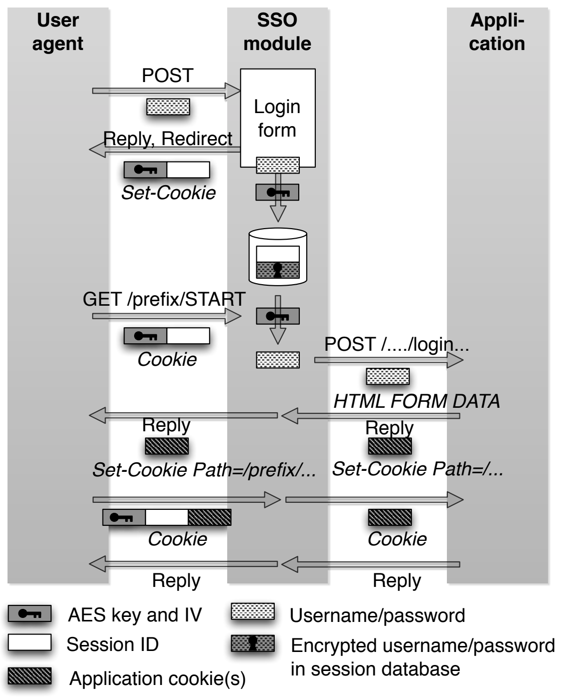

Figure 3: Cookie flow diagram for the session cookie scenario

4.2 Cookie-based authentication

The interaction with legacy web applications using a cookie-based session management is also possible using a "Polybius Login Helper" for the application. The communication diagram is shown as Figure 3.

- After the login phase (same as above) the user requests a special session start URL which in turn starts a login helper for that application.
- This login helper POSTs the appropriate FORM data to the application to "log in" as if it where the real user agent.
- The Cookie given back by the application is passed slightly modified to the user's browser: The Cookie-Path specification is changed so that this cookie is only sent when requesting the application (distinguished by the Apache proxy prefix).
- The SSO module makes sure that no SSO key and no SSO Session ID is passed to the application.

4.3 Example Apache configuration

In our test case installation, we included our SVN service in this scenario like this:

```
LoadModule polybius_module /usr/lib64/apache/modules/mod_polybius.so

SetOutputFiler POLYBIUS
SSLProxyEngine On

ProxyPass /svn/ https://svn.uni-konstanz.de/
ProxyPassReverse /svn/ https://svn.uni-konstanz.de/

<Location /svn>
       PolybiusAuthType basic
       PolybiusUidAttribute cfn
</Location>
```

5 Polybius with temporary passwords

The schema above can be modified to use temporary session passwords instead of "real" ones. Some changes in the authentication infrastructure is needed however because in most authentication systems the password (and the respective hashes) is a unique single-value property. Our idea is to slightly modify the existing infrastructure to add multiple temporary passwords against the applications may authenticate instead of the "real" one. The idea is sketched in Figure 4.

Applications usind LDAP BIND may use different LDAP subtrees with temporary elements to accomplish that (and to retrieve the user account data values). Applications using LDAP requests to get directly password data may be reconfigured to use also other attributes as data store. SQL database tables (for SQL authenticaten) may be altered.

The benefits of this system would be that even if an attacker manages to get the temporary password (by getting the AES key, IV and the session database) for one entry it will be useless after this temporary password expires.

6 Performance

6.1 AES decryption with OpenSSL

The AES implementation of OpenSSL has been used – OpenSSL is already loaded as a module in Apache. For the numbers, a single thread (single CPU) AES decrypting loop (using a 256 bit key and aes-cbc as cipher on 160 byte of data) on a rather legacy AMD Athlon 64 X2 will result in 1,8 seconds for 1 million decryption operations, or ≈500,000 decryptions per second. So AES performance will not have any significant impact.

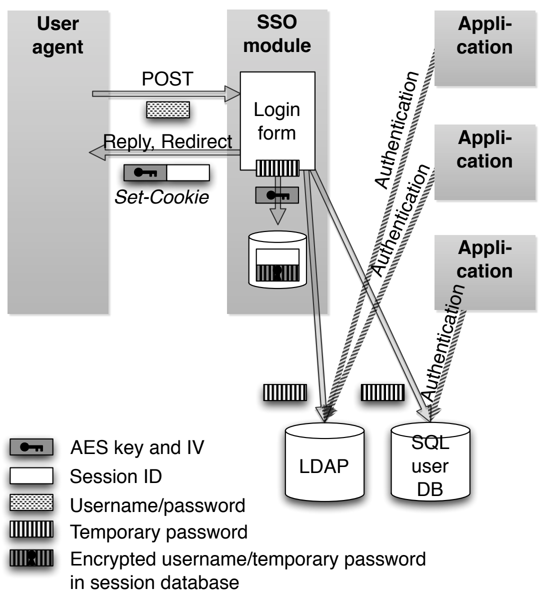

Figure 4: Using temporary passwords for SSO auth

Table 1: Performance impact of Polybius: Mean μ [requests/s] and standard deviation σ.

	Pure proxy		With SSO		
Concurrency	μ	σ	μ	σ	Difference
1	249.0	4.6	206.8	2.6	-16.9%
4	693.8	3.5	530.8	2.6	-23.5%
8	710.4	21.6	577.0	5.9	-18.8%
16	699.2	29.7	626.8	10.8	-10.4%
32	670.2	4.1	651.2	8.4	-2.8%
64	648.4	12.1	641.0	5.2	-1.1%

6.2 Latency due to database communications

Latencies occuring when communicating with the database (we used PostgreSQL, the module prepared statements to save processing time). We ran the "ab" benchmark program doing 3000 requests through the Apache Proxy to a static file on another server. To save SSL/TLS negotiation time we used the "keepalive" feature to pass the requests without reopening the connection. The results are shown in Table 1 and Figure 5, which show the following:

1. Performance loss due to the database compared to the non-SSO-passthrough is less visible when request concurrency is high. This is due to the fact that each database

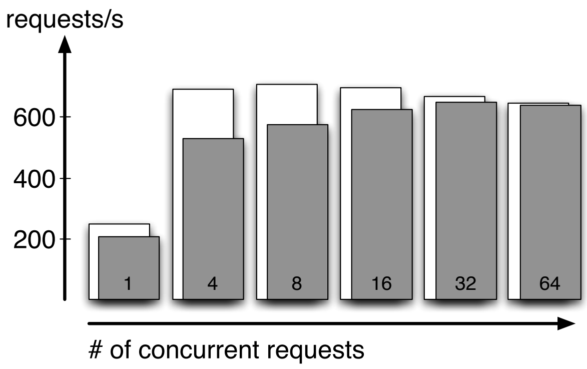

Figure 5: Request rates for proxy passthrough (white) and SSO operation (gray)

request introduces a specific latency where our module just has to wait for the answer.

2. In the single-concurrency-scenario each request takes 0.8 milliseconds longer (4.8 ms compared to 4.0 ms, resulting in 207 requests/sec compared to 249). Our database log (which I turned on for testing) shows that binding the parameter to the prepared statement plus executing it takes 0.3 ms.[4] The remaining offset is due to network latency.

7 Conclusions and future work

Recent developments allow the simplification of Single-Sign-On within an organization, as most applications are web-based and even more so in the near future. Server-side Web proxies are commonly used as load-balancing and failover mechanisms, show high performance, and are well known by systems administrators. Even in the unlikely case the SSO proxy should ever fail, the backend applications may still be accessed in the traditional way.

Combining these insights with a secret sharing mechanism allows Polybius to not only create a flexible and easily configurable as well as extensible SSO mechanism, it can also reduce the impact of break-ins or unauthorized disclosure. These combinations make Polybius well-suited for the academic environment with its openness and heterogeneity.

We are working on extending the number of backend protocols supported in Polybius and also believe it will be possible to include Polybius into cross-domain SSO systems such as Shibboleth or certificate-based login mechanisms.

[4] $\approx$ 0.2ms to bind to the variable and 0.1ms to execute it and return the result.

References

[Jas] Jasig. CAS 2 Architecture. http://www.jasig.org/cas/cas2-architecture. Accessed 2011-01-11. 2

[JZB06] Karthik Jaganathan, Larry Zhu, and John Brezak. SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows, June 2006. 2

[NYHR05] Clifford Neuman, Tom Yu, Sam Hartman, and Kenneth Raeburn. The Kerberos Network Authentication Service (V5). IETF RFC 4120, July 2005. 2

[Sco05] Scott Cantor, editor. Shibboleth Architecture: Protocols and Profiles. http://shibboleth.internet2.edu/docs/internet2-mace-shibboleth-arch-protocols-200509.pdf, September 2005. Accessed 2011-01-11. 2