A Hardware-secured Credential Repository for Grid PKIs

Markus Lorch
Virginia Tech
mlorch@vt.edu

Jim Basney
NCSA
jbasney@ncsa.uiuc.edu

Dennis Kafura
Virginia Tech
kafura@cs.vt.edu

Abstract

Public Key Infrastructures suffer from usability and security problems associated with the request for and secure management of end user credentials. Online credential repositories provide mechanisms to ease these shortcomings but pose attractive targets for attacks due to the accumulation of credentials and the need for remote access to these credentials. Through the extension of an existing credential repository with a cryptographic co-processor for secure storage of credentials an increase in the security of the service can be achieved. This higher security permits the use of online credential repositories with a wide variety of certificates without violating certification authority regulations. Also, the improved performance afforded by hardware support improves the scalability of a centralized credential storage.

1. Introduction

In a Public Key Infrastructure (PKI), a subject authenticates by proving possession of a private key. The subject's identity is bound to a publicly known key (inverse to the private key) via a certificate issued and signed by a commonly trusted certification authority. The security of PKI mechanisms is dependent on the secrecy of the subject's private key. Protection of the private key is the responsibility of the subject (the user).

All too often the subject's private key is vulnerable due to improper "key hygiene". Common lapses in "key hygiene" include failing to encrypt the key with a strong password and storing the key on an insecure file system. Tools and protocols sometimes increase key vulnerability. For example, Internet Explorer stores private keys without a passphrase by default [1], and both OpenSSH and OpenSSL make it trivial to create unencrypted private keys. Inadequate protection of the private key opens the door for key compromise when the security of the hosting computer or network is breached. A network breach can compromise keys on network filesystems or network backups. Due to the distributed nature of PKI mechanisms it is difficult or impossible to enforce proper key protection policies unless the keys are stored only on hardware tokens that do not permit key extraction (e.g. smart-cards).

User management of private keys also presents significant usability challenges. Generating keys and obtaining a signed certificate can be a confusing and time-consuming process. For example, certificate and key generation in Internet Explorer "may take up to 12 steps with scary messages about scripting violations" [1]. The resulting certificate and private key are stored locally on the user's computer, in a dedicated certificate store or in regular files. To use the certificate and key on another computer, the user must export the credentials from one computer's certificate store and import them to the other computer's certificate store, or manually copy files between computers, being careful to maintain correct file permissions and use secure transmission protocols.

Generating and storing users' private keys on smart-cards is an attractive solution for managing private keys. The private key remains secure in the smart-card's memory, and the smart-card provides key mobility, allowing the user to use his credential on different terminals. In addition, smart-cards provide for "two-factor" authentication: something you have (the card) plus something you know (the PIN needed to access the smart-card).

However, deployment of smart-cards is not a trivial task mainly due to the lack of smart-card reader devices at users' workstations as well as software integration problems. For example, a Grid computing pilot project on the Virginia Tech campus that used smart-cards for authentication encountered issues with a lack of reader drivers for Unix based workstations and the lack of support for the use of smart-cards in the Globus Toolkit [2]. While it was possible to use smart cards with the Java CoG Kit for Globus [3], the licensing of the necessary libraries prohibited a deployment of this solution.

For applications and environments where deploying smart-cards is not feasible, a "password-enabled PKI" can provide a similar level of security and usability using trusted PKI servers for managing user keys [4]. In this paper we describe a password-enabled PKI solution using a hardware-secured online credential repository that provides secure storage for users' private keys. The system creates the keys in a tamper resistant cryptographic co-processor and issues short-term credentials, signed with the protected keys, to authenticated users. Storing keys in the co-processor addresses the hygiene problem by not allowing keys to be extracted. The co-processor also provides for faster key generation using good random numbers. The security of the repository is improved because keys are protected in the co-processor even if the repository server is compromised. We have implemented the system by modifying an existing credential repository for Grid computing called MyProxy [5].

Our paper is organized as follows. In the next section, we present related work, reviewing existing implementations and research of password-enabled authentication systems. In section 3, we give a detailed description of the MyProxy system, which we modified for this work. Section 4 presents our contribution, a hardware-secured version of the MyProxy system. We then conclude with a discussion of future work and a summary of the paper.

2. Related work

The Kerberos authentication service [6] is a widely-used password-enabled cryptographic system, providing password-based access to cryptographic keys from an online Key Distribution Center (KDC). Each user shares with the KDC a secret key generated from a password. Clients retrieve limited-lifetime credentials encrypted with the secret key from the KDC and decrypt them with the user's password. Version 5 of the Kerberos protocol adds pre-authentication [7], requiring requests for credentials to be encrypted with the shared secret, so attackers can not make valid requests for credentials to then perform offline dictionary attacks on them. The Kerberos KDC is an attractive target for attack, as the compromise of the KDC machine can reveal all user keys for the Kerberos realm. Itoi [8] integrated the IBM 4758 secure co-processor into the Kerberos V5 KDC to protect against KDC compromise by moving critical KDC operations to the co-processor so user keys are never available unencrypted on the KDC host machine.

SPX [9] is an authentication system similar to Kerberos but built on public key cryptography. An SPX server holds users' long-term PKI credentials, encrypted with the users' passwords, and users authenticate to the SPX server to retrieve their long-term credentials which they use to sign short-term credentials that are stored unencrypted on the local filesystem to be used for the remainder of the session. The short lifetime of the stored credentials limits their exposure to compromise.

The Globus Toolkit's Grid Security Infrastructure (GSI) [10], a widely adopted standard for Grid computing, also has short-term PKI session credentials, called proxies. MyProxy [5] is an online credential repository for GSI credentials that allows users to authenticate with a password and retrieve short-term proxy credentials. Unlike the SPX server, users' long-term credentials never leave the MyProxy server. Instead, the MyProxy server uses the user's long-term key in the repository to sign a proxy (or delegation [11]) certificate, and sends the certificate to the authenticated client. As with the Kerberos KDC and the SPX server, the MyProxy server is an attractive target for attack, as it holds many user keys. In both the SPX server and the MyProxy server, user keys are encrypted with user-chosen passwords, so a server compromise opens the keys to dictionary attack. This vulnerability motivates our work to integrate the MyProxy server with the IBM 4758 cryptographic co-processor.

Sandhu et al. [4] survey two classes of password-enabled PKI solutions: virtual soft tokens and virtual smart-cards. In their terminology, virtual soft token systems allow users to retrieve private keys from a credential server (for example, a traditional credential repository) and then use the keys directly without further interaction with the server, whereas in virtual smart-card systems users don't have direct access to their private keys but instead must interact with the server to perform signing operations. Virtual smart-card systems split the private key into two components, one computed from the user's password and the other stored on a secure online server. The user and server participate in a signing protocol that does not require either of them to disclose their split key to the other or to reconstruct the complete private key at any point. If the virtual smart-card server is compromised, the attacker can perform a dictionary attack against the server key shares, since the corresponding user shares are computed from user passwords. However, this type of attack is much slower than a dictionary attack against DES encrypted keys. Yaksha, an early example of a virtual smart-card system, replaced shared user secrets in Kerberos with split RSA keys to protect against KDC compromise [12]. It is possible to further protect against server compromise by distributing threshold split keys to multiple servers such that a subset t of n servers work together to generate a

threshold signature without reconstructing the complete private key, so a compromise of less than *t* servers cannot reconstruct the complete private key [13].

A number of commercial PKI credential repositories are available, typically as an add-on to Certification Authority products to support credential mobility [14, 15]. The nCipher netHSM (www.ncipher.com/nethsm/) provides a network-attached hardware security module for storing private keys. Also, the IETF Securely Available Credentials (SACRED) working group is developing a standard protocol for network-based access to credential repositories [16].

Online credential repositories can be categorized as either mechanism-aware or mechanism-neutral. A mechanism-aware repository (like MyProxy) can support mechanism-specific protocols for credential retrieval, which allows the repository to implement policies on the credentials that clients can retrieve. For example, a repository can hold long-term user keys that never leave the repository but are instead used to sign short-term credentials. Thus, credential revocation can be implemented by simply removing the keys from the repository. However, allowing the repository server direct access to the keys weakens non-repudiation claims. In contrast, a mechanism-neutral credential repository (like SACRED) can store many types of credentials. Credential encryption and decryption is performed by the client, so the repository itself never has access to the unencrypted credentials.

Online Certificate Authorities [17, 18, 19, 20], which create new credentials on demand, can be an alternative to credential repositories. Users authenticate to the online CA and issue a certificate request. The CA sets the user's authenticated identity in a certificate then signs and returns the certificate to the requester. This allows users to retrieve short-term certificates from the online CA on demand without needing to manage long-term private keys. For example, KCA [18] is a popular online CA for Grid sites that allows users to authenticate via Kerberos to retrieve short-term GSI credentials. Like a traditional CA, the security of the online CA's private key is paramount. The CA's private key may be secured by a hardware security module. Threshold split-key approaches can distribute CA functionality across multiple CAs for further protection [19]. One drawback to online CAs is the potentially high cost of adding a new CA to the PKI, which may require renegotiating trust agreements with relying parties. Credential repositories can provide a more flexible solution, since they need not be tied directly with a CA but could be deployed to manage credentials for a single user, a small group within an organization, or a large collaborative group that spans organizations (and CAs).

3. MyProxy Online Credential Repository

MyProxy was originally developed to delegate Grid credentials to trusted web servers (called Grid portals) so they can perform authenticated operations (submit jobs, transfer files, etc.) on the user's behalf without modifying standard web browsers [5]. Users login to the portal and enter a MyProxy server name, username, and passphrase that the portal can use to retrieve short-term proxy credentials for the user. Instead of storing long-term user credentials on the web server, the MyProxy approach uses a separate, dedicated credential server (the MyProxy server) to protect the long-term credentials against web server compromise. The web server holds only short-term proxy credentials it has retrieved from the MyProxy server.

MyProxy has since been extended to support Grid credential mobility and credential renewal. It is common practice for users to sign-on to the Grid from different machines, creating a requirement for credential mobility. Rather than copying their long-term credentials to these different machines, with the associated usability and security concerns, users can store the credentials on the MyProxy server, protected by a passphrase, and retrieve a short-term proxy credential from the MyProxy server when needed by authenticating with the credential passphrase. Credential renewal allows users to avoid delegating long-lived credentials to long-running tasks. Instead, they can set renewal policies for credentials in the MyProxy repository that allow trusted job management systems (JMS) to retrieve new credentials for running tasks before they expire. The JMS must authenticate with a credential that is allowed by the user's renewal policy, then prove possession of the credential to be renewed, before retrieving a new short-lived proxy credential for a user's long-running task.

MyProxy can be integrated with a Certificate Authority whereby new user credentials are created by the CA and loaded into the MyProxy repository with a preset password. Administrators distribute default passwords to users who should immediately change them. Using MyProxy in this way allows users to obtain PKI credentials without going through a (potentially confusing) certificate request process and without placing a key management burden on users.

By keeping long-term keys in the MyProxy repository and restricting clients to retrieving only short-term credentials, the MyProxy server becomes a central point of control and monitoring for the user's credentials. Removing long-term credentials from the

repository provides a simple form of revocation, as any outstanding short-term credentials will soon expire. Administrators can monitor the MyProxy logs to detect suspicious activity or assess damage if a password or credential is compromised.

4. Hardware-secured MyProxy

We have built an online credential repository that extends the widely used MyProxy software to employ a cryptographic co-processor for the secure storage of private keys. The co-processor not only protects the user's keys from potential attackers in the case of a host compromise, but also prevents access to those keys by administrators. The co-processor adds three important features to the functionality that comes with a software-only MyProxy installation:

1. By generating and storing the keys only on the cryptographic co-processor the issues associated with key hygiene are nonexistent. Also, the use of a hardware random number generator enables stronger and faster key generation.
2. As the keys are not extractible from the co-processor, no third party can ever have access to the keys directly. Many Certification Authorities require this as it is the basis for non-repudiation guarantees.
3. Due to the tamper resistant properties of the hardware token a host machine does not require extensive physical security and can be located in semi-secure areas comparable to housing an ordinary server machine.

4.1 Hardware and software used

The hardware token in use is an IBM 4758 cryptographic co-processor. The IBM 4758 is FIPS-140 certified at level 4, which is the highest possible certification for commercial security granted by the U.S. Department of Commerce's National Institute of Standards. It consists of a fully self-contained, programmable computer on a battery powered PCI adapter card with a tamper responsive casing. It has a built in cryptographic processor for fast public key cryptography and comes with 4MB of non-volatile storage for keys, certificates and firmware. We were able to create and store more than 800 2048-bit RSA key pairs on the device. Through customization of the firmware much larger numbers may be possible (theoretically up to 10,000) by optimizing the on-card memory management. Several co-processors could be installed in a single machine to increase the capacity even more. Typical smart-cards in contrast provide storage of 16-32 KB which is enough for 3-5 key pairs and certificates. The IBM 4758 can run secure applications directly on the adapter's own general purpose processor (an Intel 486) while cypto operations are realized in a crypto processor.

To interface the card with MyProxy, IBM's open source software "openCryptoki" (http://www-106.ibm.com/developerworks/security/library/s-pkcs) was used. OpenCryptoki provides an implementation of the standard PKCS11 [21] programming interface developed by RSA for interaction between applications and personal security tokens such as smart-cards. Most of the PKCS11 functionality is implemented on the co-processor itself in the form of a specialized PKCS11 firmware that replaces the standard IBM Cryptographic Component Architecture firmware more typically used with the 4758. The complete system is implemented on Linux using IBM's Open Source Linux driver and the Linux management toolkit for the 4758. Since our modified version of MyProxy interfaces with the 4758 via the standard PKCS11 interface, we expect it to be portable to other cryptographic hardware modules.

4.2 Credential generation

To enforce proper "key hygiene", keys are generated directly on the co-processor and are marked non-extractable. Neither the end-entity (the user) nor the administrator can extract such keys from the device. Instead a user can use his key exclusively to request a time-limited proxy credential. The user must supply his identity and password. The decision to retain the keys in the co-processor limits our approach to those applications that support the use of proxy certificates.

Traditionally end-entity certificate requests are generated as follows: First a new key pair is generated on the user's workstation (i.e. by a web browser), and then a certificate request structure is created with the requesting user's identity information and his newly generated public key. This structure is then signed with the newly generated private key and sent to a CA. The CA, upon verifying the user's identity, will create a certificate that binds the user's identity to the public key the user specified in the certificate request.

The creation of user key-pairs on the (remote) hardware token required a new protocol and mechanism for certificate requests. The traditional procedure described above is not possible if the user does not have direct access to the private key. Thus we have created a web interface to the credential repository where a user can provide his personal information together with an initial password that will later protect his credentials. The repository will create a key pair and a certificate request and send it on to the CA. The user will get a confirmation including his

newly created public key which, if required by the CA, the user can present to a CA appointed registration authority for the identity vetting procedure. CAs typically make new certificates available for download to the user. As the certificates themselves do not need to be protected an automated routine on the MyProxy server can retrieve and install the certificates without user or administrator intervention once these become available. Installation of a new certificate merely requires the storage of this certificate along with the user's public key and access control information in the MyProxy credential database.

4.3 Changes to MyProxy

The changes required to enable MyProxy to leverage the IBM 4758 were mainly placed in the underlying OpenSSL security library. OpenSSL provides functionality by which the application code can select an external implementation of a mechanism-specific set of cryptographic functions at runtime. Such external implementations are referred to as cryptographic engines within OpenSSL. The MyProxy server has been modified to select a custom PKCS11 engine for all RSA related operations. This almost alleviated the need for changes in the MyProxy server code as the use of PKCS11 is otherwise transparent to MyProxy as well as to the security libraries of the Grid Security Infrastructure that MyProxy uses.

However, a significant change to the behavior of MyProxy with respect to process management was required. The software-only MyProxy created a new child process to serve each incoming client request but PKCS11 forbids the use of the same PKCS11 session in different process spaces. Our current prototype handles requests sequentially in a single process to temporarily work-around this problem. Of course, a single process provides inadequate performance for multiple clients. Creating individual PKCS11 sessions after a client process has been forked is undesirable as a PIN has to be supplied to gain access to the co-processor. Storing the PIN in the server process memory makes it vulnerable to attack. Furthermore the possible large number of concurrent sessions may overload the co-processor and make it vulnerable to denial-of-service attacks. Instead, we plan to implement a server that utilizes a fixed pool of worker processes. At startup, the administrator will enter the co-processor PIN and the server will spawn a set of processes which will use the PIN to initiate individual sessions with the co-processor and then immediately scrub the PIN from memory.

A significant functional change to the standard MyProxy is the way a requested proxy certificate is signed. In the software-only version the user's private key is decrypted and loaded into the main memory of the host machine and then a signature for the proxy certificate is created using the main CPU of the host. In the hardware enhanced version only the public key of the user is loaded into the host computer's memory. The PKCS11 interface is used by the OpenSSL engine to transparently locate the corresponding private key on the co-processor and to create a signature utilizing the co-processor's RSA implementation in hardware. The private key never leaves the co-processor.

In the standard MyProxy the user's private keys are encrypted individually with a user supplied password when a user uploads his credentials. When the delegation of a new proxy certificate is requested this password has to be supplied in order for MyProxy to be able to access the user's private key (additional access control mechanisms such as mutual authentication of the computer from which the user requests a proxy may also apply). In our modified version of MyProxy the user's private key is never loaded into the host computer's memory but rather all cryptographic functions that require access to the key are handled on the co-processor. PKCS11 only provides for a single login which grants the accessing application the right to use all the private keys on the device. Access control to individual PKCS11 objects is not supported by the interface. Instead, access control is performed by the MyProxy server by checking a one-way hash of the user supplied password against the corresponding entry in a password file.

While performance aspects are not the most important considerations of this work it is interesting to note that the IBM 4758 implements RSA operations in hardware and thus frees up the main CPU of the host computer for other tasks. However, the co-processor is not a crypto accelerator and considerable overhead is imposed by additional the software abstraction layer (the PKCS11 implementation). The result is a lower performance for crypto operations when compared to the standard implementation using the main CPU of the host computer (in our case an Intel Xeon @ 2.4GHz). We experienced a total performance hit of 15% for delegation requests without mutual PKI authentication and 19% when the clients authenticated with a PKI credential to the MyProxy server (all RSA operations are performed on the co-processor, including those for the TLS protocol used by the Grid Security Infrastructure). Utilization of the main CPU was significantly lower with the co-processor enabled. Newer generations of cryptographic co-processors may provide much higher public key speeds. For example in our evaluation of an IBM cryptographic accelerator (IBM 2058) we could perform 66 times more 2048bit RSA sign operations per second with the accelerator

support than with the above mentioned general purpose CPU (using the openSSL speed command).

4.4 Security assessment

Our modified system provides higher key protection than the original mechanisms in MyProxy. In the original version an attacker that compromises the MyProxy server could learn user's passwords as they send requests for credentials. Once in possession of the password, the attacker could decrypt and exploit the user's private key.

In the new system neither an attacker nor a user nor an administrator can ever extract the private key from the co-processor even if the security of the host machine is breached. The PKCS11 PIN with which MyProxy authenticates to the co-processor has to be manually keyed in at the console when the service is started. For higher security and to avoid keystroke logging an external keypad could be used.

The fact that the co-processor guarantees that nobody can ever extract the user's private keys allows for the use of this credential repository with most currently issued certificates. Common key handling procedures and requirements (i.e. that entities other than the end user have no access to the private key; that the private key is never available unencrypted) stated in many CA's Certificate Policy and Certification Practice Statements are fulfilled.

In addition, using a hardware-secured online credential store with password protected access to credentials not only provides for the simpler dissemination and protection of PKI credentials but also offers higher security than a username/password authentication scheme alone. The reason for this lies in the fact that the user's keys can never be extracted from the hardware token and thus, even if an attacker manages to gain knowledge of the user's password, the keys can not be copied. The attacker could merely request proxy credentials from the service if the user's password is known. This has the significant advantage that keys and certificates do not need to be revoked but merely the password changed when such a compromise is detected. As the credential repository is the only place from where such a proxy could be requested these requests can be monitored closely and misuse detected (which is much less possible if a key is compromised and copied to a remote location; thus the traditional need for a revocation).

We believe that the advantages of relieving the user from key hygiene procedures coupled with the convenience of remote access to delegated credentials are well worth the usage restrictions that come with this scheme as a user's end-entity credentials are not available directly to the user's applications. In our experience, certificates issued for grid computing are rarely used for other tasks than authentication and the creation of proxy certificates. Many grid CA policies do not even allow them to be used in other applications.

5. Future work

We found that we could further improve the security of the credential service if the PKCS11 token would support access control decisions on an individual bases for each private key. Currently our credential service authenticates and authorizes access to use a specific private key but, due to limitations in the PKCS11 standard, the credential service itself has access to all the keys on the hardware token. This violates the least privilege access principle [22] and does not allow for localization of a security breach. We have investigated the possibility of protecting the end-entity keys from unauthorized usage in such a case and limiting the effects of a security breach.

The inability of PKCS11 to perform access-control on an individual key basis is due to its original focus on personal security tokens, such as smart-cards. Today PKCS11 is moving towards the use in servers where many different credentials are stored on the token and thus individual access control is required. There has been some discussion in the PKCS community on how these scenarios can be supported better and the next major version of PKCS11 (version 3) may incorporate appropriate extensions.

An approach that would indefinitely increase the storage capacity of the hardware token is the outsourcing of protected keys. This requires a guarantee that keys can never exist unencrypted outside the device, even if the user's password has been compromised. Such a storage system could be realized as follows.

1. Key pairs are created on the device only and the private key is marked "sensitive", which allows extraction only in encrypted form.
2. Upon creation of a new private key it is wrapped (encrypted) with a combination of two keys, one key is derived from the user's password and the other key that is only available to the hardware device internally and un-extractable. The second key is also marked such that it can only be used for wrapping and unwrapping of keys.
3. A wrapped key can be stored in any location outside the hardware token.
4. If the wrapped key needs to be used it will be re-imported into the hardware token and unwrapped using the user's password and the token's internal key. As the

unwrap operation requires both the user's password and the wrapping key held on the device it is guaranteed that even if the user's password is compromised the user's key could never be unwrapped anywhere but in the hardware token.

5. The semantics of the unwrap operation have to be such that an unwrapped key cannot be extracted from the hardware token in the clear and may only be wrapped with token internal wrapping keys that are non extractable.

Unfortunately this mechanism cannot be implemented using the current PKCS11 standard (V2.x) due to the following two reasons. First, in PKCS11 there is no mechanism that would allow us to define that a user's key contained in the co-processor can only be wrapped using the combination of user and token keys as described above. While an attacker could not extract a key unencrypted it is possible to extract the key encrypted with the attacker's choice of a wrapping key. Second, the current PKCS11 definition of the unwrap operation does not mark the resulting keys as "sensitive." A subsequent modification of the attributes could fix this setting. But this leaves open the possibility of a timing-based attack that can allow the extraction of the key in unencrypted form. An alternative would be to program the co-processor with custom functionality similar to the work done by Itoi [9] but we would lose the abstraction and thus portability advantages that PKCS11 provides.

Another item for future work is a system extension that will allow a user from a different administrative domain to get a locally trusted certificate. Instead of entering his personal information into the certificate request module, the user would authenticate to the certificate request module using a foreign public key credential (issued by a CA that is not generally trusted on the local resources, a common situation in today's grid environments). The certificate request module would generate a certificate request to the local CA and also provide the proof of identity (based on the user's authentication) to the CA. The CA in turn can then, based on the trust in the foreign certificate, either issue a local certificate without the need to vet the identity of the user again or perform additional identity vetting. This is similar to an Online CA as the time and effort involved in getting a local certificate is much reduced when compared to a direct request at the CA but has the advantage that no additional Online CA is required, nor do new trust relationships (either for an Online CA or for the foreign CA issuing the original certificate) need to be configured on the local resources.

As mentioned earlier, the fact that the user's private keys are only stored in a remote credential repository and cannot be extracted limits the application of this system to grid authentication scenarios that employ proxy certificates. The development of a client interface exposing a standard PKCS11 application programming interface would allow the system to be used with any PKCS11 capable application. The hardware device would simply be outsourced. The connection between the local PKCS11 library stub and the remote PKCS11 server could be secured with existing secure transport technologies like TLS with server-only authentication where the client would authenticate using his password (provided by the PKCS11 application via the PIN login mechanism). The identity of the user could be provided though a similar mechanism as outlined in the existing PKCS11 standard (in the appendix on multiple PINS and virtual tokens) by appending the user's identity to the token name or out of band through configuration parameters supplied to the local PKCS11 client stub.

Another possible item for future work would be to support delegating credentials to the co-processor using the MyProxy protocol. We could make additional modifications to the MyProxy server so it accepts delegated user credentials by generating keys on the co-processor and has the co-processor sign the delegation certificate request. This would give users the flexibility to store credentials from other CAs, credentials with restricted rights, or other types of credentials in the repository.

6. Summary

In this paper we showed that the usability and security of PKI authentication was undermined by failure to practice appropriate "key hygiene" and by the complexity of certificate request and distribution. The use of a centralized PKI credential store solves many of these problems but poses an attractive target for attacks and may violate CA regulations. We showed that strengthening a credential repository with hardware security devices improves security, provides compliance with CA regulations, and offers additional attractive properties such as performance improvements. The experimental implementation described in this paper is available from the MyProxy website at *http://myproxy.ncsa.uiuc.edu*.

7. Acknowledgements

The authors would like to acknowledge the contributions to this work by Jeevak Kasarkod and Bahaaldin Al-Amood (Virginia Tech) as well as the support for this research from the Virginia Commonwealth Information Security Center (CISC) and the IBM Corporation, specifically Shawn Mullen, Stephen Bade and Kent Yoder of IBM Austin.

This material is based upon work supported by the National Science Foundation under Award No. ANI-02-22571 and the Office of Naval Research under Award No. N00014-03-1-0765.

8. References

[1] M. Carnut, E. Hora, and C. Mattos, "FreeICP.ORG: Free Trusted Certificates by Combining the X.509 and PGP Hierarchy Through a Collaborative Trust Scoring System", *Proceedings of the 2nd Annual PKI Research Workshop*, Gaithersburg, Maryland, April 2003, pp. 13-29.

[2] I. Foster and C. Kesselmanm, "Globus: A Metacomputing Infrastructure Toolkit", *International Journal of Supercomputer Applications*, vol. 11, no. 2, 1997, pp. 115-128.

[3] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit", *Concurrency and Computation: Practice and Experience*, vol. 13, no. 8-9, 2001, pp. 643-662.

[4] R. Sandhu, M. Bellare, and R. Ganesan, "Password-Enabled PKI: Virtual Smart-cards versus Virtual Soft Tokens", *Proceedings of the 1st Annual PKI Research Workshop*, Gaithersburg, Maryland, April 2002, pp. 89-96.

[5] J. Novotny, S. Tuecke, and V. Welch, "An Online Credential Repository for the Grid: MyProxy", *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, IEEE Press, 2001.

[6] J. Steiner, C. Neuman, and J. Schiller, "Kerberos: An Authentication Service for Open Network Systems", *Proc. of the Winter 1988 Usenix Conference*, February, 1988.

[7] C. Neuman and T. Ts'o, "Kerberos: An Authentication Service for Computer Networks", *IEEE Communications*, vol. 32, no. 9, September 1994, pp. 33-38.

[8] N. Itoi, "Secure Coprocessor Integration with Kerberos V5", *Proceedings of the 9th USENIX Security Symposium*, Denver, Colorado, August 2000.

[9] J. Tardo and K. Alagappan, "SPX: Global Authentication Using Public Key Certificates", *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, May 1991, pp. 232-244.

[10] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, "A National-Scale Authentication Infrastructure", *IEEE Computer*, 33(12), December 2000, pp. 60-66.

[11] M. Gasser and E. McDermott, "An Architecture for Practical Delegation in a Distributed System", *IEEE Symposium on Research in Security and Privacy*, May 1990, pp. 20-30.

[12] Ganesan, R., "Yaksha: Augmenting Kerberos with Public Key Cryptography", *Proceedings of the Symposium on Network and Distributed System Security*, Februrary 1995, pp. 132 -143.

[13] X. Wang, "Intrusion Tolerant Password-Enabled PKI", *Proceedings of the 2nd Annual PKI Research Workshop*, Gaithersburg, Maryland, April 2003, pp. 44-53.

[14] G. Sarbari, "Security Characteristics of Cryptographic Mobility Solutions", *Proceedings of the 1st Annual PKI Research Workshop*, Gaithersburg, Maryland, April 2002.

[15] J. Basney, W. Yurcik, R. Bonilla, and A. Slagell, "The Credential Wallet: A Classification of Credential Repositories Highlighting MyProxy", *31st Research Conference on Communication, Information and Internet Policy (TPRC 2003)*, Arlington, Virginia, September 2003.

[16] A. Arsenault and S. Ferrell, "Securely Available Credentials – Requirements", IETF RFC 3157, 2001.

[17] Y. Hsu and S. Seymour, "An Intranet Security Framework Based on Short-Lived Certificates", *IEEE Internet Computing*, vol. 2, no. 2, April 1998, pp. 73-79.

[18] O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman, "Kerberized Credential Translation: A Solution to Web Access Control", *USENIX Security Symposium*, 2001.

[19] L. Zhou, F. B. Schneider, and R. van Renesse, "COCA: A Secure Distributed On-line Certification Authority", *ACM Transactions on Computer Systems,* vol. 20, no. 4, November 2002, pp. 329-368.

[20] P. Gutmann, "Plug-and-Play PKI: A PKI Your Mother Can Use", *Proceedings of the 12th USENIX Security Symposium*, Washington, DC, August 2003, pp 45-68.

[21] RSA Laboratories, "PKCS #11 v2.11: Cryptographic Token Interface Standard", *RSA Security Inc. Public-Key Cryptography Standards (PKCS)*, November 2001.

[22] J. R. Salzer and M. D. Schroeder, "The Protection of Information in Computer Systems", *Proceedings of the IEEE*, Vol 63, No. 9, September 1975, pp. 1278-1308.