Protocol Decode Based Stateful Firewall Policy Definition Language

Pankaj N. Parmar Priya Rajagopal Ravi Sahita
Intel Corporation

Abstract

The policies for thwarting attacks on systems vary greatly in complexity, ranging from simple static firewall rules to complex stateful protocol state machine analysis. As intrusion detection systems are getting integrated into firewall solutions, there is a need for a language that can define both firewall policies and system intrusion behavior and exhibit inter-operable traits. This paper presents an XML based, self-documenting State-Aware Firewall Language (SAFire) that is designed to express the various kinds of firewall and intrusion behavior.

1. Introduction

The simplest type of firewalls includes simple stateless packet-based filters wherein a set of static rules is applied to every packet in isolation. The second type includes simple stateful firewalls that perform simple protocol decode analysis wherein they detect violations of network and transport layer protocol behavior. The third category of firewalls includes Application Level Gateways and proxies that analyze the more complex application-level protocol behavior. To reduce the processing overhead, these firewalls may be optimized to do selective and intelligent processing based on flow state information and control payload. Increasingly, Intrusion Detection Systems (IDS) are becoming an integral supplement to firewalls by providing comprehensive attack coverage. Hence, the firewall rule definition language must be capable of expressing the varying complexity associated with rules as well as system intrusion.

2. Existing Languages

In this section, we briefly discuss existing policy languages and how SAFire differs from them. One such language defined in academia is Ponder [8]. Ponder supports access control policies with meta constructs like roles and relationships. Ponder allows assigning policies to organization roles and their relationships. Via this abstraction, Ponder policies can map onto access control mechanisms for firewalls, software and other infrastructure elements. Compared to SAFire, Ponder is at a higher abstraction level for security policies. SAFire is the abstraction of firewall/IDS policies within a system, whereas Ponder is used for security policies across an organization. Another security policy language is the Security Policy Specification Language (SPSL) [9]. SPSL was designed to express policies for packet filters, security domains, and the entities that manage policies with focus on IP Security (IPSec). SAFire supports SPSL style filtering rules and extends it for autonomic configuration features. Autonomic configuration means the system itself creates dynamic filters when it notices configured events occurring. There are commercial firewall/IDS languages like Checkpoint's INSPECT* [5], Cisco PIX* configuration [4] and NFR's N-CODE* [7] that are used for defining firewall policies. One major drawback of these languages is that they use proprietary formats for expressing security rules, thereby making integration into other firewall implementations tedious. SAFire's XML based specification facilitates interoperability with other firewall vendors, for example, by the use of open standards like the Extensible Stylesheet Language Transformations (XSLT). XML based specification also allows administrators/developers to leverage well tested authoring, validation [12] and translation tools/APIs that make automation to express firewall policies across devices promising. An enterprise network may have firewalls with varying capabilities. In order to express the policies for these different firewalls using SAFire, the firewall vendors will have to indicate the SAFire features that map to their capabilities.

3. Overview

SAFire is a text-based, scriptable, self-documenting language designed to express protocol decode rules for firewalls. The language is specified as XML schema to

Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)
0-7695-2141-X/04 $ 20.00 © 2004 IEEE

define the inter-dependent set of filters and actions for protocol decode analysis. SAFire scripts or derivatives can be used to configure a firewall via a local or remote secure interface. The scripts can then be interpreted by a configuration component, which sets up the low-level sub-components of the firewall. One of SAFire's unique capabilities is that it supports autonomic configuration based on packet events. SAFire employs extensive use of string manipulation functions for application layer protocol processing and uses an abstract interface for calling user-defined functions. By design, SAFire allows for algorithmic translation to other firewall languages. SAFire retains features from existing languages such as minimal support for loops, implicit memory allocation, macros and functions with no recursion. Standard XML APIs allows integration of SAFire into a security framework like Checkpoint's OPSEC [11] via the Suspicious Activity Monitoring (SAM) or Content Vectoring Protocol (CVP) APIs. The goal of this language is to be able to express protocol decode rules to control overall protocol behavior. Section 3.1 enumerates the requirements of the language to achieve this goal.

3.1. SAFire requirements

The grammar has the following requirements:
R.1. Packet data extraction and filtering
R.2. Packet data, variables, constants declaration
R.3. Context-aware flow state table management
R.4. Multiple non-conflicting actions lists
R.5. Outsourcing policy decisions for specific flows
R.6. Express application layer information within rules
R.7. Definition of templates/macros for reuse
R.8. Math, binary, relational and logical operations
R.9. Packet variables like direction, interface, and host
R.10. String data manipulation
R.11. Dynamic rule creation /enabling /disabling
R.12. Implicit memory management
R.13. Support for integer data types of widths 8-64 bits

4. SAFire language details

This section describes SAFire capabilities and cross-references these with the requirements. It describes the functionality of the language elements.

4.1. SAFire functional elements

SAFire rules are built using XML schema elements that fall into the following functional categories:

4.1.1. Data extraction capability. Allows extraction of fixed or variable sized data from packet header or payload that can be saved for later use. The extracted data can be masked before extraction. Fulfills R.1.

4.1.2. String manipulation capability. Provides a rich set of string operation functions with the ability to store extracted strings for later use. Fulfills R.10.

4.1.3. Math operations. Allows basic math functions on integers of sizes 8-64 bits. The results can be stored in symbols for later use. Fulfill R.8.

4.1.4. Filter management. Allows dynamic creation, removal and modification of n-tuple bi-directional filters. Fulfills R.11.

4.1.5. State table management. Here, we introduce our concept of a General Protocol State Machine or GPSM. A PSM is a Deterministic Finite Automaton (DFA), which has a start state, multiple intermediate states, and one or more terminating states. The following seven states have been defined in SAFire to capture protocol state transitions: Suinit (Un-initialized), Sinit (Initialized), Scest (Connection Established), Sde (Data Exchange), Sctd (Connection Termination), Sterm (Termination), and Sabort (Abort). Protocol conditions or events cause the PSM to transition from one state to another. This capability in SAFire exposes the functionality to compute unique flow identifiers and based on these identifiers, the ability to create, delete, get or set state and context associated with a flow. Fulfills R.3 and R.12.

4.1.6. Pattern table management. Exposes intrinsics to create and populate a pattern table, and then search for these patterns in a given packet. Fulfills R.6.

4.1.7. Packet-related utilities. Exposes actions to be performed on packets such as allow, deny, copy, redirect or mark. Fulfills R.4 and R.5.

SAFire supports other actions like user-defined functions, generation of alerts, outsourcing or hold/resume of a flow, updating flow context, installation/removal of filters based on past events. The functional elements described above are arranged under *policyrule* and *policyaction* complex types. Autonomic configuration is achieved by recursively including the *policyrule* complex type as one of the attributes of *policyaction*. Such rules can be used for hierarchical protocol decode scripts. Requirements R.2, R.7, R.9 and R.13 are addressed with support for

Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)
0-7695-2141-X/04 $ 20.00 © 2004 IEEE

a declaration section for constants, variables and macros.

4.2 Creating SAFire scripts

This section describes the procedure to create SAFire scripts for analyzing protocol behavior. Firstly, the protocol is expressed in terms of a Deterministic Finite Automata (DFA) that is then appropriately mapped to the seven states described in section 4.1.5. An example of a protocol state machine for tracking active FTP sessions is shown in Figure 4.2.1 below.

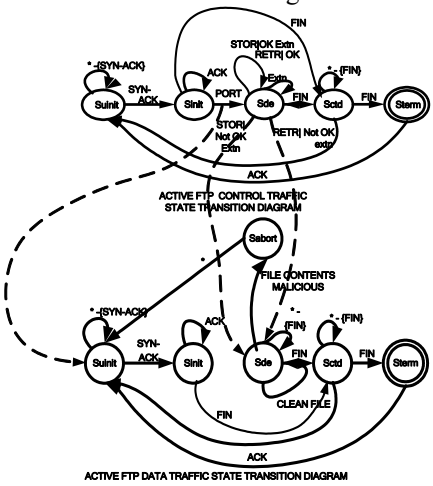

Figure 4.2.1. AFTP State Transition Diagram

Although the entire AFTP state machine as described in RFC 959 [2] can be described in SAFire, we present here only the subset of the state machine that pertains to file transfer tracking. Once that is done, all the transitions from the normalized DFA are extracted, noting the source state, destination state, the event that caused the transition, and any events caused due to the transition. Transition rules are logically grouped and are mapped to SAFire as follows - The transition causing events map to SAFire rules with a check for the source state and the events caused due to the transition map to SAFire action elements, with the state set to the destination state in the action. The resultant set of SAFire rules are then translated (using XSLT) to the firewall specific format.

For example, from Figure 4.2.1, consider the state transition that takes the control flow from Suinit to Sinit upon detection of a TCP packet with SYN - ACK flags set. The start state corresponding to the transition is Suinit, the event causing transition is the detection of SYN-ACK flags set in the TCP packet header and the destination state is Sinit. The corresponding SAFire rule creates a unique flow identifier (hash) in the setup portion of the rule. To compute this identifier, pattern extraction templates are used to extract the 5-tuple data from all TCP packets. The rule's condition elements are then executed. These use other TCP pattern templates (or operations) to check if the TCP ACK and SYN flags are set. If they are, the state associated with the flow in the TCP flow table is updated to Sinit state (by default, all flows are in Suinit state). Section 4.3 gives a more detailed example.

4.3. SAFire examples

SAFire can be used to define a reusable set of scripts for common protocol definitions like TCP/IP. Each script has a name and a count of the rules described in the script. Typically, a script begins with the definitions of constants such as standard protocol values, for example, TCP 6, FTP port 21 and so on. This section of the script also defines pattern templates to be used in the script, for example pattern extraction template for IPv4 source and destination addresses etc. This is followed by the definition of the variables used in the script, for example, for the dynamic FTP port opened, a 16 bit unsigned integer variable is declared. Also associated with the variable is the format of the type as seen in the packet, namely, binary, ASCII, etc., and whether the data is expected in network byte order. The core part of the script is the set of SAFire statements explained in more detail below.

This sample SAFire FTP script:

- Captures outgoing AFTP PORT command packets
- Extracts the data port from the PORT command
- Creates a transient filter using the data port

To start with, the pattern extraction templates are used to extract the 5-tuple data from a flow and the flow identifier is computed. This part of the script is executed for all packets. The script then checks if the packet is an outgoing FTP control packet (i.e. is the destination port 21) with the ASCII data "PORT" at the expected offset, and if it is, it checks if the flow in the Sinit and Sde states. If the conditions are satisfied, the action section of the script defines the treatment to be given to the flow. The base action is to allow this packet, but other associated actions are to extract the port from the ASCII PORT command using the tokenizer action. ASCII data extracted also has to be operated upon using shift operations to store the port value in a variable. Based on the extracted and computed data, new flow states are created for the *expected* data flow and reverse traffic flow. This is also used to create a dynamic filter for these flows. When data traffic starts, the dynamic rules installed will be matched.

Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)
0-7695-2141-X/04 $ 20.00 © 2004 IEEE

Further analysis can be performed on the data traffic as required. The script can finally be programmed to intercept the teardown of the control session and remove the dynamic rules installed for the data session.

5. Prototype Protocol Analysis Engine

This section provides implementation details of our prototype Protocol Analysis Engine (PAE) that executed SAFire firewall scripts. The PAE was implemented on Windows* XP as a Network Driver Interface Specification (NDIS) [14] intermediate driver. The SAFire script was compiled using Apache Xerces-C++ SAX libraries [10] [13] that generated the binary output that was interpreted and executed by the PAE. The PAE works on the principle of following the Protocol State Machine (PSM) of any protocol (FTP, HTTP, etc.), detecting flows that deviate from the defined protocol behavior and taking appropriate actions. Using SAFire, the PSM was expressed as a collection of rules that dictated state transitions and defined corresponding actions. The SAFire parser parsed the SAFire rules and passed it to the PAE via *ioctl()* calls. The PAE was comprised of a packet header based classifier and the filter database. The filter database stored static and transient filters. Static filters were applied to aggregate flows and remained in the filter database until explicitly removed. Transient filters were added and deleted by the PAE core to track per-flow state changes. A flow state table kept track of state changes and was PAE managed.

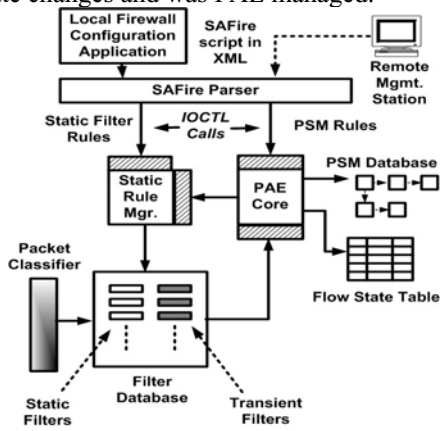

Figure 5.1. PAE architecture

6. Conclusion

With the heterogeneity in firewall capabilities and intrusion detection systems being increasingly integrated into firewall solutions, there is undoubtedly a need for a translatable, powerful language that can effectively express complex firewall behavior. In this paper we have identified the requirements of such a language and proposed a policy definition language, SAFire based on the requirements. The language is XML based, which enables the use of existing authoring and validation tools to write policies and to deploy them using web-based interfaces. A Birds-of-a-feather session was held at the 55th IETF meeting to discuss standards for distributed end-point firewall control. We feel SAFire is a good starting point for this effort.

7. References

[1] William R. Cheswick, Steven Bellovin, and Aviel Rubin, *Firewalls and Internet Security: Repelling the Wily Hacker,* Addison-Wesley, Second Edition.

[2] J.K.Reynolds, J.Postel, "File Transfer Protocol", RFC 959, STD 9, Internet Engineering Task Force, Nov 1998.

[3] "The Science of Intrusion Detection Systems Attack Identification, whitepaper", Cisco Sys, Inc.

[4] *Cisco Secure PIX* Firewall Command References*, Cisco Systems.

[5] *Check Point Reference Manual*, Check Point Software Technologies Ltd.

[6] *PAX Pattern Description Language Reference Manual*, Integrated Device Technology Inc.

[7] *NFR Network Intrusion Detection System* N-Code Guide*, NFR Security Inc.

[8] N. Damianou, N. Dulay, E. Lupu, M Sloman, "The Ponder Specification Language*",* Workshop on Policies for Distributed Systems and Networks, Policy2001.

[9] M. Condell, C. Lynn, J. Zao , "Security Policy Specification Language*",* Internet Engineering Task Force, October1998.

[10] *Xerces C++ Version 2.2.0,* The Apache XML Project.

[11] *Open Platform for Security (OPSEC)*, Checkpoint Software Technologies Ltd.

[12] *XmlSpy - XML Development Environment,* Altova Inc.

[13] *Simple API for XML*, Sourceforge Project SAX.

[14] *Network Driver Interface Specification*, Microsoft Windows XP* Driver Development Kit.