Mithril: Adaptable Security for Survivability in Collaborative Computing Sites

Jim Basney, Patrick Flanigan, Jin Heo, Himanshu Khurana, Joe Muggli,
Meenal Pant, Adam Slagell, Von Welch
National Center for Supercomputing Applications (NCSA)
University of Illinois
{jbasney, flans, jinheo, hkhurana, jmuggli, mpant, slagell, vwelch}@ncsa.uiuc.edu

Abstract

Scientific computing sites enable collaborations over the Internet but also face increased risks from malicious parties. Ensuring availability of computing services at these sites is critical for scientists and engineers across the world and requires addressing the challenges of openness, usability, performance, and compatibility. We have developed an approach for survivability of sites using adaptable security mechanisms that address these challenges. Using this approach we have develop tools that tackle immediate threats facing open sites as illustrated in recent events.

1. Introduction

Collaborative scientific computing sites, such as the NRL Center for Computational Science, NSF computing sites (NCSA, SDSC, PSC, NCAR) and similar labs in DOE (e.g. NERSC, LBNL) provide an important service to the scientific and engineering communities. They enable pooling of a variety of distributed resources that provide a number of general purpose and specialized computing services to a large number of geographically distributed users over the Internet.

The growing ubiquity of the Internet allows collaborations that have enabled previously impossible breakthroughs in scientific research, but has also brought increased risks, both in terms of numbers and methods, from malicious parties seeking to subvert computing resources for a variety of reasons. These increases in both the desire for collaboration and threats present security architects with opposing goals: enabling their sites to collaborate with other centers while at the same time increasing security to manage the risks posed by greater threats. Sites respond to these opposing demands with a security architecture that attempts to straddle the line between being open to enable collaboration and being closed to control risks.

Ensuring availability of computing services is critical for users and requires maintaining site security in the presence of unique challenges. First, the end users and their machines pose challenges with openness, usability, and control. The users access the services over the Internet, which implies that the services offered by the sites must be open in nature. The users have limited knowledge of security tools and techniques, which implies that usability of security tools is as important as their effectiveness. The users' machines are off-site, which implies that the vulnerabilities present in these machines are outside the control of site administrators. Second, the nature of computing services at the sites poses challenges of performance and efficiency. These sites provide high-performance computing services to a large number of users that access the services simultaneously. Consequently, site security mechanisms instrumented on the site's compute systems must neither reduce the numbers of users these sites can support nor degrade the performance of the experiments run by the users. Third, the software systems running on the sites pose challenges with compatibility. For the most part the sites develop and employ Unix/Linux based open-source software systems, which are very different, for example, from the software systems used by commercial computing systems. Consequently, site security solutions that are compatible with these software systems may need to be specially developed.

Current site security mechanisms do not sufficiently address these challenges as evidenced by the widespread cyber attack of 2004, collectively referred to as *Incident 216*. In those attacks the adversary compromised a large number of hosts and installed SSH Trojans to harvest usernames and passwords for valid server accounts. In a relatively short time the adversary compromised enough accounts that sites had to either shut down or degrade service considerably in order to recover. These attacks highlight both the need for comprehensive site security mechanisms and specific threats to collaborative sites.

In this work we propose the adoption of adaptable site security mechanisms that ensure survivability of open collaborative sites. This approach maintains the high-level of openness and usability during normal operations but allows a response against attacks by applying security counter-measures and adjusting the level of service accordingly. The approach builds on survivability design principles studied earlier [1][3][5] but specifically focuses on addressing the challenges identified above.

The rest of this paper is organized as follows. In Section 2 we provide an overview of our Mithril approach and solution. In Section 3 we present the core coordination and management system of Mithril. In Section 4 we present tools that we have developed to address major threats to open sites. In Section 5 we review related work and we conclude in Section 6.

2. Adaptable Security for Survivability

The current approach to dealing with threats in collaborative sites is to ascertain a new or increased threat emerging and deploy increased security to meet the threat. Since new security mechanisms take time to integrate and deploy, sites hope the threat emerges slowly, allowing the site to carefully plan the deployment of the new security mechanism and train its users and staff in its use. However, in some cases, such as with Incident 216, a threat can emerge suddenly; in these circumstances sites do not have time to deploy a new technology, but instead either deploy more personnel to increase security (usually redirecting those personnel from other tasks) or isolate themselves from the threat by taking systems or their entire site off the Internet. If this paradigm continues, security will steadily increase until it hits the point where the usability of a site reaches a minimum level of usefulness for its user community. At that point the sites will be trapped, unable to raise security without making themselves unavailable to the community that they were created to serve.

At the heart of this problem is the static nature of site security mechanisms. Site security mechanisms cannot today change in time to respond to these quickly emerging strong threats. Our approach, Mithril, introduces a new paradigm for site security, one based on the notion of survivability. Mithril provides the ability for a site to raise its level of security dynamically in response to these ephemeral threats. This will allow a site to keep an acceptable level of security during normal day-to-day operations, but respond quickly to sudden increased threats. Following the notion of survivability, we will allow this response to take place in a manner such that the increased security and resulting reduced usability will happen in a graceful manner, as opposed to today where the site is stuck between providing service at a set level of security or taking itself off the net.

Mithril's design towards survivability is based on the development of security mechanisms that prevent, detect, and respond to specific threats and are coordinated by a core management system. The result of applying the Mithril solution is that sites will be able to survive during increased threats by making a policy decision to temporarily raise security. This raise will necessarily result in decreased usability, however since this state is temporary, this approach addresses the first challenge of dealing with openness, usability and control with end users. For example, while computing sites have a huge number of users, at any particular point in time there are typically a very small number of users consuming the bulk of the computational resources. Consider the following statistics regarding usage of CPU cycles of the TeraGrid Mercury cluster at NCSA:

- For all time, the top 10 PI's used 85% and top 20 PI's used 95% of the consumed computing cycles.
- For the two weeks prior to February 4th, 2005, the top 10 PI's used 92% and top 20 PI's used 98% of the consumed computing cycles.

These statistics indicate that by even serving just a few dozen users (each PI might map to a handful of project members) during a temporary state of increased security would allow a site to keep a significant percentage of it computational cycles consumed. That is, only a subset of users need to deploy and use advanced security mechanisms and even then, only for short periods of time.

The core management system of Mithril leverages existing intrusion detection systems (IDSs) and requires minimal additional instrumentation on computing sites thereby addressing the second challenge of performance and efficiency. The management system builds on Prelude (*http://www.prelude-ids.org*), which is an open-source IDS management tool that has a distributed architecture with flexible support for instrumentation of sensors on compute systems.

The management system coordinates the prevention, detection, and response mechanisms for the site. To prevent attacks it establishes policies for (1) firewalls, (2) network-based and host-based IDSs, (3) authentication and authorization mechanisms, and (4) (optionally) commonly used applications that prevent adversaries from attacking the site successfully. To detect attacks it includes an alert correlation engine that allows for specification and detection of complex attacks. To respond to attacks it modifies the policies of IDSs, authentication and authorization mechanisms, and applications via cfengine (*http://www.cfengine.org*), which is an open-source adaptive configuration engine. The response mechanisms are collated under simple security levels (normal, high, critical, etc.) that can be triggered via a simple user interface.

In addition to the development of the management system we have focused our work on two specific threats that were highlighted in Incident 216, namely, account compromise via password stealing and email compromise via account and mail server compromise. To address the first threat we have developed response mechanisms that combine One-Time Passwords (OTP) with an SSH remote agent that protects user private keys. To address the second threat we have developed a secure mailing list [4] that encrypts email messages even while they are in transit at the list server. (Mailing lists are commonly used by administrators to coordinate a response to an attack.) Our choice of software systems for development including OpenSSH, Mailman, GnuPG, etc. addresses the third challenge of software compatibility with open collaborative sites.

3. Mithril Core Management System

The core management system of Mithril addresses three primary concerns — detection, correlation and response. More specifically, the three are anomaly detection in the form of alerts, the accurate correlation of triggering events, and the timely and appropriate response to actual attacks. We use a progressive escalation model which acts upon attacks that have been correlated using known anomalies and signatures. This approach provides a flexible and scalable solution for security monitoring and policy enforcement. Policy adjustments can be enforced at the following three levels. First, we can filter events at the Intrusion Detection System (IDS) level by altering configuration files that are used to indicate known attack signatures present on the network or embedded in log files. Some of these events are called triggers, which instantiate correlations based upon basic pattern analysis. Such correlations can be encapsulated so that different techniques and methods can be applied. Ultimately, when a correlation result is positive, we can assume that an attack is in progress or that an actual intrusion has occurred. Given such information, we can decide upon an appropriate response. This may include a reaction to the specific attack and/or may require a system-wide response that enforces accessibility restrictions upon certain resources. The focus of the Mithril core management system has been heuristic accuracy and adaptability. By examining each level in the core system, we will show that these primary concerns are being met using flexible policy guidelines and an overall collaborative architecture.

We chose the well-known open source system Prelude as our IDS. Besides being open source, Prelude also has a highly flexible and scalable architecture that allows the Mithril system to integrate with it easily. Prelude is a hybrid IDS in that it allows sensors to exist on the network itself (NIDS) as well as on the hosts (HIDS). It is also built upon the premise that existing IDS configurations could integrate into such a hybrid framework. It is fairly straightforward to wrap existing third-party sensors so that they communicate with Prelude through an underlying C library communication layer. Prelude is essentially a three-tier system consisting of distributed sensors reporting to distributed managers that in turn report to a centralized manager logging the events into a database. This MySQL database is monitored by the Prelude user interface, which is built using Python; the current version is called Prewikka. We decided to integrate Mithril with the Prelude framework while remaining as independent as possible from the code itself. This allowed us to avoid the inherent problems that are encountered with open source projects such as code revisions and incompatibilities. Mithril is only dependent upon Prelude database revisions and to a lesser extent, user interface incorporation. We decided to build a Python daemon called the 'mithril-engine' that has three distinct, yet interactive functions represented by internal managers. It monitors Prelude event generation for 'interesting' events. These 'interesting' events act as triggers for specific correlations that are analyzed by a 'correlations manager'. Once an attack has been determined due to such correlations, an 'agent-manager' employs distributed agents to alter system configurations accordingly. This response is dependent upon the current security policies, which are managed by a 'policy manager'. All three of these managers communicate with the Mithril console, which is an extension of the Prelude user interface. This allows the user to alter correlation criteria and design, change policy configurations dynamically and to direct agents for specific responses to attacks.

Event correlations are performed by the 'correlations manager' layer of the 'mithril-engine'. The correlation process has been designed as an encapsulated engine that receives discrete alerts as input and generates specific attack correlations as output. Each correlation type can combine variables such as alert types, time and rate indications, as well as packet source and target information. Different types of correlations are described by filters, which are built using criteria that capture triggering events. These filters are stored in the Mithril database and are used to maintain state for ongoing analysis. Currently, we have implemented two correlation types based on the "alert aggregation" and "focused analysis" correlation utilities of Ning *et al.* [8] by extending the simple grouping and filtering capabilities of Prelude. (In the future, we will seek to integrate other more advanced correlation techniques such as clustering and frequency analysis to detect more advanced attacks). These two correlation filters allow detection of brute force attempts on SSH logons, both failed and successful, and general scanning attacks.

In a typical brute force correlation, alerts are picked up by the Prelude sensors and are then sorted by correlation filters to determine which alerts are considered triggers. In a brute force attack, these triggers are failed attempts on an SSH logon. These alerts are generated by analyzing the authorization log file using the Prelude log file analyzer (Prelude LML). Once a correlation filter has been 'triggered', an active correlation is instantiated. This correlation is built using specific criteria and waits for prerequisites to be filled. In this case, we are watching for a rate of attack. If a certain number of alerts are generated for login attempts within a certain time period, the active correlation reaches its threshold. At this point, it is considered an attack.

Response to an attack is undertaken via enforcement of security policies that are managed by the 'policy-manager' portion of the 'mithril-engine'. It is designed so that policies can be customized dynamically and can be enforced either automatically or by human intervention. These policies are stored in the Mithril database and exist as specific attack response guidelines as well as multi-level system wide responses. The Mithril policy architecture is organized as a series of general levels that can be individually customized according to need. This allows increases or decreases in security levels with 'broad strokes' instead of addressing specific restrictions wherein certain limitations may be overlooked. Because of this, Mithril level adjustments enable adaptability and rapid response mechanisms to enforce policy changes resulting in a highly survivable system. These policies are then enforced by invoking agents such as cfengine, which are able to specifically set system-wide needs. For example, in responding to the attempted brute force attack we raise the security level to 'high, which then requires the use of OTP authentication at servers.

4. Addressing Threats with Mithril

In this section we discuss the design and implementation of our integrated SSH remote agent and one-time password solution as well as our secure mailing list solution.

4.1. SSH Remote Agent

Remote login is a core service provided by many collaborative computing sites. Allowing direct login access to a site's computing platforms is often essential for providing a general-purpose service and allowing users to compile and tune their own codes for what are often specialized computing platforms. Single sign-on eases the integration of remote computing platforms into the user's environment, avoids the need for users to remember multiple passwords for different sites and avoids the need for users to respond to an ongoing series of password prompts during distributed computing sessions.

SSH public key authentication [10] is a widely used standard for secure remote login with single sign-on. Users associate their public keys with their remote accounts, allowing them to authenticate with their corresponding private keys for secure remote login. Users are encouraged to protect their private keys by encrypting them with a passphrase. Then, to provide single sign-on, users can unlock/decrypt and load their private keys into an SSH agent at the start of their session, and the SSH client will forward authentication challenges to the agent for signing. Furthermore, as illustrated in Figure 1, the agent connection can be forwarded over the SSH transport layer protocol, allowing users to initiate multi-hop remote logins with single sign-on. Forwarding all authentication challenges to the agent minimizes exposure of the user's unencrypted private key during the session, as the key is held in the agent's private memory and never written unencrypted to disk or forwarded over the network.

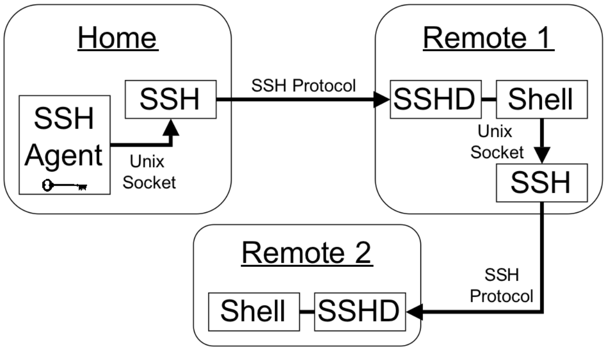

Figure 1: SSH Authentication Agent Forwarding

However, the user's private key remains vulnerable to attack on the user's home machine. Keystroke loggers and Trojan SSH clients can intercept the user's passphrase to decrypt the private key. Private keys protected by poorly chosen passphrases are vulnerable to dictionary attacks. Furthermore, some users (for convenience) avoid using the SSH agent altogether by storing their private keys unencrypted on disk, where they can be stolen if the user's account is compromised. The convenience for users of single sign-on unfortunately also provides convenience for attackers in propagating their attacks using compromised private keys. A recent study found that over 60% of users' SSH private keys were stored unencrypted and described how attacks were propagated via SSH in Incident 216 [11]

Unfortunately, when an SSH private key is compromised, it is difficult to contain the breach. A user's public key may be associated with accounts at sites across the Internet. In contrast to keys in PKI certificates, these keys have no lifetime restrictions or revocation capability. The compromised public keys must be disassociated with each remote account.

To address this vulnerability, we have developed an SSH Remote Agent illustrated in Figure 2. Rather than storing SSH private keys on a user's desktop, where they are vulnerable to many attacks, the SSH Remote Agent provides a mechanism for generating and storing keys on a dedicated, secured key server. By running the *ssh-remote-agent* program, users authenticate and establish a secure connection to the key server at the start of their session, and all SSH public key authentication challenges are forwarded to the key server via the SSH agent forwarding protocol, so the user private keys need never leave the key server. Our approach is similar to the MyProxy systems [12], which protect long-lived private keys in a dedicated repository and issue short-lived X.509 proxy certificates [13] for single sign-on authentication.

.

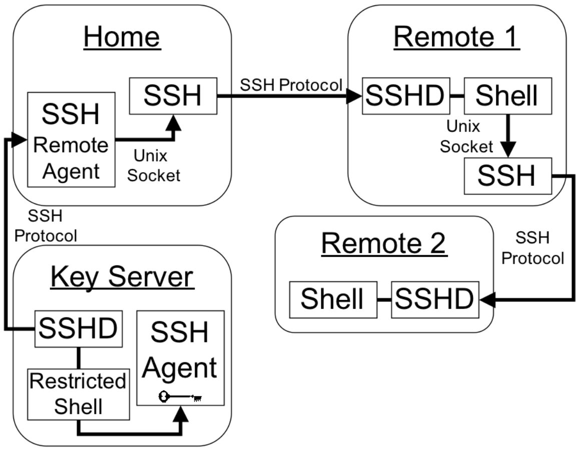

Figure 2: SSH Remote Agent

The security of the key server is the foundation of the system. The dedicated key server has only one open network port, to an SSH server which provides access to a restricted shell that implements the limited set of SSH agent operations. Thus, there are limited vectors for private key attack. Users themselves never have direct access to the private keys, so a user account compromise does not compromise the associated private keys. Using SSH to access the key server provides many strong authentication options, including Kerberos and one-time passwords.

4.2. Secure Mailing Lists

The security system of a site comprises human administrators, machines with sensors that detect attacks, and machines with actuators that respond to attacks. These entities need to communicate with each other in order to keep the site secure and email is the most common form of such communications. Specifically, mailing lists are setup that allow devices to send messages to a set of administrators and allows the administrators to discuss and coordinate a response. In Incident 216 the email system was attacked (both as the client side and at the server side) with a specific goal of eavesdropping on discussions on the coordinated response. To address this threat we have developed SELS, a Secure Email List Service [2][4] that provides confidentiality, integrity, and authentication for messages exchanged over a mailing list.

Encryption and signing with digital certificates as done in PGP and S/MIME for secure two party email suffices to address the client-side threat. However, extending such solutions for mailing lists would expose email plaintext at the list server. SELS uses proxy encryption techniques to ensure that messages remain encrypted as the list server processes them for forwarding to the list and, consequently, succeeds in addressing the server-side threat. In addition to addressing this threat, we envision that in the future secure emails may be used to send commands and configuration updates to actuators and not just for receiving event notifications from sensors. Furthermore, in order to address the software compatibility challenge, SELS uses (1) out-of-the-box GnuPG for client-side security to ensure compatibility with a large number of email clients and (2) a plugin for the Mailman list server software to ensure ease of deployment and use in sites along with a modified GnuPG library that provides proxy encryption capabilities.

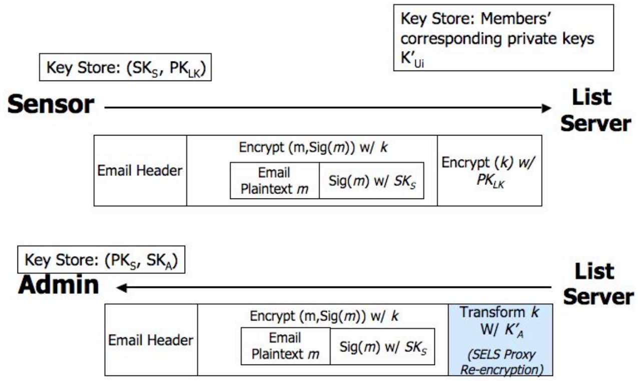

Figure 3: Sending Secure Emails with SELS

Protocol and implementation details of SELS are provided in [2]. We now give a brief overview of how SELS ensures security of emails exchanged in a list as illustrated in Figure 3. Here a mailing list has been setup to include a sensor and an administrator among other list members. All list members have received a list public encryption key, PK_{LK}, signature keys (e.g., SK_S for the sensor), signature verification keys (e.g., PK_S for the sensor), and decryption keys (e.g., SK_A for the administrator). In addition the list server has a corresponding private key for every member that is uses for proxy transformations (e.g., K'_A for the administrator).

In order to send a message to the list, the sensor signs the email message and then encrypts the signed message with a randomly generated symmetric key *k*. The sensor then encrypts *k* with the list public key and sends this message to the list server. The list server then processes the encrypted key part of the message separately for every recipient by applying proxy transformation techniques using that member's corresponding private key. The result is then forwarded to the administrator who can then use his decryption key

(SK_A) to decrypt the message and the signature verification key to verify the sensor's signature.

5. Related Work

Survivability of distributed systems, services, and applications has been a major focus of attention lately. Moore and Ellison [6] propose a general framework for survivable architectures. Keromytis *et al.* [3] have developed Saber, which integrates tools for denial-of-service prevention, process migration, software patching, and command and control to ensure service survivability. Knight *et al.* [5] have developed Willow, which provides the architecture and tools for coordination and control between various components of a large distributed application for survivability. Atighetchi *et al.* [1] have developed APOD, which integrates tools for firewalls, intrusion detection, and response via bandwidth reservation and traffic shaping to ensure system survivability. Our work builds on the principles of survivability studied by these works. However, the tools developed by these works were not used because they are either not mature enough in terms of advanced testing or not compatible with software systems running on open collaborative sites.

On the commercial site several enterprise security tools have been developed; e.g., SSH Tectia, ArcSight's Enterprise Security Manager, Symantec Enterprise Security Manager. However, these solutions are not compatible with software systems running on open collaborative sites.

6. Conclusion

We have described Mithril, which is a system that ensures survivability of open collaborative sites using adaptable security mechanisms. The core management system of Mithril coordinates the prevention, detection, and response mechanisms while specific tools such as the openssh-remote-agent and secure mailing lists address specific threats to open sites. In the near future we will deploy and test our system's effectiveness on NCSA's compute infrastructure. We will also continue to develop and integrate tools that address foreseeable threats to open computing sites.

7. References

[1] M. Atighetchi, P. Pal, C. Jones, P. Rubel, R. Schantz, J. Loyall, and J. Zinky, "Building Auto-Adaptive Distributed Applications: The QuO-APOD Experience", in Proceedings of the 3rd International Workshop on Distributed Auto-adaptive and Reconfigurable Systems, in conjunction with the 23rd International Conference on Distributed Computing Systems, Providence, Rhode Island, USA. May 19-22, 2003.

[2] J. Heo and H. Khurana, "Requirements and Solutions for Secure Mailing Lists", Technical Report, NCSA, February 2006.

[3] A. D. Keromytis, J. Parekh, P. N. Gross, G. Kaiser, V. Misra, J. Nieh, D. Rubenstein, and S. Stolfo, "A Holistic Approach to Service Survivability", Proceedings of the 2003 ACM Workshop on Survivable and Self-Regenerative Systems, Fairfax, VA, October 31, 2003, pp. 11-22.

[4] H. Khurana, A. Slagell, and R. Bonilla, "SELS: A Secure E-mail List Service", in proceedings of the Security Track of the ACM Symposium on Applied Computing (SAC), March 2005.

[5] J. C. Knight, D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, P. Devanbu, M. Gertz, "The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications", in Intrusion Tolerance Workshop, DSN-2002 The International Conference on Dependable Systems and Networks, Washington DC, June 2002.

[6] A. P. Moore and R. J. Ellison, "TRIAD: A Framework for Survivability Architecting", in Proceedings of the Workshop on Survivable and Self-Regenerative Systems, 10th ACM Conference on Computer and Communications Security. Washington, D.C., October 31, 2003.

[7] P. Ning, Y. Cui, D. Reeves, and D. Xu, "Tools and Techniques for Analyzing Intrusion Alerts," in *ACM Transactions on Information and System Security*, Vol. 7, No. 2, pages 273--318, May 2004.

[8] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", *Computer Networks*, 31(23-24), pp. 2435-2463, 14 Dec. 1999.

[9] M. Roesch, "Snort – lightweight intrusion detection for networks", in Proceedings of USENIX Lisa, 1999.

[10] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol", IETF RFC 4252 (Standards Track), January 2006.

[11] S. Schechter, J. Jung, W. Stockwell, and C. McLain, "Inoculating SSH Against Address Harvesting", Network and Distributed System Security Symposium (NDSS), February 2006.

[12] J. Basney, M. Humphrey, and V. Welch, "The MyProxy Online Credential Repository," *Software: Practice and Experience*, Volume 35, Issue 9, July 2005, pages 801-816.

[13] S. Tuecke, V. Welch, D. Engert, L. Perlman, and M. Thompson, "Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile", IETF RFC 3820, June 2004.