Presented at: 4. DFN-Forum Kommunikationstechnologien ; 20./21.6.2011, Bonn, Germany

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[2]: 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 SSO session key is stored as an HTTP cookie in the user's browser (cf. 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 reduces the abilities of an attacker to gain any secrets.

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 the reverse proxy features. Every web application running under Polybius SSO is accessed using an application-specific prefix specified in the proxy configuration. The chosen prefix makes Apache forward

[2]We define a **legacy web application** to be an application which does not use some shared authentication mechanism consisting of exchanging cryptograms like kerberos tickets or shibboleth hashes to be verified against an authentication server and which cannot be modified to use them. Legacy web applications rely on the input of a username and a related password via a an HTTP POST request or a WWW-Authenticate-Header.

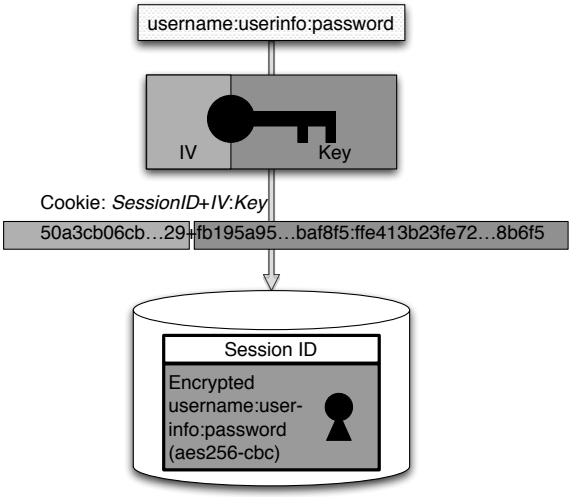

Figure 1: User credentials are stored encrypted in the session database.

(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 accessed without a Single-Sign-On. Every application may remain a standalone installation. Polybius also can be used to allow helper applications to automate processes involving one or more backend systems. Single logout is possible by simply erasing the database record belonging to this user.

4 Polybius with synchronized passwords

In this case, the organization has already integrated its applications to access a single user store or multiple stores with synchronized passwords. This can be obtained by identity or user management software.[3]

4.1 Basic HTTP authentication

The interaction with legacy web applications using HTTP basic authentication[4] is shown in Figure 2.

[3]The actual identities need not to be the same, but then Polybius needs access to a service which can link those identities.

[4]`WWW-Authenticate:` header in the server's reply, followed by the client repeating the request including `Authorization: Basic XXXX` in the request, where `XXXX` is the base64-encoded form of `username:password`.

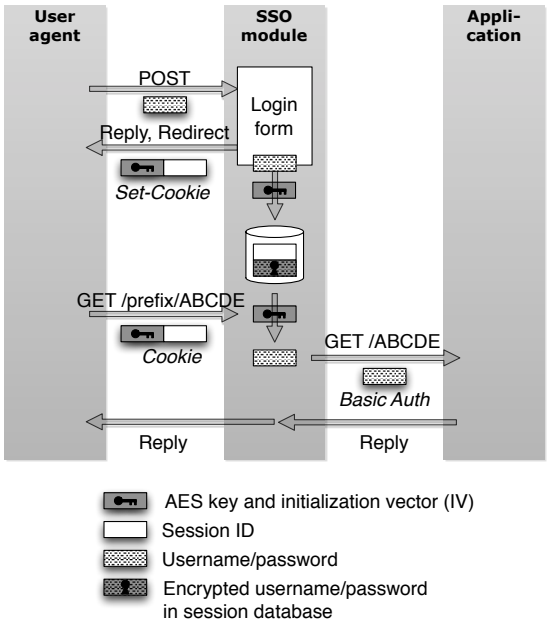

Figure 2: Cookie flow diagram for the basic http auth scenario

- 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, the servers aborts the request with an appropriate 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, IV) tuple using CBC mode of AES.
- This encrypted data is stored in the session database under the Session ID as the database key.
- A `Set-Cookie:` header 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 arrives regarding an URL under this prefix the following happens:

- The user browser sends the SSO cookie along with his request. The Polybius Apache module extracts this cookie and deletes it from the request header. The application behind the Apache proxy will not see it.

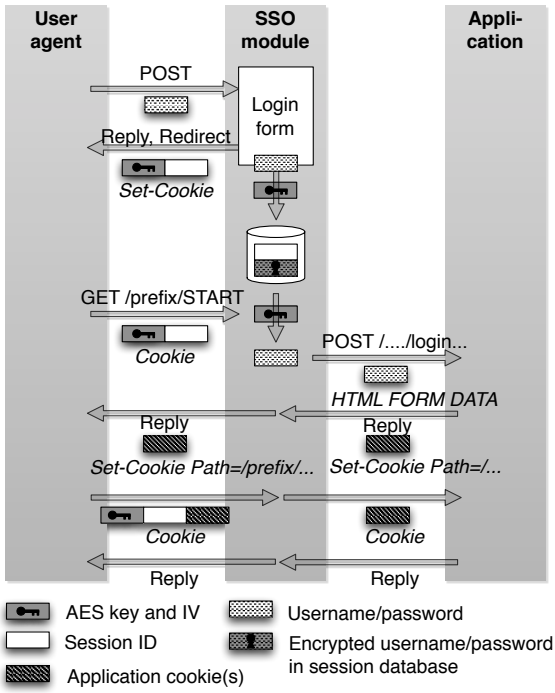

Figure 3: Cookie flow diagram for the session cookie scenario

- 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 included 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.
- The Apache proxy module now takes this HTTP request and forwards it to the web application.
- The response is passed through to the user's browser.

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 ensures that neither SSO key nor SSO session ID are passed to the application. Also, no secrets from the backend cookie will be leaked to browser.

4.3 Example Apache configuration

To include a service authenticated using HTTP Basic, such as an Subversion (SVN) version-control repository, the following Apache configuration fragment could be used:

```
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

Sometimes passwords cannot be synchronized between all applications. This can be due to conflicting character set restrictions or due to the fact that the application does not include a mechanism compatible with the identity management system.

This is no problem for Polybius, as the schema above can be extended to make use of temporary session passwords instead of "real" ones. Some changes in the authentication infrastructure is needed however, as the password (and the respective hashes) is a unique single-value property in many systems. The concept is to slightly modify the existing infrastructure to add multiple temporary passwords[5] against the applications may authenticate instead of the "real" one. The idea is sketched in Figure 4.

[5]The usage of a single-value would result in a situation where multiple concurrent logins for the same user are prohibited - which can be a desired feature.

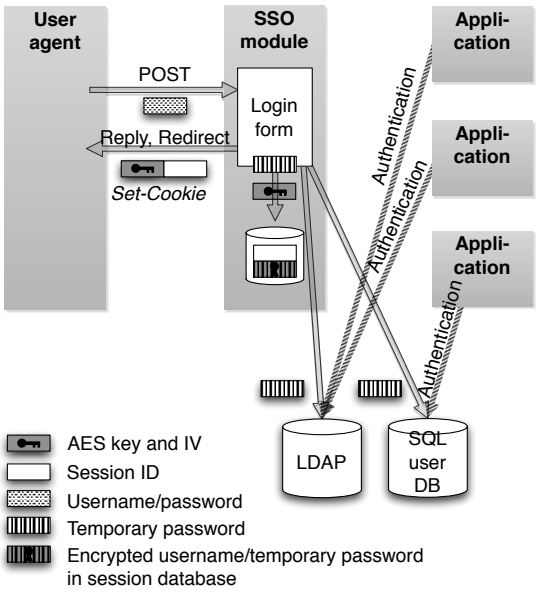

Figure 4: Using temporary passwords for SSO auth

Applications using LDAP BIND may use different LDAP subtrees containing only such temporary elements to benefit from multiple passwords (and to retrieve the user account data values). Applications using LDAP requests directly to read user data and verify it themselves may be reconfigured to use also other attributes as data store. Also, SQL database tables (for SQL authentication) can generally 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.

The idea - which is not implemented yet - consists of Polybius creating a random password and pushing it to all authentication sources/databases used by the configured legacy web applications. This random password is then stored in the same way as the “normal” user password would be saved encrypted in the Polybius session database.

6 Performance

6.1 AES decryption with OpenSSL

For our performance tests, the AES implementation built into OpenSSL has been used – OpenSSL is already loaded as a module in Apache. A single thread (single CPU) AES decrypting loop has been run on an AMD Athlon 64 X2 machine several years old, contin-

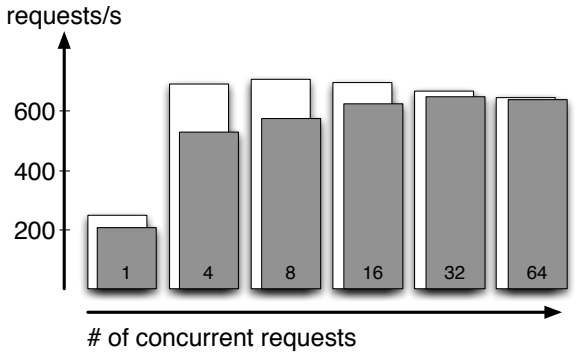

Figure 5: Request rates for proxy passthrough (white) and SSO operation (gray)

uously performing 256 bit key selection followed by AES-CBC decryption on 160 bytes of data. 1 million decryption operations, this setup requires 1.8s, equivalent to ≈550,000 decryptions per second. So AES performance on even an aging machine will not be the bottleneck in this application.

6.2 Latency due to database communications

Latencies occurring when communicating with the database (we used PostgreSQL as the module's storage engine, accessed using prepared statements for efficiency and security). We ran the "ab" Apache benchmark program doing 3000 requests through the Apache Proxy to a static file on another server, also consisting of 160 bytes. To model browsers keeping their sessions to the proxy open and thus save SSL/TLS negotiation time, we used the 'keepalive' feature of "ab". The results are shown in Figure 5, which allow the following interpretation:

1. Performance loss due to the database lookup is almost visible when request concurrency is high, which will be the typical operating point for highly loaded proxies. For weakly loaded proxies, performance will also not be a problem.

2. In the sequential test (concurrency=1), each request is slowed down by 0.8 ms longer (4.8 ms compared to 4.0 ms, resulting in 207 requests/s 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.[6] The remaining offset is due to network latency.

This has all been measured on a version of Polybius which has not yet been tuned for optimal performance, as our focus so far has been on functionality and correctness.

[6] ≈ 0.2ms to bind to the variable and 0.1ms to execute it and return the result.

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.

Also, the concept of temporary passwords can be extended to provide (1) flexible user rights delegation or (2) single-use authentication data, enabling the use of a one-time passwords or other restrictions when having to use a public terminal to access one's applications. The beauty of this approach is that, unlike other SSO infrastructure, such mechanisms can be added purely inside the SSO gateway, not a single configuration or source change is needed for the backend applications.

The University of Konstanz is evaluating the usage of Polybius to include their webmail and collaboration applications as well as the central SVN service in a uniform user portal.

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