Identity Management Best Practices

A CTSC Blog Series

November 2015

For Public Distribution

Jim Basney

About CTSC

The mission of the Center for Trustworthy Scientific Cyberinfrastructure (CTSC, trustedci.org) is to improve the cybersecurity of NSF science and engineering projects, while allowing those projects to focus on their science endeavors. This mission is accomplished through one-on-one engagements with projects to solve their specific problems, broad education, outreach and training to raise the practice-of-security across the community, and looking for opportunities for improvement to bring in research to raise the state-of-practice.

Acknowledgments

This document is a product of the Center for Trustworthy Scientific Cyberinfrastructure (CTSC). CTSC is supported by the National Science Foundation under Grant Number OCI-1234408. For more information about the Center for Trustworthy Scientific Cyberinfrastructure please visit: http://trustedci.org/. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Using & Citing this Work

Cite this work using the following information:
J. Basney. (2015). *Identity management best practices: A CTSC blog series* [Online]. Available: http://hdl.handle.net/2022/20452

This work is available on the web at the following URL:
http://trustedci.org/useful-links/

Table of Contents

1 Introduction

This technical report collects a series of CTSC blog posts on identity management (IdM) best practices published in 2014 for archival purposes.

2 Outsourcing Identity Management

Identity Management (IdM) in scientific cyberinfrastructure is a means to an end: provide convenient and secure access to applications, data, instruments, and services so scientists can focus on the science. Implementing a custom IdM solution can be a significant drain on resources for science projects. Outsourcing IdM can help with managing user identities, credentials, groups, and profiles. Three IdM outsourcing options that we see used in US scientific cyberinfrastructure are Globus Nexus, Agave, and Google Apps for Education.

Globus Nexus provides identity, profile, and group management as part of the Globus Platform, which is designed to support data-intensive collaborative research. Multiple research projects have adopted Globus Nexus for identity management, including US ATLAS, OSG, and KBase. Globus Nexus supports federated identities via InCommon/CILogon and Google OpenID, as well as user-managed groups. US ATLAS and OSG use CI Connect to integrate with Globus Nexus. The KBase Authentication developer tutorial demonstrates KBase's integration with Globus Nexus via an OAuth/REST API.

Agave is a science-as-a-service platform, designed to support science gateways, that was developed by the iPlant Collaborative. Agave provides hosted identity, profile, and group management via an OAuth/REST API. Science gateways can integrate with iPlant identities (hosted in the Agave platform), use Agave as a hosted OpenID Connect interface to their existing identity solutions, or leverage Agave's identity-as-a-service offering. For example, the Bioextract Server and CIPRES science gateways now accept iPlant identities for login via Agave, the Arabidopsis Informatics Portal and VDJServer leverage Agave's hosted identity service, and the Texas Advanced Computing Center uses Agave with their Active Directory to provide OAuth protection to their internal and external APIs.

The Google Apps for Education collaboration platform is widely used for smaller scientific collaborations, due to ease of setup and powerful collaboration tools (Docs, Forms, Groups, Chat, Hangouts, etc.) provided out-of-the-box. External applications can integrate with Google identities and Google+ profiles via OpenID Connect. Google Apps also supports SAML SSO for integration with campus identities. Google's Directory API provides an OAuth/REST interface to Google Groups.

3 HTTPS Best Practices

Using HTTP Over TLS (HTTPS or Hypertext Transfer Protocol Secure) helps users ensure that they are connecting to the correct web site and that their communications are protected from eavesdropping and tampering in transit. Using HTTPS for banking and e-commerce web sites is an obvious requirement, but HTTPS is also valuable for scientific cyberinfrastructure (CI), where users log in to access valuable scientific resources and rely on the integrity of their research data. Always-On-SSL describes the recommended best practice of enabling HTTPS across an entire website, rather than only specific areas of the site, to protect the user's entire web session.

The first step to enable HTTPS on a web site is to obtain a certificate for the site. CI providers should use HTTPS certificates issued by certificate authorities that are well known to standard web browsers so users do not learn to click through security warnings for so-called self-signed certificates. In many cases, CI providers can obtain HTTPS certificates from their home campus' IT department, which may be participating in the InCommon Certificate Service program or have an established contract with another certificate provider.

Tools and guides are available to assist with maintaining a secure HTTPS configuration. The Qualys SSL Server Test tool analyzes a site's HTTPS configuration to identify problems and areas for improvement. Qualys updates their test tool to check for the latest HTTPS security issues (such as the recent Heartbleed bug), so using the tool periodically can help verify that a site's HTTPS configuration remains up-to-date. Comodo and DigiCert also provide HTTPS test tools. For up-to-date HTTPS server configuration recommendations and examples, see the guides provided and maintained by Mozilla and Qualys.

4 Self Service Password Reset

Self service password reset is an important capability for any scientific cyberinfrastructure (CI) providers that manage passwords for a large user community. Without it, providers risk being overwhelmed by support requests from users who forgot their password and risk being the victim of social engineering attacks against support staff following ad-hoc, manual password reset procedures.

We differentiate between *password change* and *password reset*. Password *change* is when the user enters both current and new passwords to update the password for an account. Password *reset* is when the user has forgotten the current password for the account and needs to establish a new password.

CI providers should avoid re-implementing the password reset workflow if possible. Using external identity providers (e.g., InCommon and/or Google) can avoid the risks of managing passwords directly, enables users to log in via an account that they use regularly (so users are less likely to forget the password), benefits from security features such as two factor authentication, and leverages the password recovery support of the external identity provider(s). Alternatively, CI providers could use an existing password reset workflow built-in to their web application framework (e.g., Joomla or Laravel) or identity management (IDM) platform.

However, in our experience CI providers still sometimes find the need to implement password reset in their unique environment. In this article, we provide an example email-based password reset workflow and discuss design choices and risks that CI providers should consider when implementing password reset. The workflow assumes that users have previously registered a contact email address that can be used for password resets.

4.1 Password Reset Workflow

The goal of the following workflow is to allow a registered user to reset their password without requiring assistance from support staff.

1. On the “Sign In” page, the user clicks the “Forgot Password?” link.
2. The user is prompted to enter their username or registered email address.
3. The user is prompted to “check your email for a password reset code to enter below”.
4. The user enters the password reset code on the web form.
5. The user enters their newly chosen password.
6. The user can now sign in with the newly chosen password.

4.1.1 Behind the Scenes

To implement this workflow, the web application performs the following actions behind the scenes:

When the user enters a username or email address (step #2), the web application checks for a match with a valid user account. As with any input provided by the user, the web application sanitizes the username and email address values before querying back-end databases, to protect against injection attacks. In response to the user's submission, the web application does not indicate whether a match was found, to avoid disclosing the existence of registered accounts to unauthenticated users. Instead, whether a match was found or not, the web application displays, "Please check your email for a password reset code to enter below. If you do not receive an email message, please contact the help desk" (step #3). If the username or email address provided by the user does not match a valid account, no further action is taken.

If the web application finds a valid account matching the user's input, the next step is to generate the password reset code (for example, an 8 digit random number), store a salted hash of the reset code (for example, using bcrypt) with a timestamp and the user's account ID, and send the email message to the user's registered email address. The password reset code is a time-limited, one-time-use random "nonce" value to confirm that the user received the message at the registered email address. The web application removes stored reset code entries immediately after use or after the time limit (for example, 15 minutes) has elapsed.

Next, the user enters the password reset code from the email message on the web form (step #4). The web application hashes the entered reset code and searches for a match among the current stored values. If no match is found, the application displays an error and prompts the user to try again (for example, allowing up to 3 tries before aborting the process). If a match is found, the application prompts the user to enter a new password (twice for confirmation) (step #5) and checks that the new password is sufficiently strong. If it is, the application changes the user's password to the new value, removes the stored reset code entry, and sends a confirmation email to the user's registered email address. The user can now return to the standard "Sign In" page and proceed to log in with the new password (step #6).

4.1.2 Password Reset Email Messages

The above workflow sends two email messages to the user's registered email address: 1) the message containing the password reset code and 2) the message notifying the user that the password reset completed successfully. These messages should follow recommended practices for email communications, including a trustworthy From address (in the correct DNS domain) and no HTML content. The messages should also include instructions for contacting the help desk if the user did not initiate the password reset. The user's password should never be sent in email messages.

4.1.3 Logging and Monitoring

The password reset capability can be a target for attacks and a source of user support issues, so it is especially important to log all system activities related to password resets and monitor for unexpected behavior. Log messages should have accurate timestamps and should include the originating IP address for password reset requests.

4.1.4 Risks

The primary risk for the password reset process is the possibility that an attacker could reset a valid user's password and thereby obtain unauthorized access. Potential attack vectors include:

- ❏ *Disclosure of reset code via email*: Since the password reset code is sent over unsecured email, it could potentially be disclosed via email account compromise or network eavesdropping, allowing an attacker to use the code to change the user's password. Enforcing a short lifetime on the reset code limits the window of vulnerability against this attack.
- ❏ *Network disclosure of passwords*: The web application should follow HTTPS best practices to protect passwords against active and passive man-in-the-middle and phishing attacks.
- ❏ *Exposure of reset code database*: Storing reset codes in hashed (salted) form in the web application protects against disclosure of valid reset codes due to inadvertent disclosure of the reset code database.
- ❏ *Brute force attacks on reset codes*: Generating long, random reset codes, valid for only a short time, makes it infeasible for an attacker to successfully guess a reset code through brute force. Aborting the reset process after 3 failed reset code entries also protects against guessing attacks. However, beware making reset codes so long that they are inconvenient for users to input. (8 random digits is a reasonable length.)
- ❏ *Compromise of the password reset front-end web application*: In a system architecture with a back-end authentication system (LDAP, Kerberos, etc.) that may be shared across multiple front-end systems, enabling a front-end web application to reset passwords introduces the risk that an attacker who compromises the web application could reset many user passwords and gain further unauthorized access. Unlike password *change* (which requires knowledge of a current valid password prior to making password updates), password *reset* trusts the web application to update passwords without further validation by the back-end authentication system. Isolating the password reset functionality to a dedicated, well-secured front-end system can help to mitigate this risk, as well as logging and monitoring on the back-end system.

4.1.5 Examples

The "Forgot Password?" links on the XSEDE User Portal and on HUBzero provide illustrative examples of self service password reset functionality similar to what is described above.

4.2 Self Service and Exceptional Cases

If all goes well with the above workflow, the user is able to reset their password without assistance from help desk staff, hence "self service". However, the user may still need assistance if (for example) they lose access to a previously registered email address and therefore can not complete the self service workflow. It is important to have documented processes for handling these exceptional cases at the help desk without introducing new risks for social engineering attacks. A phone call from the help desk to a previously registered phone number can help re-establish account ownership. However, when in doubt as to the identity of the account holder, it may be better to ask the user to create a new account rather than risk improperly resetting the password on an existing account.

5 Conclusion

For current best practice recommendations and other information about CTSC activities, visit the CTSC blog at http://blog.trustedci.org/.