Extensible policy framework for heterogeneous network environments

Lawrence Teo

UNC Charlotte,
9201 University City Blvd.,
Charlotte, NC 28223, USA
E-mail: lcteo@uncc.edu

Gail-Joon Ahn*

Lab. of Security Engineering for Future Computing (SEFCOM),
Arizona State University,
699 South Mill Ave.,
Tempe, AZ 85281, USA
E-mail: gahn@asu.edu
*Corresponding author

Abstract: Security policy management is critical to meet organisational needs and reduce potential risks because almost every organisation depends on computer networks and the internet for their daily operations. It is therefore important to specify and enforce security policies effectively. However, as organisations grow, so do their networks – this increases 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 organisations implement security policies in changing, diversity-rich environments. We also describe our ongoing work in the experimentation of Chameleos-x, where we have obtained promising results.

Keywords: access control; grid systems; assured sharing; security.

Reference to this paper should be made as follows: Teo, L. and Ahn, G-J. (2013) 'Extensible policy framework for heterogeneous network environments', *Int. J. Information and Computer Security*, Vol. 5, No. 4, pp.251–274.

Biographical notes: Lawrence Teo received his PhD at the College of Computing and Informatics, University of North Carolina at Charlotte, Charlotte.

Gail-Joon Ahn is a Full Professor in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. His current research interests include information and systems security, vulnerability and risk management, access control, and security architecture for distributed

systems. His research has been supported by the US National Science Foundation, US National Security Agency, US Department of Defense, Bank of America, Hewlett Packard, Microsoft, and Robert Wood Johnson Foundation. He is also a recipient of the US Department of Energy CAREER Award and the Educator of the Year Award from the Federal Information Systems Security Educators Association.

1 Introduction

Businesses and organisations 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 organisations to implement a carefully-designed security policy for their networks and information systems.

However, this is not without its challenges. When organisations 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 (Sachs et al., 2009). As the number and types of systems increase, the security of the organisational 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

2 the ability to maintain the consistency of the policy in the face of changing organisational requirements becomes diminished.

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.

Chameleos-x began its life as a language to specify access control policies across different operating systems (Teo and Ahn, 2004, 2005). Through our experience and the many lessons learned while designing and developing Chameleos-x, we are currently extending Chameleos-x to work beyond operating systems. Our new vision for Chameleos-x is for it to enforce security policies consistently on different security-aware systems (also known as INFOSEC devices by some researchers). We define a security-aware system as any electronic system that is responsible for enforcing any part of the organisational security policy. Examples of security-aware systems include operating systems, firewalls, and intrusion detection systems (IDSs) – those are the systems that we will focus on in this paper.

Chameleos-x differs from previous approaches in two important ways. First, Chameleos-x attempts to deploy comprehensive security policies for many types of systems as we have already discussed. Second, it uses a three-pronged strategy to enforce policies for those systems

1 Chameleos-x assists in the *configuration* phase of the policy deployment process

2 Chameleos-x allows the *evaluation* of policies so that the organisation can be confident that the policies work as expected

3 Chameleos-x has a *response* mode, which refers to the ability of the Chameleos-x architecture to proactively enforce the policy based on changing conditions and events using a risk-aware mechanism.

In this paper, we focus on the configuration phase of the policy deployment process; the evaluation and response phases remain as future work, but we shall address them briefly.

Chameleos-x benefits organisations by introducing numerous cost and time savings. Users would be able to use a language with a common syntax to design the security policy for heterogeneous systems. They would not have to relearn a different syntax in order to deploy the same policy from one system to the next. System designers and network engineers benefit by being able to design more secure and reliable systems and networks as the result of a systematic policy deployment process.

This paper is organised as follows. Section 2 describes other approaches that are related to our work. Section 3 presents the objectives of the Chameleos-x policy framework. We then discuss the design, architecture, and realisation of Chameleos-x in Section 4, followed by a discussion of our experiments and results in Section 5. Next, we describe our ongoing and future work in Section 6. Section 7 concludes the paper.

2 Related work

Chameleos-x is a family of languages and policy specification and enforcement architectures. Therefore, there are quite a number of projects that are related to our work. To simplify our discussion, we present them in three categories: policy specification languages, network management systems, and security management systems.

Ponder (Damianou et al., 2001) is a policy specification language for distributed systems. It is also a flexible language that targets a number of different systems. The difference between Ponder and our work is that Ponder is strictly a policy specification language, while Chameleos-x is involved in both policy specification and enforcement. In terms of responding to events, the Ponder framework has a 'self-management' feature, which consists of *statically* defined obligation policies that decide what action should be taken when certain events happen. In contrast, the Chameleos-x architecture includes a *dynamic* and risk-aware response mechanism at the enforcement phase. Also, Chameleos-x has pluggable policy sets to support multiple system entities in large and heterogeneous environments and also provides a facility to help evaluate specified policies. In addition, it helps us identify missing or conflicting policies as well as guide us to design effective policy sets.

Woo and Lam (1993) designed a flexible language that used default logic to model authorisation rules. The Authorisation Specification Language (ASL) (Jajodia et al., 1997a, 1997b) is a flexible and expressive language that can be used for multiple access control policies. Related projects also include work on modular authorisation (Wedde and Lischka, 2001), logical access control frameworks (Bertino et al., 2003), and enterprise-level privacy policies (Karjoth and Schunter, 2002).

The motivation of our work should not be confused with that of eXtensible Access Control Markup Language (XACML) (OASIS, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.). XACML is an XML-based language. Its goal is to allow access control policies to be specified by any application that requires authorisation for users. In our work, we strive to design a flexible language within a particular domain. Currently, Chameleos-x focuses on network services in operating systems, firewalls, and IDSs.

In the network management area, the most popular system is the simple network management protocol (Case et al., 1990). Although it is widely used, SNMP has many drawbacks, especially its very simplistic design that makes it unsuitable for specifying and enforcing comprehensive security policies on a network. A different network management protocol, the common management information protocol (CMIP) (ITU-T ISO/IEC, 1991) has a wider scope compared to SNMP but its resource-intensive nature has limited its adoption on the internet. Another network management system is Telcordia's Transaction Language 1 (TL1) (iDeskCentric, Inc., http://ireasoning.com/), which is used in telecommunications equipment. In contrast, Chameleos-x targets TCP/IP networks. The NEon architecture (Schuba et al., 2005) uses a very different paradigm for network management, where it offers an integrated approach to manage network services. It is designed mainly from the network perspective with a focus on generating rules for firewalls. Chameleos-x adopts a different approach where it aims to cover a variety of systems, both in terms of heterogeneity (multiple platforms) and type of system (firewalls, IDSs, and other security-aware systems).

We now discuss security management systems. In the operating systems area, Rippert (2003) has proposed a framework called THINK to protect flexible operating system architectures. However, THINK is tightly integrated to operating system kernels. In contrast, we are developing a framework with a declarative language to support security policies for operating systems, firewalls, and IDSs. In addition, we aim to support these applications above the kernel layer, in order to make it accessible to practitioners who do not wish to use kernel-level access control facilities.

Firmato (Bartal et al., 2003) is a firewall management toolkit that aims to manage firewalls based on an entity-relationship model that represents security policy and topology. While its approach is very novel, it caters mainly to firewalls. Also, the ability to respond to changes, which is a Chameleos-x requirement, is not part of Firmato's objectives.

Telcordia's 'Smart Firewalls' project (Bhatt et al., 2003; Burns et al., 2001) focuses on the specification and enforcement of high-level policies on a dynamic coalition of networks. Their concentration is on ensuring that various coalition networks conform to a coalition policy that has been agreed upon, and that configuration changes during the life of the networks do not result in any violations of that policy.

3 Objectives and design decisions

The Chameleos-x policy framework has three main objectives:

1 *To facilitate security policy management for heterogeneous environments.* We are primarily concerned with the configuration and evaluation of systems for conformance to security policies in heterogeneous environments. We also wish to

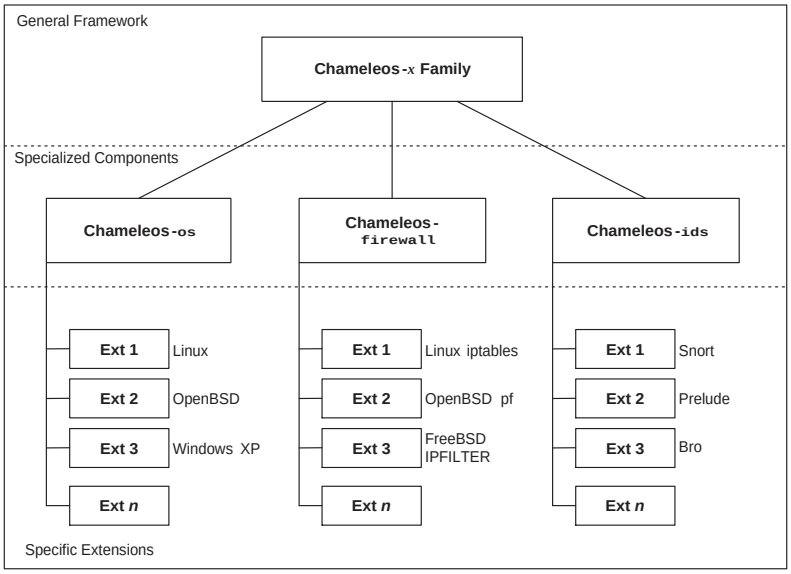
explore methods to build a mechanism that can flexibly and actively respond to malicious events during real-time operation.

2 *To facilitate the specification of sound and effective security policies for multiple kinds of systems and software.* It is difficult to write security policies for complex information systems and networks. In Chameleos-x, we propose to design and develop a simple but powerful declarative language to enable the specification of sound security policies for systems in a large-scale environment.

3 *To develop a reliable architecture to enforce security policies in heterogeneous environments.* We intend to build a policy enforcement architecture that would allow the execution of the security policies written using the Chameleos-x framework. This implies that the architecture has to be designed well to support multiple systems and platforms in heterogeneous network environments.

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 number of supported systems will be increased in the future.

Figure 1 The Chameleos-x framework



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 two 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.

3.1 Approach

We now discuss the approach that was used to design the Chameleos-x policy framework. We present the two key 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 (Teo et al., 2003), 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, we need to consider whether to use a top-down approach or a bottom-up approach. In other words, we need to decide whether to develop our language by testing it regularly with general concepts, or on actual systems. General concepts in this context refer to implementation-independent paradigms, such as access control lists and role-based access control (Giordano et al., 2010). The top-down approach is suitable for flexibility. For Chameleos-x, however, we believe that developing for actual systems (the bottom-up approach) would be more beneficial, since Chameleos-x has to be implemented on real systems in the end.

Some have suggested that the bottom-up approach results in an inflexible framework that is too specific to the underlying systems; however, we argue the reverse to be true. Since Chameleos-x has a language component to it, it is useful to draw parallels with existing programming languages to demonstrate why the bottom-up approach is more suitable in the case of Chameleos-x. We must stress 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. We will initially develop for a small number of systems, and increase the number as we progress. Using this evolutionary model, we believe we will be able to support the specific features of each system more effectively.

3.2 Users

The Chameleos-x policy framework is primarily concerned with the configuration and evaluation of security policies on information systems. Therefore, the most likely users would be system administrators, since they have the most access to the information systems of an organisation. Other users include IT managers, security officers, auditors, and technical staff who are involved with security policies. To simplify our discussion, we use the catch-all term *evaluator* to describe the user of the Chameleos-x framework.

3.3 Criteria

Based on the objectives and approach outlined in the previous sections, we now explain the criteria for Chameleos-x:

1 *Flexibility.* First and foremost, Chameleos-x should be able to support the security policies of multiple types of systems. While we wish to support all types of systems that have security policies, at this point in time, we are mainly interested in operating systems, firewalls, and IDSs.

2 *Extensibility.* Of equal importance is the extensibility of Chameleos-x (Lesniewski-Laas et al., 2007). Chameleos-x must be able to support the specific features of each system. A benefit of this is that it can allow a system to use only the features that it requires and nothing more. Consider the operating systems domain. Two different OSs may support different kinds of security policies. For instance, an OS like Security-Enhanced Linux (SELinux) (NSA, http://www.nsa.gov/selinux/) has comprehensive security policies that support discretionary access control (DAC), role-based access control (RBAC), and type enforcement. Another OS like OpenBSD follows the DAC paradigm only. Therefore, if the Chameleos-x is being used on OpenBSD, it would need to support just the DAC paradigm; however, on SELinux, Chameleos-x can be extensible enough to support RBAC and type enforcement.

3 *Practical and system-driven.* Chameleos-x must be a practical policy framework for real systems as opposed to a theoretical one.

4 *Language with well-defined syntax.* The language component of Chameleos-x must have a well-defined syntax to promote clarity and reduce the ability to introduce ambiguity.

5 *Comprehensiveness.* Large and heterogeneous environments may have different kinds of platforms and systems. Therefore, Chameleos-x must provide facilities to enable the support for any target system. This includes the ability of the language component of Chameleos-x to define groups and sets, and other features that we will describe later in Section 4.2.

6 *Use of a textual language.* By textual language, we mean that the language component in Chameleos-x should not be confined to a graphical user interface (GUI). The language itself should be expressible in ASCII text so that Chameleos-x policies are readable by humans without having to use special tools.

7 *System and platform independence.* We strive to develop a policy framework that can work across multiple systems, with the major requirement that the system must be one that supports the specification and enforcement of security policies.

Given that the criteria include flexibility and extensibility, one might suggest that XML would be a good candidate to develop the language component of the Chameleos-x policy framework. After all, XML is designed to be highly extensible while maintaining structure. However, at present, we do not wish to adopt XML as the language for Chameleos-x for two reasons. The first reason is because we would like security policies written in Chameleos-x to be simple and readable. Security policies for

diverse environments can be very complex; thus, using XML may reduce the readability of such security policies. The second reason is that we are trying to minimise changes to existing policy practices when introducing Chameleos-x. Currently, almost all operating systems, IDSs, and firewalls do not use XML for their policy frameworks.

At the same time, we are not completely dismissing the adoption of XML for Chameleos-x in the future. Eventually, there may be a need to exchange security policies across domains and systems. The ideal candidate to implement this exchange mechanism would be XML, since it is especially useful in representing information for exchange across systems. Perhaps when Chameleos-x reaches a mature stage in its development, and XML is used natively in systems, we may investigate possible ways to incorporate XML into the framework. One possible way would be to have an XML version of the Chameleos-x language component; another alternative would be to write a new translation component for the Chameleos-x architecture that would allow the conversion of Chameleos-x policies to and from XML.

4 Policy framework: Chameleos-x

The Chameleos-x policy framework can be implemented using two major components: a policy specification language and a policy enforcement architecture. In this section, we describe the terminology used to describe the components in Chameleos-x, and discuss the Chameleos-x language and architecture. As we are presently developing the Chameleos-x framework, some of the details in this section may change in the future, when we revise the design based on our experimental results. However, the core ideas and concepts used in the design of the Chameleos-x framework are valid and will be retained.

4.1 Terminology

The terms and definitions used in the context of Chameleos-x are as follows:

1 *Operation modes.* Chameleos-x is designed to work with three high-level operation modes: configuration, evaluation, and response. The configuration mode is used to propagate configuration settings to a variety of systems. The evaluation mode is used to test the configuration against a set of events (say, legitimate traffic and attack traffic), to see if the configuration settings and the security policy are working effectively as they should be. The response mode is used during real-time operation, and is meant to enforce the security policy by taking actions based on certain conditions and events.

2 *Session.* A session represents a typical period of time when Chameleos-x is run. For example, the session can define which Chameleos-x variants are used and what operation mode to invoke in that session.

3 *Context.* A Chameleos-x context refers to an entity that is affected by the configuration, evaluation, and response operation modes. A context can refer to an entity on just one single system, or span across multiple systems. For example, a single context can be defined to represent the HTTP service on an operating system, the IDS rules affecting HTTP on an IDS, and firewall rules that allow or

deny the HTTP service. Since we are focusing on networks at this stage, most contexts will be related to network services.

4 *Context library.* To facilitate the definition of contexts, Chameleos-x supports context libraries. A context library, as its name implies, is a collection of contexts related to a particular domain. Currently, we are primarily interested in building a context library in the network domain.

4.2 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 do not include the EBNF grammar specification in this paper.

In order for the Chameleos-x framework to work effectively in an organisation, it would naturally need an architecture to support it. Since the size of the information systems that Chameleos-x will work with could potentially be very large, we made a decision to have just a small number of components in the Chameleos-x architecture.

The three major components of the Chameleos-x architecture are the management console, translator, and enforcement mon. The management console, like its name implies, 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. As we can see, the management console is used to push the Chameleos-x policies to the Chameleos-x enforcement monitors residing on each system, where it is translated and applied. This process can be 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.

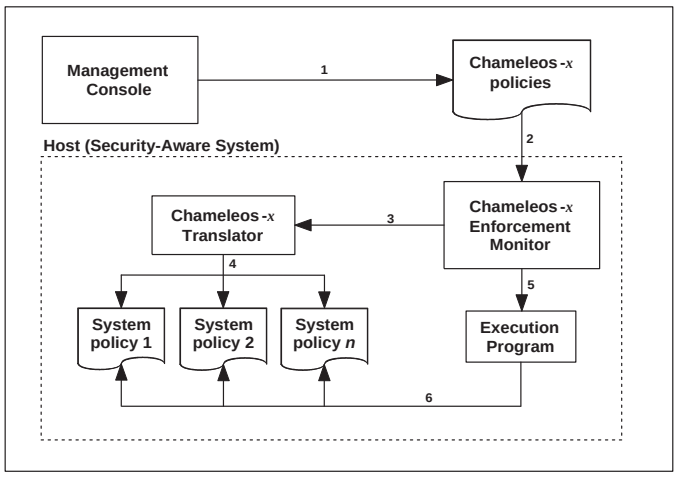
3 The Chameleos-x enforcement monitor invokes the translator to translate the policies it just received into system-specific policies.

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.

Figure 2 The Chameleos-x architecture



Since the Chameleos-x framework is intended to work with heterogeneous networks, it is critical to consider installation and scalability issues in the deployment of the Chameleos-x architecture. The main idea is to have many instances of the Chameleos-x enforcement monitor running on multiple hosts in a large network. The evaluator can then write a single policy on a central management console and 'broadcast' updates to the monitors. With this in mind, we need to concentrate on the installation issues involving the Chameleos-x enforcement monitor, since the monitor will be installed throughout the network. First of all, the monitor has to be installed manually by hand on the hosts in the network – this is something that cannot be avoided. However, once installed, it is not realistic to assume that the administrator should update all the monitors by hand as new versions of the monitor become available.

It is therefore highly desirable for the monitor to be as simple and lightweight as possible. The monitor also needs to be self-contained; it should be able to run reliably with minimal oversight by the administrator. The monitor should also preferably be able to poll a specific update server periodically and update itself securely with a new version if it is available. All configuration and updates should be conducted in a centralised

manner as much as possible. As we are targeting heterogeneous networks, the monitor should also be developed in a portable manner.

4.3 Realisation of Chameleos-x

In this section, we discuss our ongoing work that we are carrying out towards a full implementation of Chameleos-x. As mentioned in the introduction, we concentrate on the configuration phase of the policy deployment process only at this point. Since Chameleos-x is still undergoing development and experimentation, we will just present excerpts of policies that we are currently working on, which would help us refine our goals and future directions. Our goal is to carry out experiments using these implementations to develop newer and more robust versions of Chameleos-x. Also, we have used open source systems only at this stage due to their wider availability.

The first implementation decision we have to make is the language that we should use to implement the Chameleos-x components like the Chameleos-x enforcement monitor and the Chameleos-x translator. Due to the multi-platform nature of Chameleos-x, Java would sound like the natural choice as the development language for the components, especially the monitor. However, we chose to implement the components in portable C instead, since the monitor would need to access low-level services on the hosts, which is something that Java does not offer conveniently. The translator was implemented using Perl, while the extension routine libraries were developed using Perl and Bourne shell scripts, where appropriate. These components ran on the three UNIX systems: Linux, FreeBSD, and OpenBSD. In addition, our management console used NetBSD as its operating system. The decision to use different platforms was deliberately made in order to test Chameleos-x's ability to function in heterogeneous environments, which is our goal. Although we have implemented these components on UNIX-based systems so far, we may port them to Windows-based systems in the future, either using native tools like WScript or development tools like ActiveState Perl.

We shall now discuss the example policies for Chameleos-`os` (Figure 3), Chameleos-`firewall` [Figure 5(a)], and Chameleos-`ids` [Figure 6(a)] in turn. As mentioned in Section 4, where we discussed the language and architecture of Chameleos-x, the Chameleos-x policy framework uses contexts to describe various entities that are affected by the configuration and evaluation processes. Contexts are stored in context libraries to facilitate the creation of new Chameleos-x policies. While we could develop our own context library, for the purposes of experimentation in this section, we chose to select a context library that is ubiquitous on UNIX systems – the `/etc/services` file. We are aware that this file has certain deficiencies such as the issue of working with protocols and architectures that dynamically select ports in real time (e.g., RPC mapping, passive FTP, and the IBM AS/400 system). However, we believe that the file itself is comprehensive enough to be used to demonstrate the feasibility of our approach. In a real world implementation, we may augment our current approach with a dynamic network discovery mechanism to populate the context library.

Figure 3 A simple Chameleos-os policy – *simple.cos*

```
chl_os {
 config {
 activate_context("http");
 activate_context("ftp");
 }

 eval {
 activate_context("http");
 activate_context("ftp");
 }

 response {
 threshold("http", 100, 10) {
 deactivate_context("http");
 }

 threshold("ftp", 40, 10) {
 deactivate_context("ftp");
 }
 }
}
```

4.3.1 Chameleos-os

Each Chameleos-x policy can be divided into three sections, which relates to the three operation modes that we discussed in Section 4: configuration (`config`), evaluation (`eval`), and response (`response`). The correct section of the policy will be invoked depending on which of these three operation modes is being used in a typical session. Consider the Chameleos-os policy in Figure 3. In this policy, the evaluator is interested in activating the *http* and *ftp* contexts during the configuration and evaluation operation modes.

The response operation mode requires slightly further explanation. Eventually, we intend to integrate Chameleos-x with mechanisms on dynamic network access management (Teo et al., 2003). In Teo et al. (2003), a risk-based network access management architecture was designed that uses two primary parameters – threat level and threshold – to allow or deny certain behaviour (such as incoming traffic into a network). The threat level represents the amount of risk associated with a certain event, where a high threat level represents high risk. If the threat level increases beyond a certain threshold, a certain response action is taken, such as stopping the event. The threat level and threshold mechanism is intentionally 'generic' in nature, so that it can be used to accommodate systems outside the network domain. This generic property enables the risk-based architecture to be flexible and effective, while not being confined to a specific system. This is proven by the fact that the architecture has been deployed in the context of a generic network traffic analyser (Teo et al., 2003) and a honeypot-based security framework (Teo et al., 2004) in the past.

In the Chameleos-os policy in Figure 3, we use this threat level and threshold mechanism in the `response` block. The `threshold()` function is used to declare the

action that should be done if the threat level of a context exceeds the threshold. In the example, the evaluator is defining the threshold of the *http* context with two parameters: 100 and 10. This simply means that the *http* context should be deactivated when it receives 100 connections in 10 seconds (note that deactivation is just one possible response; we shall address this in the next paragraph). Likewise, the *ftp* context should be deactivated when it receives 40 connections in 10 seconds. At this stage, we are still exploring possible responses in terms of their feasibility, interactions with other components, and granularity. The full details will be published in a future paper.

It should be emphasised that the response need not be confined to just activation and deactivation. We use activation and deactivation here strictly to aid our current prototype only (and we are fully aware that deactivation may seem like a denial-of-service attack). As future work, we intend to explore other responses such as redirection to a honeypot (to examine suspicious events) or redirection to another host in a server farm (for load balancing, say to offset a distributed denial-of-service attack).

Exactly how a context is activated or deactivated is dependent on the implementation and the underlying platform. To illustrate, suppose the underlying platform is a UNIX variant like Linux running an Apache web server and some stock FTP server. In this case, activation for the *http* context would consist of starting the Apache web server using the command `apachectl start`. In a similar vein, deactivation would be done using the command `apachectl stop`. These commands would be issued by the Chameleos-*x* enforcement monitor that is running on the host. A similar approach would be used for FTP servers. Since FTP servers are frequently started from the `inetd` superserver, activation and deactivation can be done by modifying `inetd`'s configuration file and sending it a SIGHUP signal to request it to reload its configuration. On Windows-based systems, activating and deactivating contexts could be done by starting and stopping the correct service respectively.

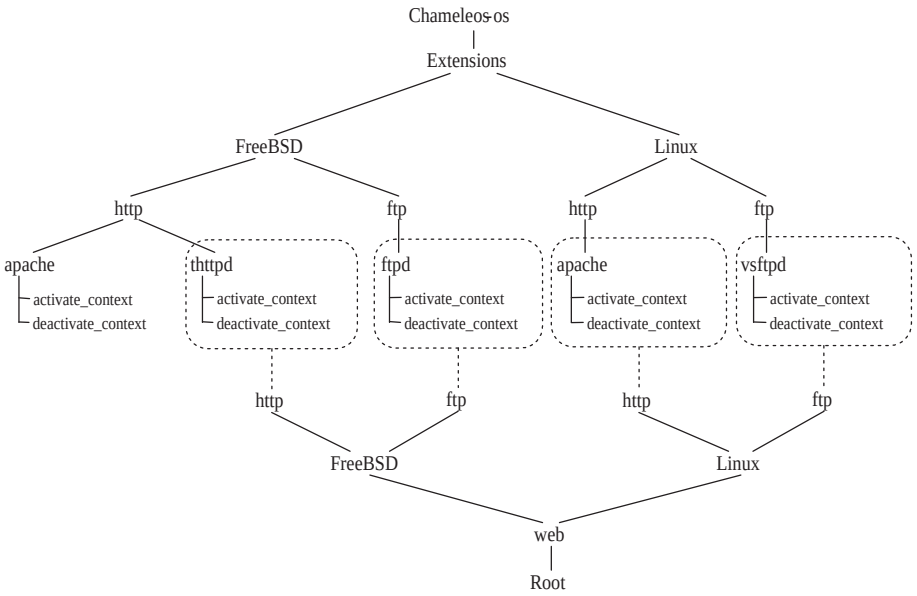
There is one major challenge when considering the practical method to activate and deactivate contexts on servers: There are many operating systems and there are many types of servers. Supposing we need to deal with a webserver, an administrator may prefer to run Apache if it is a Linux webserver, or thttpd if it is a FreeBSD webserver. Although both Linux and FreeBSD are UNIX-like systems, the actual techniques to start and stop Apache and thttpd on Linux and FreeBSD are different. From the viewpoint of Chameleos-*x*, all we really want to do is just to start and stop a webserver.

To reduce the complexity and to facilitate the management of this situation, we use a role-based representation structure that identifies the capabilities of the underlying system, so that the administrator can assign 'roles'[1] to servers with the desired behaviour. The role is related to Chameleos-`os` extensions, so that the correct activation and deactivation routines can be invoked for a particular role.

This concept is illustrated in Figure 4. In the diagram, the top-down tree represents the Chameleos-`os` extensions (in this case, there are two extensions: FreeBSD and Linux). Each extension has the two contexts *http* and *ftp* associated with it. In turn, the context may be linked to one or more server activation/deactivation routines. The *http* context for FreeBSD, for instance, has routines for activating and deactivating Apache and thttpd. The Linux *http* context, on the other hand, only has routines for Apache. The bottom-up tree shows how roles can be used to select the desired servers to activate. In Figure 4, the *web* role is linked to the thttpd and ftpd routine libraries in the FreeBSD extension, and the Apache and vsftpd libraries in the Linux extension (while we only show a single role called *web* in the diagram, multiple roles are possible).

Thus, if the administrator assigns the *web* role to the webserver, and the webserver is running FreeBSD, the thttpd and ftpd servers on FreeBSD will be started when the Chameleos-`os` *http* and *ftp* contexts are activated. Likewise, if the webserver is running Linux, the Apache and vsftpd servers will be run when the *http* and *ftp* contexts are activated.

Figure 4 Chameleos-os extensions



In our implementation of the Chameleos-x enforcement monitor, we organised the extensions as a hierarchy of directories according to the top-down tree in Figure 4. This hierarchical representation provides Chameleos-`os` with maximum flexibility and extensibility: the administrator is able to select various server and operating systems intuitively, and assign the desired combinations to a role. That way, the administrator can specify and enforce policies by simply specifying the appropriate role identifier in the session file from the management console.

4.3.2 Chameleos-`firewall`

The Chameleos-`firewall` policy in Figure 5(a) is similar to the Chameleos-`os` policy, where it also has three sections: configuration, evaluation, and response. Firewalls tend to follow either a default-deny or default-allow policy, depending on the nature of the organisation. For instance, an organisation that favours a more restricted and closed environment, such as the military, tend to opt for a default-deny policy. More open environments like academic institutions may opt for a default-allow policy. To support these policies, we have implemented a `global_policy()` function which accepts one of two constants: `CHL_FW_DEFAULT_DENY` (for default-deny) or `CHL_FW_DEFAULT_ALLOW` (for default-allow). We also use a `define_group()` function which allows groups to be defined. In this example, we have defined a group called *blacklist* that consists of the IP address 10.0.0.3. Note that we can actually include more than one IP address in a

group; we are just using a single IP address in this case to simplify our discussion. In the configuration section, we deny access to the *ftp* context from the *blacklist* group by using the `deny_context()` function. We allow access to the *http* context from everyone else by using the `allow_context()` function with the constant `CHL_FW_ALL`.

In terms of implementation, we used a directory tree to store the extension routines shown in Figure 5(b). In the diagram, Chameleos-`firewall` has two extensions: *pf* and *iptables*. Each routine in the extension (init, global_policy, etc.) was implemented as either a Perl script or Bourne shell script. The idea is to have these scripts generate a configuration file for their respective extensions. For instance, in the case of *pf*, the translator would translate the Chameleos-`firewall` policy in Figure 5(a) by converting it into a configuration file that *pf* can load (using the pf.conf syntax). On the other hand, if *iptables* was used, the translator would generate a shell script with the relevant `iptables` commands according to the original Chameleos-`firewall` policy.

The actual translation process is outlined in Figure 5(c). The initialisation process is in charge of initialising system-specific variables, as these may differ from system to system (for example, Linux usually uses 'eth0' as the generic name for the first network interface, while the BSD systems tend to use driver-based names, such as 'xl0' or 'dc0'). The next stage in the translation process is generation, which refers to the generation of a system policy, which is either a configuration file (as is the case for *pf*) or another type of file (like the shell script for *iptables*). If we are translating the Chameleos-`firewall` policy in Figure 5(a), the scripts `global_policy`, `define_group`, `deny_context`, and `allow_context` would be called with the correct parameters. These scripts append the correct system-specific configuration details or commands to the system policy. The next step is validation. At this point, validation is selective, since some extensions do not allow for straightforward validation. For instance, it is easy to validate a *pf* configuration file, since *pf* provides the necessary tool to do so (`pfctl -n <filename>`). However, it is not easy to validate a shell script with `iptables` commands. It is not impossible but for the current prototype, we are omitting this step for now and will explore it in the future. The last step of translation is execution. The `exec` script is called, which actually loads the system policy so that the Chameleos-`firewall` policy is enforced.

4.3.3 Chameleos-`ids`

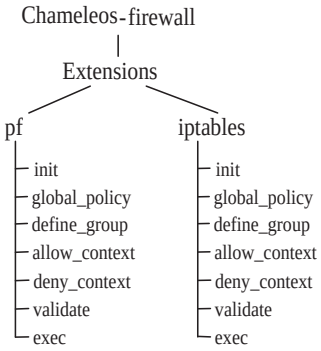
Now that we have discussed the Chameleos-x policies for operating systems and firewalls, the discussion of the Chameleos-x policy for IDSs in Figure 6(a) would be relatively straightforward. We initially considered using Snort (http://www.snort.org/) and Prelude (Vandoorselaere, 2012) as Chameleos-`ids` extensions – however, Prelude actually uses Snort as its sensor, so we decided just to focus on Snort only so that we do not perform redundant work. This results in a simple extension tree as shown in Figure 6(b).

In our sample policy [Figure 6(a)], we directed the Chameleos-x enforcement monitor to use the ruleset in the file *web-attacks.rules* during configuration by calling the function `ids_apply_ruleset()`. The IDS is informed that we are interested in attacks related to the *http* context.

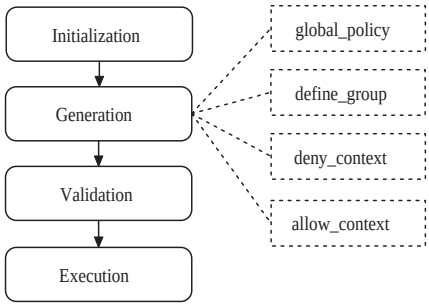
Figure 5 (a) A simple Chameleos-`firewall` policy – *simple.cfw*
(b) Chameleos-`firewall` extensions (c) Translation flowchart for all Chameleos-x variants, using Chameleos-`firewall` as an example

```
chl_fw {
 global_policy(CHL_FW_DEFAULT_ALLOW);

 define_group(blacklist, "10.0.0.3");

 config {
 deny_context("ftp", "tcp", blacklist);
 allow_context("http", "tcp", CHL_FW_ALL);
 }
}
```

(a)



(b)



(c)

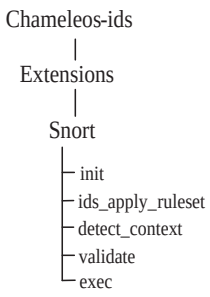
The translation process is similar to the one used for Chameleos-`firewall` [Figure 5(c)]. In this case, the system policy is the Snort configuration file. In the initialisation phase, system-specific variables are assigned the correct values (such as the home subnet, which is 172.16.0.0/16). In the generation phase, the correct ruleset is added to the

configuration file by calling the script `ids_apply_ruleset` with the correct parameters. The next command in the Chameleos-`ids` policy is `detect_context()`. In Snort, this translates to a Berkeley packet filter (BPF) filter rule, which we output to a file called `bpf-filter.txt` (since we are interested in the *http* context, the contents of the text file is simply 'port http' according to BPF syntax). Validation is then easily done by calling Snort to test the configuration file with the `-T` command-line option. Finally, execution is done by restarting Snort with the new configuration file, thus enforcing the Chameleos-`ids` policy in Figure 6(a).

Figure 6 (a) A simple Chameleos-`ids` policy – *simple.cids* (b) Chameleos-`ids` extensions

```
chl_ids {
 config {
 ids_apply_ruleset("web-attacks.rules");
 detect_context("http");
 }
}
```

(a)



(b)

5 Experiments and results

Our experiments were designed with two objectives:

1 to test the translation process of each Chameleos-x variant

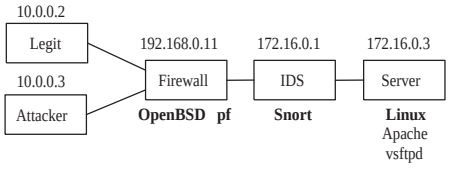
2 to test the enforcement/execution process of each Chameleos-x variant.

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.

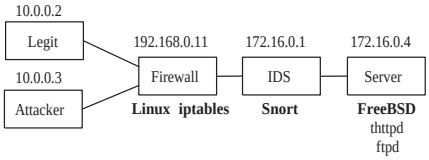
As we considered the possible approaches we could use to fulfill the two experimental objectives, we came to the conclusion that we needed to use two 'configuration suites' for testing each variant. A configuration suite would consist of a specific firewall, IDS, and server operating system and associated 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.

With that in mind, we designed the configuration suites shown in Figures 7(a) and 7(c). 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 management console is not shown in the figures to conserve space).

Figure 7 (a) Configuration suite 1 (b) Session file for configuration suite 1 – *session-suite1.chl* (c) Configuration suite 2 (d) Session file for configuration suite 2 – *session-suite2.chl*



(a)



(c)

```
declare_session("session-suite1") {
chl_fw("pf", "web",
"192.168.0.11", "simple.cfw");
chl_ids("snort", "web",
"172.16.0.1", "simple.cids");
 chl_os("linux", "web",
"172.16.0.3",  "simple.cos");
}
```

(b)

```
declare_session("session-suite2") {
chl_fw("iptables", "web",
"192.168.0.11",  "simple.cfw");
chl_ids("snort", "web",
"172.16.0.1",  "simple.cids");
chl_os("freebsd", "web",
"172.16.0.4",  "simple.cos");
```

(d)

We then ran our experiment in two sessions: one with configuration suite 1, and one with configuration suite 2. In each session, the management console sent the Chameleos-x variant policies defined in Figures 3, 5(a), and 6(a) to the respective machines on the network. We then examined the machines to see if the translation process and the enforcement/execution process worked as expected. If the behaviour of the machines in each configuration suite is the same, this means that our experimental objectives have been fulfilled.

To run the sessions, we developed a session file for each suite [Figures 7(b) and 7(d)]. The session file states where the Chameleos-x policy should be sent: for example, in Figure 7(b), the `chl_fw` statement specifies that the Chameleos-`firewall` policy in the file `simple.cfw` should be sent to 192.168.0.11 and the *pf* extension should be invoked with the *web* role (actually only Chameleos-`os` uses roles right now, but this is specified to support other Chameleos-x variants in the future). We then invoked the following command at the management console:

```
$ chl-apply config session-suite1.chl
```

The `chl-apply` program sends the policies specified in the session file to the Chameleos-x enforcement monitors. It also has the option to specify which operation mode should be used, which in this case is `config`. This will cause the monitors to only translate the excerpt within the `config` clause in each Chameleos-x policy.

The actual communication process involves the transmission of the policies from the management console to the monitors using a straightforward TCP socket connection at this stage. This limitation can be overcome through a more secure architecture that incorporates proven technologies like PKI and SSL. It remains as our future work.

The results from the experiments are shown using either actual screen snapshots or the translated output from the firewall, IDS, and server machines. These results are obtained *after* the Chameleos-x enforcement monitor has applied the Chameleos-x policies on each system.

We begin with the results for configuration suite 1. We first initialised all systems to their 'default' states – for example, the server was not running any HTTP and FTP servers, the firewall did not have any customised firewall rules, and the IDS did not have any rules installed. After invoking the Chameleos-x enforcement monitor, the Apache and vsftpd programs were activated on their respective ports (80 and 21) on the server [Figure 8(a)], which is the expected result.

Figure 8 Configuration suite 1, (a) screen snapshot for Chameleos-`os` (b) output firewall rules for Chameleos-`firewall`

```
root# ps ax | grep -e httpd -e vsftpd
8014 ? Ss 0:00 /usr/sbin/httpd
8017 ? S 0:00 /usr/sbin/httpd
8018 ? S 0:00 /usr/sbin/httpd
8019 ? S 0:00 /usr/sbin/httpd
8020 ? S 0:00 /usr/sbin/httpd
8021 ? S 0:00 /usr/sbin/httpd
8016 pts/0 S 0:00 /usr/sbin/vsftpd /etc/vsftpd.conf
```

(a)

```
ext_if = "em1"
int_if = "em0"
pass in all
pass out all
table <blacklist> { 10.0.0.3 }
block in on $ext_if proto tcp from { <blacklist> } to port ftp
pass in on $ext_if proto tcp from any to port http
```

(b)

The Chameleos-`firewall` results in the form of firewall rules are shown in Figure 8(b). The default-allow policy specified in the original Chameleos-`firewall` policy has been translated into two *pf* rules: 'pass in all' and 'pass out all'. The *blacklist* group was defined using *pf*'s table feature, and the `deny_context` and `allow_context` functions have been translated into *pf*'s 'block' and 'pass' rules respectively.

The Chameleos-`ids` results in the form of a Snort configuration file are shown in Figure 9(a). After running the Chameleos-x enforcement monitor, the system

policy was translated from the Chameleos-ids policy in Figure 6(a). Note that the web-attacks.rules file has been loaded, which was what was requested via ids_apply_ruleset in the original Chameleos-ids policy. Also, Snort has been instructed to listen to the HTTP port only via detect_context, which translates to a BPF filter rule stored in the file output-ids/bpf-filter.txt [Figure 9(b)].

Figure 9 Configuration suites 1 and 2, (a) output IDS rules for Chameleos-ids (b) screen snapshot showing Snort's command-line arguments and the output of the output-ids/bpf-filter.txt file

```
var HOME_NET [172.16.0.0/16]
var HTTP_SERVERS $HOME_NET
var HTTP_PORTS 80
var EXTERNAL_NET any
var RULE_PATH /usr/local/share/examples/snort

preprocessor flow: stats_interval 0 hash 2
preprocessor frag2
preprocessor stream4: disable_evasion_alerts
preprocessor stream4_reassemble
preprocessor rpc_decode: 111 32771
preprocessor bo
preprocessor telnet_decode

include $RULE_PATH/classification.config
include $RULE_PATH/reference.config
include $RULE_PATH/web-attacks.rules
```

(a)

```
root# ps ax | grep snort
30618 ?? Is 0:00.06 snort -D -c output.policy -F output-ids/bpf-filter.txt
root# cat output-ids/bpf-filter.txt
port http
```

(b)

The systems in configuration suite 2 are run using the same steps as configuration suite 1. In each case, we show the snapshots of the systems after the monitor was run (Figure 10). The Chameleos-os results in Figure 10(a) show that the thttpd and ftpd daemons are run when the *http* and *ftp* contexts are activated, since the *web* role directs FreeBSD to run those particular servers (Figure 4). The Chameleos-firewall results in Figure 10(b) show that the original Chameleos-firewall policy [Figure 5(a)] has been translated into a shell scripts with iptables commands as expected. The default-allow requirement was implemented with three iptables commands on the three chains FORWARD, INPUT, and ACCEPT. Also, since *iptables* does not support the grouping of IP addresses, we have translated the Chameleos-firewall define_group function as an iptables command describing what to do with the members of the group. In this example, we show the iptables command for only one group member – 10.0.0.3.

For more group members, there would be one `iptables` command for each individual member.

Figure 10 Configuration suite 2, (a) screen snapshot for Chameleos-`os` (b) output firewall rules for Chameleos-`firewall`

```
root# ps ax | grep -e thttpd -e ftpd
38156 ?? Ss 0:00.01 /usr/local/sbin/thttpd
38159 ?? Ss 0:00.00 /usr/libexec/ftpd -D
```

(a)

```
ext_if="eth2"
int_if="eth1"
iptables -P FORWARD ACCEPT
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -A FORWARD -s 10.0.0.3 -i $ext_if -o $int_if -p tcp --dport ftp -j REJECT
iptables -A FORWARD -s 0/0 -i $ext_if -o $int_if -p tcp --dport http -j ACCEPT
```

(b)

The translated policies showed consistent behaviour 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 behaviour was consistent).

These favourable 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 behaviour exhibited by the multiple kinds of systems in configuration suites 1 and 2. These objectives were achieved by designing the framework in accordance to the criteria discussed in Section 3.3.

6 Ongoing and future work

Our experiments have confirmed the feasibility of Chameleos-x. Our next step is to extend and develop Chameleos-x to make it support more target systems. For instance, we intend to examine Windows-based operating systems and complex operating systems with comprehensive access control policies like SELinux (NSA, http://www.nsa.gov/selinux/), as well as a few commercial firewalls and intrusion detection systems.

We are also working 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, analyser, and reverse translator. The *syntax checker* would serve as the foundation for all syntax checking requirements in the other components. The *analyser* would be used to analyse a Chameleos-x policy for conflicts and ambiguities. The analyser would have to take constraints (Jaeger, 1999) and conflict resolution techniques (Jaeger et al., 2004) into account, especially for complex systems like SELinux (Jaeger et al., 2003). The *reverse translator*'s role is to translate a system-specific policy into a Chameleos-x policy.

Other areas that we are working on include improvements to the language itself, such as safety analysis, safety checks, and support for dependencies among extensions (we could borrow concepts like inheritance from the object-oriented domain). In the usability area, we intend to introduce stock Chameleos-x templates, which would help administrators define Chameleos-x policies. This would be most useful when using different variants in the Chameleos-x family.

We are also examining possible methods where we can integrate the Chameleos-x framework with some other projects. One of these projects, which involves dynamic network access management (Teo et al., 2003), have already been mentioned earlier in Section 4.3.1. The response capability of the Chameleos-x framework that is based on threat levels and thresholds would be especially useful for that project.

7 Conclusions

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.

In the paper, we discussed the objectives and design decisions of Chameleos-x, and showed how the framework can be built. The implementation of the Chameleos-x framework that is described in this paper consists of two major components:

1 a simple but powerful language that can be used to specify sound policies in heterogeneous environments

2 a multi-platform architecture that can be used to enforce these policies.

We also described our ongoing work in the development and experimentation of Chameleos-x. Our experiments involved the feasibility assessment of Chameleos-x on two heterogeneous 'configuration suites', where each suite comprises a specific operating system, firewall, and IDS. Chameleos-x successfully demonstrated that it is able to translate a single Chameleos-x policy into the system policies for each suite, and still retain consistent behaviour in each case. This confirms 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 organisations, especially those with large and heterogeneous information networks. Based on the promising results obtained through these experiments, we are currently developing Chameleos-x actively for more systems and more complex environments.

Acknowledgements

This work was partially supported by the grants from National Science Foundation and Department of Energy.

References

Bartal, Y., Mayer, A., Nissim, K. and Wool, A. (2003) 'Firmato: a novel firewall management toolkit', *ACM Transactions on Computer Systems*, Vol. 22, No. 4, pp.381–420.

Bertino, E., Catania, B., Ferrari, E. and Perlasca, P. (2003) 'A logical framework for reasoning about access control models', *ACM Trans. Inf. Syst. Secur.*, Vol. 6, No. 1, pp.71–127.

Bhatt, S., Rajagopalan, S. and Rao, P. (2003) 'Federated security management for dynamic coalitions', *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX II)*, pp.47–48.

Burns, J., Cheng, A., Gurung, P., Rajagopalan, S., Rao, P., Rosenbluth, D., Surendran, A.V. and Martin Jr., D.M. (2001) 'Automatic management of network security policy', *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX II)*, pp.12–26.

Case, J., Fedor, M., Schoffstall, M. and Davin, J. (1990) *Simple Network Management Protocol (SNMP)*, RFC 1157.

Damianou, N., Dulay, N., Lupu, E. and Sloman, M. (2001) 'The Ponder policy specification language', *Policies for Distributed Systems and Networks: International Workshop, Lecture Notes in Computer Science*, pp.18–38.

Giordano, M., Polese, G., Scanniello, G. and Tortora, G. (2010) 'A system for visual role-based policy modeling', *J. Vis. Lang. Comput.*, February, Vol. 21, No. 1, pp.41–64.

iDeskCentric, Inc., *Transaction Language 1 (TL1)* [online] http://ireasoning.com/.

ITU-T ISO/IEC (1991) *Information Technology-OSI, Common Management Information Protocol (CMIP) – Part 1: Specification ISO/IEC 9596-1*, ITU-T Recommendation X.711.

Jaeger, T. (1999) 'On the increasing importance of constraints', *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pp.33–42.

Jaeger, T., Sailer, R. and Zhang, X. (2004) 'Resolving constraint conflicts', *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*.

Jaeger, T., Zhang, X. and Edwards, A. (2003) 'Policy management using access control spaces', *ACM Trans. Inf. Syst. Secur.*, Vol. 6, No. 3, pp.327–364.

Jajodia, S., Samarati, P. and Subrahmanian, V. (1997a) 'A logical language for expressing authorizations', in *Proceedings of the IEEE Symposium on Security and Privacy*, May, pp.31–42, Oakland, CA.

Jajodia, S., Samarati, P., Subrahmanian, V. and Bertino, E. (1997b) 'A unified framework for enforcing multiple access control policies', *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.474–485.

Karjoth, G. and Schunter, M. (2002) 'A privacy policy model for enterprises', *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press.

Lesniewski-Laas, C., Ford, B., Strauss, J., Morris, R. and Kaashoek, M. (2007) 'Alpaca: extensible authorization for distributed services', *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp.432–444, ACM.

NSA, *Security-Enhanced Linux* [online] http://www.nsa.gov/selinux/.

OASIS, *OASIS eXtensible Access Control Markup Language TC* [online] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

Rippert, C. (2003) 'Protection in flexible operating system architectures', *ACM SIGOPS Operating Systems Review*, Vol. 37, No. 4, pp.8–18.

Sachs, J., Prytz, M. and Gebert, J. (2009) 'Multi-access management in heterogeneous networks', *Wireless Pervasive Communication*, Vol. 48, No. 1, pp.7–32.

Schuba, C., Goldschmidt, J., Speer, M. and Hefeeda, M. (2005) 'Scaling network services using programmable network devices', *Computer*, Vol. 38, No. 4, pp.52–60.

SNORT, *Snort Intrusion Detection System* [online] http://www.snort.org/.

Teo, L. and Ahn, G-J. (2004) 'Towards the specification of access control policies on multiple operating systems', *Proceedings of the 5th IEEE Workshop on Information Assurance*, pp.210–217.

Teo, L. and Ahn, G-J. (2005) 'Supporting access control policies across multiple operating systems. in *Proceedings of the 43rd ACM Southeast Conference*, pp.288–293.

Teo, L., Ahn, G-J. and Zheng, Y. (2003) 'Dynamic and risk-aware network access management', *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, pp.217–230.

Teo, L., Sun, Y-A. and Ahn, G-J. (2004) 'Defeating Internet attacks using risk awareness and active honeypots', *Proceedings of the Second IEEE International Workshop on Information Assurance*, pp.155–167.

Vandoorselaere, Y. (2012) *Prelude Hybrid IDS* [online] http://www.prelude-ids.org/.

Wedde, H. and Lischka, M. (2001) 'Modular authorization', *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*, pp.97–105.

Woo, T.Y.C. and Lam, S.S. (1993) 'Authorizations in distributed systems: aA new approach', *Journal of Computer Science*, Vol. 6, No. 2, pp.107–136.

Notes

1 Note that we are not using 'role' as defined in access control; instead, we are using the definition of role in the network management context as defined in Teo et al. (2003).