Managing Heterogeneous Network Environments Using an Extensible Policy Framework

Lawrence Teo
University of North Carolina at Charlotte
College of Computing and Informatics
lcteo@uncc.edu

Gail-Joon Ahn †
University of North Carolina at Charlotte
College of Computing and Informatics
gahn@uncc.edu

ABSTRACT

Security policy management is critical to meet organizational needs and reduce potential risks because almost every organization depends on computer networks and the Internet for their daily operations. It is therefore important to specify and enforce security policies effectively. However, as organizations grow, so do their networks increasing the difficulty of deploying a security policy, especially across heterogeneous systems. In this paper, we introduce a policy framework called Chameleos-x which is designed to enforce security policies consistently across security-aware systems with network services–primarily operating systems, firewalls, and intrusion detection systems. Throughout this paper, we focus on the design and architecture of Chameleos-x and demonstrate how our policy framework helps organizations implement security policies in changing, diversity-rich environments. We also describe our experimentation of Chameleos-x to demonstrate the feasibility of the proposed approach.

1. INTRODUCTION

Businesses and organizations depend heavily on computer networks and information systems for their daily operations. Due to this ever-increasing reliance on computer systems, it is critical for organizations to implement a carefully-designed security policy for their networks and information systems.

When organizations grow, so do their computer networks and information systems. This growth tends to introduce diversity and heterogeneity into the network, especially as new operating systems, network devices, and security technologies are adopted. As the number and types of systems increase, the security of the organizational networks is affected in two major ways: (1) the difficulty of designing and enforcing a security policy that works consistently across different systems is significantly multiplied; and (2) the ability to maintain the consistency of the policy in the face of changing organizational requirements

In this paper, we argue that a practical, system-driven approach should be used to address the problem of enforcing security policies consistently in a changing, diversity-rich environment. We propose a solution in the form of a system-driven policy framework called Chameleos-x, which consists of both a policy specification language and a policy enforcement architecture. The Chameleos-x framework is specially designed to facilitate the management of consistent security policies in heterogeneous environments. The objective of Chameleos-x is to enforce security policies consistently on different security-aware systems as well as operating systems. In this paper, security-aware systems include operating systems, firewalls, and intrusion detection systems (IDSs) – those are responsible for enforcing any part of the organizational security policy. Our prior works clearly demonstrated how we could develop a policy language to specify access control policies across different operating systems [4, 5].

This paper is organized as follows. Section 2 presents the objectives, design, and architecture of Chameleos-x followed by a discussion of our experiments and results in Section 3. Section 4 concludes the paper including our ongoing and future work.

2. POLICY FRAMEWORK: Chameleos-x

The Chameleos-x policy framework has two major components: a policy specification language and a policy enforcement architecture. As a family of languages and architectures, Chameleos-x supports different kinds of systems – currently it works with operating systems, firewalls, and IDSs (Figure 1). The advantages of implementing a single language for many security-aware systems are manifold. Having a single language would provide a common syntax for administrators to implement various policies. There is no need to relearn the syntax for different systems, thus presenting a convenient way for the administrator to specify multiple system policies. This is especially true when the evaluation of different systems is taking place. Also, if there are similar systems, we do not need to convert the policies from one system to the other.

2.1 Approach

We first discuss the approach that was used to design the Chameleos-x policy framework. We present the two key

†All correspondence should be addressed to: Dr. Gail-Joon Ahn, Software and Information Systems Department, College of Computing and Informatics, University of North Carolina at Charlotte, 9201 University City Blvd., Charlotte, NC 28223; email:gahn@uncc.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ASIACCS'07, March 20-22, 2007, Singapore.
Copyright 2007 ACM 1-59593-574-6/07/0003.

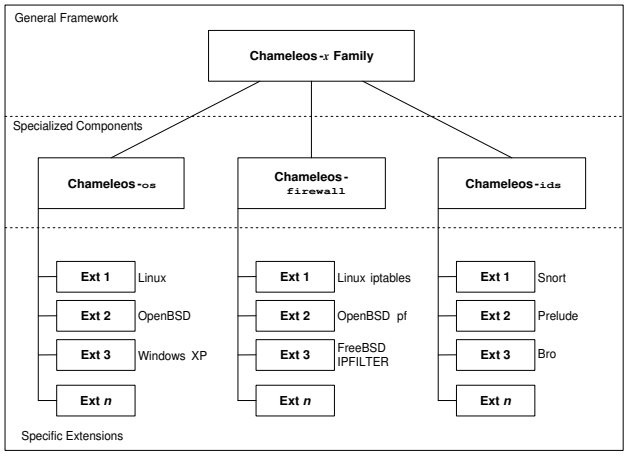

Figure 1: The Chameleos-x framework.

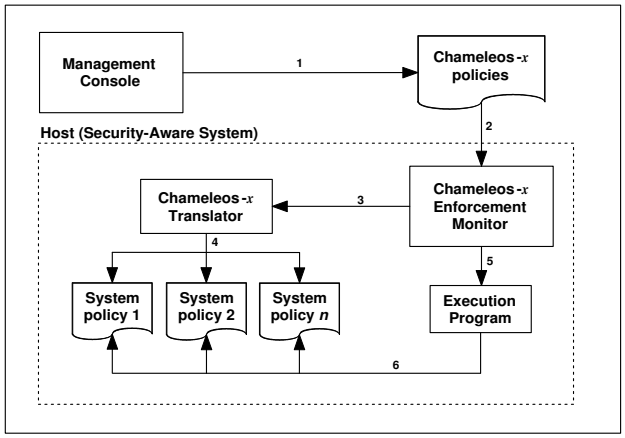

Figure 2: The Chameleos-x architecture.

decisions that we made in the design of the language, and how they affected the development of our framework.

Firstly, we have to decide whether to develop an extension of an existing similar policy framework or develop a new framework altogether. Unlike other frameworks, a key differentiator in the Chameleos-x is that it integrates with a risk-based network management architecture [6], thus it uses a different paradigm compared to other frameworks. This difference alone warrants the necessity to design a new framework. Another benefit of creating the framework afresh is that it helps us design all the components in our policy framework while achieving completeness and consistency of our approach.

Secondly, some have suggested that the bottom-up approach results in an inflexible framework that is too specific to the underlying systems; for Chameleos-x, however, we believe that developing for actual systems (the bottom-up approach) would be more beneficial. We must reiterate that Chameleos-x is designed to be used in the real world and is not merely a theoretical exercise. In that vein, the language component in Chameleos-x is comparable to programming languages like C and C++. Like Chameleos-x, those programming languages were designed using the bottom-up evolutionary approach. Though their design may not be very elegant, they are proven to be flexible, where they have been used to implement many kinds of solutions. Thus, they enjoyed widespread use in industry for decades.

These decisions led us to adopt an evolutionary design model for Chameleos-x. Using this evolutionary model, we believe we are able to support the specific features of each system more effectively.

2.2 Language and Architecture

The Chameleos-x policy framework includes a language component that is used for policy specification. It is intended to cover many notions that are used to specify policies, including basic access control concepts. We have developed the basic grammar of the Chameleos-x language in Extended BNF (EBNF). Due to space limitations, we omit the EBNF grammar specification in this paper.

The three major components of the Chameleos-x architecture are the Management Console, Translator, and Enforcement Monitor. The Management Console is a central management interface operated by the evaluator. It is used to "push" Chameleos-x policies to various hosts that are running the Chameleos-x Enforcement Monitor. The Management Console also specifies which operation mode should be used in each session. The Chameleos-x Enforcement Monitor is a daemon that runs continually in the background on systems that are part of the Chameleos-x framework (that is, the servers, firewalls, and IDSs). Its responsibility is to receive the Chameleos-x policy from the Management Console, and apply it on its host system. To do so, the Chameleos-x policy would have to be translated. This translation process is done by the Chameleos-x Translator, which is used to convert the Chameleos-x policy into one or more system-specific policies. The Translator resides together with the Chameleos-x Enforcement Monitor (it could either be part of the Monitor, or a separate entity that is invoked by the Monitor). Each Chameleos-x variant would have its own Translator. For instance, if we are working with the Snort IDS, the Chameleos-`ids` Translator will convert the Chameleos-`ids` policy into a Snort configuration file.

The layout of the components in the Chameleos-x architecture is shown in Figure 2. The detailed procedures between components are described as a workflow as follows (the numbers in Figure 2 relates to this description):

1. The evaluator composes one or more Chameleos-x policies and uses the Management Console to send them to the Chameleos-x Enforcement Monitor on a host.
2. The Enforcement Monitor receives the policies.
3. The Chameleos-x Enforcement Monitor invokes the Translator to translate the policies it just received into system-specific policies.
4. The Translator performs the translation process. For instance, a Chameleos-`firewall` policy could be translated into a Linux *iptables* ruleset, while a Chameleos-`ids` policy could be translated into a Snort configuration file. The translation could result in one or more system policies, depending on the requirements of the specific systems.
5. After the translation is done, the Enforcement Monitor invokes the execution program (say, Snort in the case of Chameleos-`ids`).

6. The execution program loads and applies the translated policies on the host.

3. EXPERIMENTS AND RESULTS

Our experiments were designed with two objectives: (1) to test the translation process of each Chameleos-x variant, and (2) to test the enforcement/execution process of each Chameleos-x variant. To perform these experiments, we first designed and implemented a test network consisting of six machines, each of which plays a single role: a firewall, an IDS, a server, a "legit" machine that generates good traffic, an "attacker" machine that generates bad traffic, and the management console. Chameleos-x Enforcement Monitors were installed on the firewall, IDS, and server. The management console's responsibility is to push Chameleos-x policies to the Monitors. The Monitor is in charge of translating the Chameleos-x policy from the management console into the correct system policy for its host.

To fulfill the two experimental objectives, we use two "configuration suites" for testing each variant. A configuration suite would consist of a specific firewall, IDS, and server operating system and associated servers (such as the HTTP and FTP servers). Additionally, in order to make the experiment more accurate, we would have to make sure each suite is heterogeneous and different from the other. Configuration Suite 1 consists of a firewall running OpenBSD with the *pf* firewalling subsystem, an IDS running Snort, and a Linux server that is geared to run Apache and vsftpd as its web and FTP servers respectively. Configuration Suite 2 comprises a Linux firewall with *iptables*, the Snort IDS, and a FreeBSD server configured to run thttpd and ftpd. We also used a management console (running NetBSD) with the IP address 172.16.0.2 to push Chameleos-x policies to the machines in each suite.

The translated policies showed consistent behavior in both Configuration Suites 1 and 2, even though the same original Chameleos-x policies were used without changes in each suite. In addition, the translated policies implemented certain features using the specific facilities offered by each target system. For example, groups were defined differently in *pf* and *iptables*, but the end behavior was consistent.

These favorable results show that a practical and system-driven policy framework can be used to perform effective evaluation of a network in a flexible and extensible manner. It also firmly indicates that our policy framework could successfully integrate a simple but powerful declarative language with an enforcement architecture. The results also demonstrate that Chameleos-x, with its system- and platform-independent nature, is indeed capable of facilitating security policy management for heterogeneous environments, as represented by the consistent behavior exhibited by the multiple kinds of systems in Configuration Suites 1 and 2. These objectives were achieved by designing the framework discussed in Section 2.

4. CONCLUSION

We have presented the design of Chameleos-x, a practical and system-driven policy framework that can be used to facilitate the management of security policies in heterogeneous environments effectively. The core strength of Chameleos-x is its ability to specify and enforce security policies consistently across a diverse range of security-aware systems, such as operating systems, firewalls, and intrusion detection systems. Chameleos-x is also designed to assist system and network developers in the configuration and evaluation of these systems for conformance to security policies. Our experiments confirmed that the Chameleos-x policy framework is sufficiently flexible and extensible to deploy security policies effectively across multiple security-aware systems.

We strongly believe Chameleos-x would be very beneficial to organizations, especially those with large and heterogeneous information networks. Based on the promising results obtained through these experiments, we would work on new components for the Chameleos-x policy framework. Most of these new components would be part of the Chameleos-x Translator. These components include a syntax checker, analyzer, and reverse translator. The *syntax checker* would serve as the foundation for all syntax checking requirements in the other components. The *analyzer* would be used to analyze a Chameleos-x policy for conflicts and ambiguities. The analyzer would have to take constraints [1] and conflict resolution techniques [2] into account, especially for complex systems like SELinux [3]. The *reverse translator*'s role is to translate a system-specific policy into a Chameleos-x policy.

Acknowledgements

This work was supported, in part, by funds provided by National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).

5. REFERENCES

[1] Trent Jaeger. On the increasing importance of constraints. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pages 33–42, Fairfax, VA, October 1999.

[2] Trent Jaeger, Antony Edwards, and Xiaolan Zhang. Managing access control policies using access control spaces. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 3–12, Monterey, CA, June 2002.

[3] Trent Jaeger, Reiner Sailer, and Xiaolan Zhang. Resolving constraint conflicts. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, IBM T.J. Watson Research Center, Yorktown Heights, NY, June 2004.

[4] Lawrence Teo and Gail-Joon Ahn. Towards the specification of access control policies on multiple operating systems. In *Proceedings of the 5th IEEE Workshop on Information Assurance*, pages 210–217, United States Military Academy, West Point, NY, June 2004.

[5] Lawrence Teo and Gail-Joon Ahn. Supporting access control policies across multiple operating systems. In *Proceedings of the 43rd ACM Southeast Conference*, volume 2, pages 288–293, Kennesaw, GA, March 2005.

[6] Lawrence Teo, Gail-Joon Ahn, and Yuliang Zheng. Dynamic and risk-aware network access management. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 217–230, Como, Italy, June 2003.