Dynamic Policy Enforcement in a Network Environment

Brandon Pollet, Matthew Butler, and John Hale
Center for Information Security
University of Tulsa

Abstract

The challenge of maintaining a secure and usable system in a hostile environment makes self-defending systems an attractive addition to network security. Using the conditional policy extensions in the Security Enhanced Linux (SELinux) policy language, it is now possible to dynamically adjust a SELinux system's security policy based on its environment. The Dynamic Policy Enforcement Agent (DPEA) utilizes the CLIPS expert system and the conditional policy construct to activate and deactivate portions of the SELinux policy based on DPEA alerts, SELinux audit logs, system logs, as well as to alert other hosts on the network of attacks.

1 Introduction

The security of modern computer systems depends on a number of environmental factors: services, operating system, patch level, and, perhaps most importantly, configuration [1]. Computing services can be as simple as web browsing or text editing, or as complicated as distributed decryption or satellite navigation. Regardless of what services a system may provide, some level of assurance for security is generally required. In a large enterprise with a high requirement of security, it can be difficult for administrators to provide the necessary attention to ensure a reasonable level of security.

This motivates the development of systems that can work to ensure their own security. These *self-defending systems* could be configured to respond to security threats themselves, without the constant supervision from a human administrator. When a self-defending system detects an attack, or the threat of an attack, it should adjust its configuration to defend against it, increase its logging, or shutdown services until the threat has passed.

A true self-defending system should be able to protect itself from both external and internal threats which is a feature that many of todays systems lack. One reason for this is that most operating systems do not provide the correct resources for a system to monitor and control all aspects of its operation. However, with Security Enhanced Linux (SELinux), and other operating systems that employ Mandatory Access Control, it is possible to construct systems that are more configurable from a security standpoint. These security conscious systems, coupled with an agent program capable of making informed security decisions, provide a solid foundation for self-defending systems.

The Dynamic Policy Enforcement Agent (DPEA) is a tool that provides a new mechanism for using the security policy of a SELinux system to deflect attacks. By monitoring its environment for internal and external threats and using SELinux booleans coupled with the conditional policy system, DPEA is able to modify the security level of a system to respond to dynamic threat environments. Furthemore, DPEA requires very little access to critical resources and no direct access to the system's security policy, minimizing the possibility that DPEA could be used to compromise the system. DPEA is an example of a *self-defending system* that can directly respond to both internal and external security threats without constant oversight from a system administrator.

2 Background

In early 2004, Tresys [4] augmented the SELinux policy system by introducing the Conditional Policy Language Extensions [3]. This scheme allows runtime modification of the security policy without requiring the usual reload of the policy from source files. Conditional policy provides boolean variables and expressions, which make it possible to define sections of the policy that are conditionally applied at runtime [3]. The ability to modify booleans in SELinux policy has made it possible to create multiple policy levels that can be activated by the system administrator depending on the current system environment.

3 Design and Implementation

The Dynamic Policy Enforcement Agent (DPEA) is a program that utilizes the structured security system of SELinux to make run-time changes to the security stance of a system. The DPEA allows administrators to specify portions of policy to activate or deactivate in the event of a security compromise and then run the agent to watch for such compromises and take the appropriate action. This allows security changes to take place in the administrator's absence while restricting changes to pre-approved policy.

3.1 Design Principles

The design of DPEA is based around two important principles: separation of DPEA from the actual SELinux policy and scalability in a networked environment.

The separation of DPEA from the active policy is a critical issue when it comes to the security of the system. Allowing DPEA to have run-time access to the security policy of the system would also give it the ability to disable the safeguards provided by the SELinux security server, making it a prime target of attack. This vulnerability is avoided, however, by utilizing conditional policy.

The conditional policy extensions allow an administrator to embed conditional expressions into the SELinux policy file. Through the use of these conditional expressions, a system administrator can define different security postures that can be transitioned to depending on the situation. Since all security policy decisions are made by an administrator prior to the activation of the agent program, the administrator can validate each security posture through the use of policy verification and analysis tools.

The second design principle for DPEA is scalability in networked environments. As it stands, DPEA requires a substantial amount of initial configuration. To perform this initial configuration for a sizable network of SELinux hosts would most likely prove unwieldy at best. To mitigate this burden, DPEA uses a system of configuration profiles tailored to specific types of hosts. Another opportunity in a network environment is to allow hosts to send security alerts to peers on the network. This would give hosts advanced knowledge of eminent attacks.

3.2 Software Architecture

The DPEA has access to only a small subset of its system. This subset includes the system logs, the DPEA rule-set, the necessary ports required to send and receive alerts, and the SELinux boolean files.

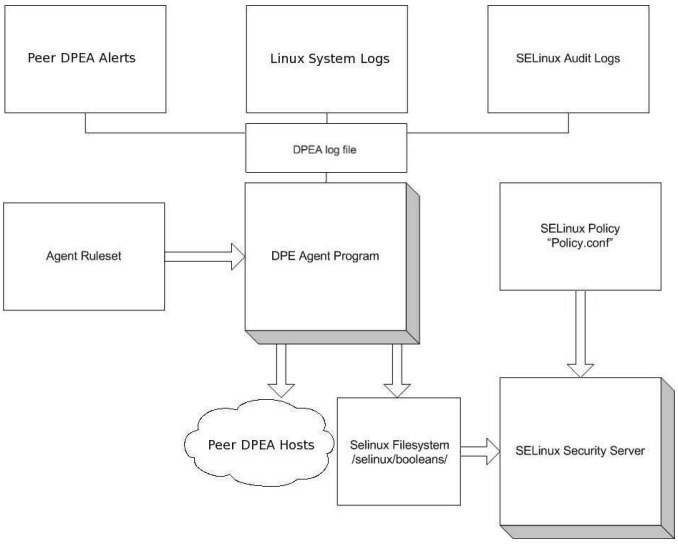

Figure 1: DPEA Software Architecture.

For DPEA to fulfill its mandate as a program that enables self-defending systems it must be aware of threats that come from inside the system as well as from the outside world. To accomplish this, DPEA bases its decisions on the standard system logs and peer DPEA alerts. As illustrated in Figure 1, DPEA gathers host event information through a DPEA-specific log file. Logs from the SELinux security server and various other services may be directed to this file through system log administration. This allows input from any program whose logs may be of interest in making security decisions. Alerts from peer DPEAs are received by a stand-alone server, which logs the source and content of each alert. Connections between the this server and the various DPEA serving as clients are encrypted via SSL. Malformed alerts are logged as such to allow the detection of possible DoS attacks or spoofing attempts.

The log file is periodically scanned by DPEA and new entries are asserted as facts in the DPEA's CLIPS Knowledge Base, which is then scanned for specified events in the rule-set. When a match is found, DPEA either makes the changes to the boolean files or sends alerts per the rule-set for the current event. The modification of the boolean files is the only interaction that DPEA needs to have with any portion of the SELinux policy system.

The interaction between the boolean policy files and the SELinux security policy occurs when the system security server reaches a conditional branch in the running policy. When a branch is encountered the boolean policy files are read for their value of one or zero. If the value is a one, then the policy inside the conditional branch is executed; if the value is zero, then the policy inside the conditional is skipped over.

3.3 DPEA Network Configuration

Currently, a large amount of initial configuration must be done to take full advantage of the DPEA's security features. This includes configuring the system's logging system, modifying the system's SELinux policy to correspond with the Booleans accessed by DPEA, and writing the rule-set for the agent program. While manually installing a custom policy for use with DPEA is the best option, a system of rule-set profiles based on the host's role on the network is used to mitigate the cost of deploying DPEA across a network.

As the types of services a host provides strongly influences the types of rules its rule-set will contain, it makes sense to create a common rule-set for hosts providing similar services. This allows us to configure types of hosts at a time, rather than each separately. Local custom configurations to allow for special needs of a single host will of course be supported as well. The DPEA also includes RSS (Real Simple Syndication) functionality [2] to provide up-to-date rule-set profiles and host lists as needed from a central host.

During installation, the role of host is defined, the appropriate rule-set profile is downloaded and installed and the host is registered with the central server based on the services it provides. Registering the services provided allows DPEA alerts to be sent to specific types of hosts.

4 Applications

The functionality of DPEA can be used to increase the security of a SELinux system in many different ways. Both internal and external threats can be detected, analyzed, and countered by the DPEA system. This allows DPEA to respond to both inappropriate access attempts from internal users and against an outside attackers attempts to exploit a vulnerable web service. As an example of DPEA's capabilities at system defense the following subsections illustrate scenarios in which DPEA can be used to prevent security breaches.

4.1 Authorization Failure

The first example is an authorization failure system event, which appears in the system logs when a user attempts to escalate their privileges to the super user on a system and enters an incorrect password. The system administrator has created a rule which states that any user who attempts to become a super user and initiates an authentication failure message should not be allowed to escalate their privileges until the situation can be reviewed. When DPEA scans recent log entries this rule will match the authentication failure log message and initiate the cooresponding action. This action modifies the *user_disable_trans* boolean variable which disallows any transitions from the generic user role to the super user role. A similar rule could be made to catch SSH brute force attacks.

This example illustrates an important aspect of DPEA usage, sound policy and DPEA configuration. While this example was constructed to showcase the power of dynamic policy enforcement, it also shows that it is possible for an attacker to create a denial of service attack on the ability to transition to *root*. If an attacker were aware of a system's DPEA enacted security policy they could construct attacks that use the system's self-defense strategy against itself. Therefore, it is paramount that system administrators fully understand the implications of the policies they allow DPEA to modify. Rules for policy modification must also be well written. It is possible that a rule as described in this example would be too easily triggered to be useable in a production system, and may make it overly difficult for the system administrator to fix the problem. A rule-set that looked for a threshold of authentication failures along with configuring DPEA to reset its knowledge base on a regular schedule could be one way to solve these issues.

4.2 Malicious Use of Services

This example uses the conjunctive properties available in DPEA rules and shows how to modify the running services on a system through the use of SELinux booleans. The system is running two services, HTTP and SSH. The primary purpose of this machine is to serve webpages over http but it also allows incoming ssh connections for administrative purposes. The policy for this system is written to support two different levels of service, *full_service* and *primary_service*. In the *full_service* mode both the web server and SSH are accessible from the network, while in *primary_service* mode only the web server is allowed to execute. These two service levels allow a quick switch from an baseline policy to a hardened security stance. This event is generated by a port-scan detection originating from an intrusion detection system. The system administrator has constructed a compound rule that states if a *port_scan* is detected and if the boolean *inet_full_service* is true, then set the boolean to false to shutdown any extra services. This effectively seals the computer from any attempts a hacker might make to infiltrate the system through the SSH service. When activated by the IDS log entry, DPEA checks the value of the *inet_full_service* fact in its knowledge base to see if a change needs to be made. Because this is a conjunctive rule the action will only take place if both conditions are met.

4.3 Network Intrusion Response

In the previous example, it was assumed that the host running DPEA was also running an intrusion detection system. This arrangement is rarely practical due to the amount of processing power required to analyze network traffic without dropping packets. While it would be possible to forward the IDS logs to all DPEA-enabled hosts, this would be also be impractical for more than a handful of hosts. Instead, this example uses a network topology consisting of multiple NIDS sensors and a central server to which IDS logs are forwarded through secured channels.

The central server is using a rule-set profile written expressly for alerting other hosts of attacks. One of these rules is to alert all servers offering secure shell that a port scan has been detected. When a port scan is logged, DPEA will send an alert to each host listed as providing SSH. Each host's alert server will then log this alert, allowing it to be added to that host's Knowledge Base and addressed as described in the host's rule-set. Another possibility would be to allow the central server to request the offending host to increase logging or restrict certain priviledges, provided the offending host was running a DPEA and the necessary rules were implemented.

5 Conclusion

DPEA is a flexible agent-based solution to create policy that can dyanmically respond to system threats. By mediating run-time changes of the system security policy, dynamic policy enforcement allows a system to respond to changes in its networked and internal environments without the presence of an administrator. Several issues remain to be resolved, including methods to lend analytical support for configuration management, strategies for handling rule conflicts, and techniques to prevent the subversion and use of DPEA itself as an attack vector. Even so, the combination of user configurable rules, dynamic policy modification, and real-time threat defense make DPEA an important addition to the security infrastructure of SELinux.

6 Acknowledgments

This research was supported by MPO Contract MDA 904-02-R-0039.

References

[1] LOSCOCCO, P. A., SMALLEY, S. D., MUCKELBAUER, P. A., TAYLOR, R. C., TURNER, S. J., AND FARRELL, J. F. The inevitability of failure: The flawed assumption of security in modern computing environments. In *Proceedings of 21st National Information Systems Security Conference* (1998), pp. 303–314.

[2] NETSCAPE. Rss (file format). http://en.wikipedia.org/wiki/RSS.

[3] NSA. Selinux conditional policy readme, September 2004.

[4] TRESYS. Tresys technology. http://www.tresys.com, 2004.