FIRECROCODILE: A Checker for Static Firewall Configurations

Norbert Lehmann
Institut f. Wissenschaftl. Rechnen
Forschungszentrum Karlsruhe
Hermann-von-Helmholtz-Platz 1
76344 Eggenstein-Leopoldshafen
Germany

Reinhard Schwarz
Fraunhofer Institute for
Experim. Software Engineering
Fraunhofer-Platz 1
67663 Kaiserslautern
Germany

Jörg Keller
LG Parallelität und VLSI
FernUniversität in Hagen
Postfach 940
58084 Hagen
Germany

Abstract – *We present* FIRECROCODILE, *a tool to check the static configuration of Cisco PIX firewalls.* FIRECROCODILE *is based on the extensible framework* CROCODILE *and thus is extensible itself. We report on* FIRECROCODILE*'s architecture, its abilities and features, and its relation to other tools. Finally we report on our experiences when analyzing the configuration of the central firewall of a research center with a complex network and application structure.*

Keywords: Firewall, static analysis, access control lists, experimental evaluation.

1 Introduction

A firewall is a local network's gatekeeper towards the Internet, serving to disable unauthorized access to the local network's resources. Such unauthorized access may range from port scans on the computers in the local network to flooding attacks on servers in order to stop their service, to intrusion with stealing or corrupting data, with many more malicious actions reported in the literature. As such, a firewall is an important part of the network's security architecture.

A firewall can only perform its task if it is correctly configured. Yet, standard configurations have to be adapted to the particular network structure and usage, and as the structure and usage is continually changing, the configuration has to be continually adapted as well. Maintenance of configurations of software systems is a difficult business that requires good documentation and care on the side of the responsible administrator. However, firewall configurations are difficult to read and understand because they involve quite complex dependencies and also contain redundant possibilities to express certain features. Therefore manual changes to the configuration are time-consuming and rather error-prone.

For this reason, we conceived and implemented a tool named FIRECROCODILE to analyze the static configuration of a firewall, as specified in the configuration file, thus supporting the job of the firewall administrator and enabling a higher level of security for an organization's local network. As firewalls are very different, we concentrated on the analysis of configurations of the widespread Cisco PIX firewalls. PIX firewalls allow to export their configuration settings in readable textual format, as a sequence of individual configuration clauses. Thus, these configuration files are accessible for static analysis.

As each organization normally has its particular structure and security policy, the checker tool should be configurable as well. In order to support administrators' work, the checker rules should be specified at a higher level of abstraction than a firewall configuration, for example, they should be oriented more towards security policies, and they should build on existing knowledge in the field. Furthermore, the results of the analysis, in particular warnings and errors, should be in a format that supports reasoning about the source of the error, in order to simplify its correction in the firewall configuration file.

In order not to start from scratch, we built upon CROCODILE, an existing checker framework for static router configuration checking. We applied our new FIRECROCODILE prototype to the firewall configuration of a large research center with a complex network structure, at the occasion of a change in equipment. We report on our findings.

The remainder of this article is organized as follows. In Section 2, we summarize the concept and architecture of the CROCODILE framework on which our firewall checker has been built. In Section 3 we report on the necessary adaptations and extensions to CROCODILE in order to check PIX firewall configurations. In Section 4 we report on the application of FIRECROCODILE to analyze a firewall with a complex configuration. In Section 5 we discuss previous and related work. In Section 6 we conclude and give an outlook.

2 The CROCODILE framework

Our firewall configuration checker is based on CROCODILE, an extensible framework for the implementation of static configuration file analyzers, originally conceived for the evaluation of Cisco IOS router configurations [1]. CROCODILE is implemented in the Perl programming language. Due to the similarities between PIX and IOS command syntax, the basic mechanisms already provided by CROCODILE fitted the needs of our PIX checker without significant adaptation. In particular, the pattern-based command line parser, the interface for pluggable checker modules, the sophisticated HTML generator for hypertext evaluation reports, and the available interfaces for the specification of user-defined, high-level evaluation rules were all reusable and provided instant support for simple evaluation tasks.

CROCODILE's capabilities go far beyond purely syntactic analysis:

- CROCODILE analyzes the configuration as a whole, instead of each line in isolation.
- The confusing wealth of evaluation results is presented using various task-specific displays. For example, the tool offers an annotated overview, a differential display relative to an earlier version of the configuration, connectivity charts, and convenient hyperlinks to vendor documentation. A particularly useful and innovative detail analysis is the computation of the sets of packets effectively accepted ("whiteset") or rejected ("blackset") by the access control lists protecting an interface [2].
- Findings are logically grouped to various analysis views, with a view gathering all findings conceptually relating to a certain configuration aspect as defined by the user (e.g., authentication or logging). The configuration clauses covered by a view may be scattered across the configuration, and one clause may contribute to several views. During inspection, the user may focus on any particular view. The view-based presentation makes the reasoning behind the configuration easier to conceive, and consequently makes errors and vulnerabilities more transparent [3].

CROCODILE thus relieves the user from cumbersome low-level work and raises configurations to a semantic level where the human expert is more adequately supported.

2.1 The CROCODILE architecture

The CROCODILE core consists of a parser with a number of pluggable checker modules attached to it (Fig. 1). At their plug-in interface, the modules announce syntax patterns that they are interested in (specified in extended Backus-Naur Format) and corresponding handlers to be called on the occurrence of a pattern. The parser reads a configuration file, searching for the specified syntax patterns. Whenever the input matches some pattern, the input line is split into its components according to the pattern specification, and the handler(s) associated with the matching pattern are invoked with the components of the splitted configuration line as invocation parameters. On invocation, the handler evaluates the given configuration clause, typically in the context of earlier invocations of other handlers and the state changes they triggered in the plug-in module.

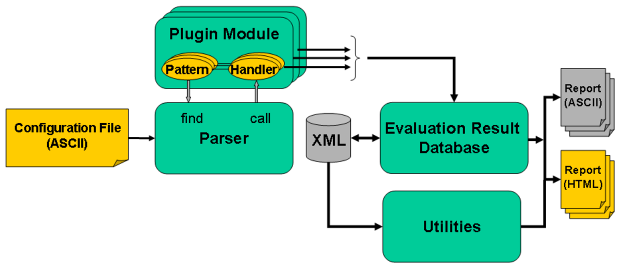

Figure 1: Architecture of the CROCODILE framework

Each plug-in module stores the results of its analysis into the evaluation result database. After the parser has finished reading and all handlers have run to completion, report generators transform the database contents into different types of evaluation reports. Some findings are reported in plaintext format, others are transformed into an intermediate XML format for later postprocessing. For interactive assessment, a comprehensive hypertext report is generated in HTML format (see [1, 3] for detailed examples).

2.2 Using and extending CROCODILE

Users do not normally come into contact with the interface between the parser and the checker modules, even when they construct their own checker modules. The interplay results automatically, as all checker modules are derived from a common base class that provides all the machinery to interface with the rest of the system. To add new evaluation functionality to the framework, a user derives a new plugin from the pluggable module base class, specifies the relevant patterns in BNF and codes the corresponding handlers that implement the evaluation logic.

But CROCODILE offers an even more convenient access to evaluation capabilities: Some plug-in modules contain generic rule interpreters that read evaluation rules from a configuration file, automatically derive the required patterns and handlers from the symbolic rule specifications, announce these patterns to the parser and perform the corresponding evaluation tasks with their predefined handlers. Instead of low-level handler programming, evaluation tasks may thus be specified as high-level checker rules. Using high-level specifications is more convenient and less error prone, but of course the high-level rule format restricts the user in her ability to specify arbitrarily involved evaluation steps. The user may formulate new evaluations rules in any of the available high-level formats (see Sections 2.3.1, 2.3.2, and 2.3.4 for examples), or write a new checker module from scratch, depending on the complexity of the evaluation task.

2.3 Available CROCODILE plugin modules

The original CROCODILE framework offers a number of predefined plugin modules, each performing a specific set of evaluation tasks. Some of the modules are specifically tailored to IOS, but several modules provide capabilities that are useful in both IOS and PIX contexts. We give details of these modules in the sequel.

2.3.1 RATemulation This module emulates the Router Audit Tool (RAT), a utility for benchmarking IOS router or PIX firewall configurations with respect to a given set of security requirements (cf. Section 5.1). Requirements are specified by the user as configurable rulesets. Predefined rulesets can be downloaded from the Internet and adapted to local security policies.

The plug-in module takes original RAT rulesets as input and faithfully applies the specified evaluation criteria to the configuration under test. In addition to customary CROCODILE views, the emulation module produces reports equivalent to those of original RAT. RATemulation enables CROCODILE to participate in future advances of the RAT evaluation rulesets.

2.3.2 CompoundPatterns The most versatile checker module delivered with CROCODILE is the CompoundPatterns module. It supports user-defined evaluation criteria for the (non)existence of certain configuration clauses. The user may specify arbitrary syntax patterns and, for each pattern, assign a severity to (1) the occurrence of this pattern in the configuration and (2) the omission of the pattern. Depending on the user's choice, CROCODILE will generate a configurable OKAY, INFO, CHECK, WARN, or ALERT message in each case. A pattern may be either a basic pattern referring to a single configuration clause, or a compound consisting of several basic patterns linked, for example, by logical AND, OR, XOR, NOT, or IF-THEN-ELSE operators, or even by FOREACH and EXISTS quantifiers with bound variables. The CompoundPatterns plugin recognizes so-called configuration modes (e.g. IOS 'interface' mode), and patterns may be restricted to be applicable in certain configuration modes only.

Fig. 2 shows an example of typical macro and IOS compound pattern specifications. The compound pattern requires that for all interfaces except 'Loopback' or 'Null' interfaces, the interface must be either disabled ("shutdown"), or else the Cisco Discovery Protocol (CDP) must be disabled — either local to the interface context ("no cdp enable"), or at global level for all interfaces("no cdp run"). If this condition is met, CROCODILE generates an OKAY message ('o') "CDP disabled"; otherwise, a WARN message ('w') "CDP enabled" is issued and attached to the configuration clause that violates the CDP policy.

Typical evaluation problems covered by CompoundPatterns are, for example, checking that (un)desired services and modes are explicitly (de)activated, that certain commands are assigned the required level of privilege, or that passwords are assigned and sufficiently securely encrypted.

```
INTERFACE ::=   { int | interface }

INTERFACE !Loopback !Null ... {
 o "CDP disabled"
 w "CDP enabled"
     if (NOT shutdown) {
    no cdp enable | Global(no cdp run)
     }
} INTERFACE ...
```

Figure 2: A simple CompoundPattern

By using quantified patterns with bound variables, we can even express relationships that require comparisons between specific attributes of multiple pattern instances, for example, that an arbitrary name found in use has been properly defined elsewhere. This yields a very intuitive way to describe many simple properties of a configuration. Our experience has shown that compound patterns contribute significantly to a large evaluation coverage. Nevertheless, an analysis of the complex aspects of a configuration needs to dig deeper than this. To this end, CROCODILE provides the more specific modules described next, which offer advanced capabilities at the expense of less flexibility in module configuration.

2.3.3 Connectivity This module extracts from the configuration information about the topology of the component's network neighborhood with its surrounding subnets and nodes. While it does report some irregularities and potential weaknesses, the focus of this module is not on security checking, but on extracting and deriving topology information for later processing by other modules or for manual inspection. The reconstructed topology data are displayed in a graphical output format, as a so-called connectivity chart. Taking the case of an interface as an example of the extracted information, the module would list which IP protocols are routed through this interface as inbound and outbound traffic to and from which subnets or hosts.

2.3.4 IngressEgress This module verifies that the configuration of access control lists (ACLs) and network interfaces conforms to the desired behavior as specified by the CROCODILE user. The user may specify blacksets and whitesets for these objects, and the IngressEgress module will list any violations found in the ACL or interface configuration, relative to these sets.

Fig. 3 shows an example of a user-defined egress restriction. Note that format and order of independent egress clauses is irrelevant: IngressEgress computes the precise meaning — i.e., blackset and whiteset — of the specification, and checks whether the evaluation target is at least as restrictive as the blackset, and open at least for the whiteset.

```
INGRESS_EGRESS_INTERFACE ::= Serial {1|2}
NETMASK  ::= 0.0.0.255
LOCALNET ::= 157.106.153.0 NETMASK

!  RFC1918 outbound address filtering

interface INGRESS_EGRESS_INTERFACE out {
   deny   ip any 10.0.0.0   0.255.255.255
   deny   ip any 127.0.0.0  0.255.255.255
   deny   ip any 172.16.0.0 0.15.255.255
   deny   ip any 192.168.0. 0 0.0.255.255
   permit ip LOCALNET       any
}
```

Figure 3: A simple IngressEgress restriction for IOS

2.3.5 CiscoLinks This plugin does not perform any analysis, but assigns to each configuration clause the appropriate hyperlinks to Cisco's Online Documentation. To this end, each configuration command is looked up in a list containing all commands, and for each command a list of URLs to relevant manual pages.

The list of Cisco links can be generated semi-automatically by traversing the HTML index pages of the Cisco Online Documentation.

3 Deriving a PIX-specific firewall analyzer from CROCODILE

We used the modules described in Section 2.3 as a starting point for the development of a PIX-specific evaluation functionality. In this section, we discuss the changes required to these modules, and how they can be used in a PIX environment. We also sketch where additional modules were necessary to test certain configuration aspects of firewalls that have no counterpart in IOS routers.

3.1 Adapting existing modules

3.1.1 RATemulation The Router Audit Tool (RAT) (cf. Section 5.1) is already emulated in CROCODILE. Therefore it was sufficient to download and install the RAT version 2.2 ruleset, which also provides some evaluation rules for PIX firewall configurations. Obviously, each organization must adapt this rule set to their local situation.

3.1.2 CompoundPatterns As the module CompoundPatterns allows to specify arbitrary syntax patterns, no changes to the module itself were necessary. Obviously, PIX-specific syntax patterns have to be specified to apply this module to firewall configurations.

We defined a set of PIX patterns on the basis of usage guidelines and recommendations contained in Cisco's 'PIX Firewall Configuration Guide' and 'PIX Firewall Command Reference'. Another important source for evaluation criteria covered by CompoundPatterns is the individual security policy of an organisation. Since the patterns are easy to create, and processing with the FIRECROCODILE parser is very fast, one can implement as many evaluation rules as needed. [1]

An example is the following pattern to recognize filter rules allowing full access to computers behind the firewall:

```
a  "avoid unlimited inbound permissions"
access-list acl_outside permit ip any ...
```

3.1.3 ConnectivityPIX The code of the original IOS Connectivity module directly refers to IOS configuration commands. Thus, a new module ConnectivityPIX had to be devised. As the relevant IOS and PIX commands are largely similar, the re-implementation of the module required only small adaptations of the code to the syntax and semantics of the appropriate PIX commands. Yet, as a consequence, some of the data container classes of CROCODILE, such as those for access lists and interfaces, had to be re-implemented as well. As some of the relevant PIX commands have no counterpart in IOS, we had to add some routines to handle these commands. Most effort was required to support PIX object groups in ACLs and the close coupling of ACLs and network address translation (NAT) that is characteristical for PIX firewalls.

3.1.4 IngressEgress Similarly to the previous module, IngressEgress also had to be re-implemented to IngressEgressPIX. As IngressEgress builds upon Connectivity, IngressEgressPIX could build upon ConnectivityPIX. Therefore, the necessary changes were relatively small. Changes had to be made to consider the "security level" of the interfaces. Without an explicit ACL, all traffic from lower to higher security levels is forbidden and all traffic from higher to lower interfaces is permitted on a PIX firewall.

3.1.5 CiscoLinks CROCODILE uses a perl script to generate an index file from the online IOS command reference provided by Cisco. The online PIX command reference differs in structure from the IOS reference, thus the perl script had to be adapted. As the index file has the same format as before, changes to the CiscoLinks module itself were not necessary.

3.2 Additional modules

An important functionality of PIX firewalls that is significantly different in IOS routers is network address translation (NAT). As mentioned earlier, there is a close connection between ACLs and NAT on PIX firewalls. In particular, if a

[1] For our tests with the prototype tool, we specified only 26 compound patterns. This is a small rule set compared to the orginal CROCODILE distribution, which provides a default rule base of about 200 compound patterns, some of which contain nested quantifiers yielding compounds of substantial complexity.

computer behind the firewall should be reachable from outside, we have to configure a static NAT for this computer. This part of the configuration is covered by the module InboundPIX. InboundPIX checks if there are appropriate translations for each of the reachable computers.

Another important aspect is the VPN configuration of a PIX firewall. A module covering VPN issues would be worthwhile, but has not been implemented yet.

4 Experimental results

We used FIRECROCODILE to analyze the configuration of the Internet firewall of the German Research Center at Karlsruhe (Forschungszentrum Karlsruhe). The Research Center has approximately 3,800 employees and more than 10,000 computers connected to the LAN, subdivided into 240 subnets. The research activities of Forschungszentrum Karlsruhe cover a total of 11 programs in five research areas. There are close cooperations with research centers, universities and industrial companies in 47 countries all over the world. Thus, the network and application structure is quite complex. The PIX configuration file consisted of about 1,800 command lines, among which about 1,000 were filtering rules.

We noticed that the modules ConnectivityPIX and IngressEgressPIX, which are known to be most time-consuming, were able to handle even quite large ACLs. In IngressEgressPIX, the running time mainly depends on length and structure of the specified blackset/whiteset. We analyzed the given configuration (with ACLs consisting of overall about 1000 lines, as mentioned above) for compliance with an IngressEgressPIX specification comprising 20 rules. In these experiments, the running time was 5 to 10 minutes on a PC with a Pentium 4 processor with 3 GHz, 512 MByte RAM, SuSE Linux 9.2 operating system, and Perl 5.8.5, depending on the structure of the specified blackset/whiteset under study.

The module ConnectivityPIX revealed that 20 of the 1,000 filtering rules were void, as they were completely covered by other rules. Hence, it served to simplify the rule structure and potentially avoid future errors due to redundant rules. The connectivity chart produced by this module presented an overview of configured routes and the services available at internal computers. The connectivity chart replaced the previous documentation that had been created manually, which is an error-prone process.

The module CompoundPatterns found 7 filtering rules that rendered internal computers unconditionally reachable from the Internet without any access restrictions. Those rules had once been inserted as a test or a quick fix, and had not been removed or documented.

The module RATemulation hinted to activate the 'floodguard' of the PIX firewall, which until then had been inactive. Floodguard is a mechanism to protect the PIX firewall against denial-of-service attacks.

In summary, FIRECROCODILE proved helpful in finding residual weaknesses in a complex firewall configuration, although this configuration had been actively maintained and documented by manual work prior to our tool check. Beyond the unveiling of potential weaknesses, for a new employee, taking over the administration of the firewall, using this tool provided a comprehensive and quick insight into the given configuration.

5 Related Work

The static analysis of firewall configurations is a field of active research. A number of tools have been developed that assist the human expert in interpreting configuration settings, locating potential misconfigurations, and removing configuration defects and weaknesses. Most approaches (e.g., [4, 5, 6, 7, 8, 9, 10]) only address the configuration of filtering rules, while other aspects of a firewall configuration — e.g., configuration clauses that control network address translation, encryption, time synchronization, logging — are out of scope. In contrast to that, the CROCODILE framework has powerful mechanisms for checking filtering properties [2], but it is equally capable of evaluating general network management and security policy aspects of a firewall configuration.

5.1 RAT

The Router Audit Tool (RAT) is probably one of the best-known tools for router configuration checking [11]. Since version 2.2, RAT also supports evaluation rulesets for PIX firewalls. The tool is freely available from the Center for Internet Security [12].

The RAT tool checks configuration files against configurable checking rules mirroring network security best practice. A RAT rule consists of a pattern, which is declared as either required or forbidden. The rule is considered as "pass" or "fail" depending only on whether or not the corresponding pattern appears in the configuration text. RAT is thus limited to line-by-line comparisons at a purely syntactical level. Such a naive approach lacks the essential abilities for deeper analysis, as it is required for multi-clause configuration settings, or for the analysis of firewall filters. While RAT is not really capable of semantic analyses of access control lists, its evaluation covers diverse configuration aspects beyond firewall filter settings.

5.2 Lumeta and AlgoSec Firewall Analyzers

The Lumeta Firewall Analyzer (LFA) [4] simulates the filtering behavior of a firewall with graph algorithms. A query engine operates on a topology graph representing different network zones and gateways between them, and answers user queries concerning the reachability of certain services on given nodes in specific zones. The LFA presents a graphically appealing HTML view of accepted and rejected incoming and outgoing packets, itemizing IP services, hosts, and networks. Unlike CROCODILE, which computes a symbolic representation of the exact blackset and whiteset of the firewall filters, the LFA explores the blackset and whiteset by continuously trying out different combinations of source address, destination address, and IP service, including the use of wildcards. In this way, dead or contradictory filtering rules cannot be identified. Unfortunately, hardly any substantiated assertion about speed, completeness, and generality of the LFA algorithm can be made, because [4] and [9] give no adequate performance data and there is no further information known to us.[2]

The AlgoSec Firewall Analyzer from Algorithmic Security Inc. is a commercial tool based on LFA [13].

5.3 Expert system for analyzing firewall rules

Eronen and Zitting developed another tool for analyzing access control lists. Their approach is to translate ACL clauses to facts expressed in a Prolog-like programming language [5], and to infer properties of the ACL from these facts.

In essence, ACL clauses are represented as 6-dimensional hypercubes, with numerical intervals restricting each dimension. These dimensions cover protocol, addresses, ports, and packet flags. In addition to ACL information, a so-called knowledge base is supplied with further topology related information. An inference engine then answers queries regarding the filtering behavior of the ACL under consideration. By taking into account routing information from the knowledge base, queries cannot only be made on ACLs themselves, but also on the network data flows controlled by these ACLs. This resembles CROCODILE's analysis that uses routing information to determine those address ranges that can be reached via a given interface.

A query is specified as a Prolog-like goal expression. The inference engine will find all possible substitutions for the free variables in a given goal that are consistent with the facts stored in the knowledge base. A typical goal for ACL analysis is, for example: "Find all ACL clauses that are never matched". This will yield a characterization of all dead clauses. Eronen and Zitting do not explicitly mention blackset/whiteset computations as a general means for ACL analysis, but in principle, arbitrarily complex queries may be specified as long as the knowledge base contains sufficient facts to determine a solution. A challenging problem with the inference engine is the retranslation of query results into a clear evaluation report. To present evaluation results in a format and order meaningful to the average firewall administrator, substantial retranslation effort may become necessary.

[2] We tried to contact the authors but received no reply. Meanwhile the firewall analyzer is a commercial product and some details of its inner working are apparently considered business secrets.

5.4 Equant ACL consistency checker and Firewall Policy Advisor

The ACL consistency checker (VACL) developed by Equant [6] and the Firewall Policy Advisor (FPA), developed by Al-Shaer and Hamed [7] both avoid any complexities caused by a comprehensive analysis of a complete filtering set, but are restricted to the computation of rule interference between pairs of filtering rules only. Considering only rule pairs simplifies the computation substantially. However, the analysis misses some types of interference, for example situations where a rule is "dead" because it is completely "shaded" by the combined effect of a preceding subset of rules — but not by any single predecessor.

5.5 Binary decision diagram for ACL analysis

Hazelhurst presents a prototypical tool for interactively querying the filtering behavior of an ACL in [8]. To this end, the ACL clauses are represented as a so-called reduced Binary Decision Diagram (BDD). This yields a unique characterization of the ACL, comparable to our blackset/whiteset representation.

Similar to the expert system described in Section 5.3, the tool answers questions about accepted and rejected packets of a specific ACL. The tool allows to calculate the dominance relation between two given ACLs, that is, whether two ACLs are equivalent, or one implies the other. Based on their BDDs the net difference between two ACLs or between an altered ACL and its previous version can also be computed, that is, their differential blackset and whiteset.

Making queries to the system is complicated by the use of boolean expressions to construct and query the BDDs. The same holds for the interpretation of query results: Suitably retranslating complex boolean expressions back into ordinary ACL syntax is far from trivial. In fact, [8] mentions the presentation problem and the inefficient implementation of the tool, as well as missing experience with real-world input data as the most important areas for further study.

5.6 Generating firewall configurations

An interesting alternative to the analysis of existing firewall configurations is their synthesis. By deriving the configuration from a high-level specification and by automatically generating the low-level configuration clauses of a specific firewall device, many of the common configuration errors can be avoided. In [10] Prandini presents a prototype tool that generates the configuration file for iptables firewalls. In this approach, the correctness of the firewall configuration rests mostly on the clarity of the abstract specification format. The tool's validation functionality is limited to simple policy syntax checking. Like most other tools, Prandini's prototype only considers filtering and forwarding aspects, but does not cover other configuration settings that may contribute to the implementation of a security policy.

6 Conclusion

We presented the concept and prototype of FIRECROCODILE, a tool to analyze the static configuration of a PIX firewall. The tool builds upon CROCODILE, a framework for the analysis of IOS router configurations. In contrast to previous work, FIRECROCODILE does not process the configuration file literally, i.e. line by line, but builds a configuration representation from the file and analyzes this representation. It allows to check for misconfigurations and for violations of policies, which can be formulated very freely. Our tool has been applied to a complex firewall configuration that was actively maintained and documented. Still, FIRECROCODILE identified several configuration errors and weaknesses. This indicates that beyond a certain network size, firewall configurations are too complex to be maintained manually, and that tool support on an advanced level, as supplied by FIRECROCODILE, is a necessity.

In the future, we would like to implement further modules to refine and extend the evaluation capabilities of our tool. Also, Cisco meanwhile released PIX version 7.0, which differs from version 6.3 that our prototype currently supports. Hence we plan to update FIRECROCODILE to the new PIX version.

Acknowledgment

The authors would like to thank Fraunhofer IESE for providing access to the source code of CROCODILE, and Forschungszentrum Karlsruhe for providing the real-world test case.

References

[1] H. Peine, R. Schwarz, T. Schwenkler, and K. Simon, CROCODILE *User Manual*, 3rd ed., Fraunhofer Institute Experimental Software Engineering (IESE), Kaiserslautern, Germany, Dec 2004. [Online]. Available: http://www.iese.fraunhofer.de/crocodile

[2] H. Peine, R. Schwarz, and T. Schwenkler, "Understanding the true effect of ip access control lists," Fraunhofer Institute Experimental Software Engineering (IESE), Kaiserslautern, Germany, Tech. Rep. IESE Report No. 36.04/E, Apr 2004. [Online]. Available: http://bib.iese.fhg.de/reports/public/2004/iese-036_04.pdf

[3] H. Peine and R. Schwarz, "A multi-view tool for checking the security semantics of router configurations," in *Proc. 19th Annual Computer Security Applications Conference (ACSAC 2003)*, Las Vegas, Nevada, Dec 2003. [Online]. Available: http://www.acsa-admin.org/2003/papers/34.pdf

[4] A. Wool, "Architecting the lumeta firewall analyzer," in *Proc. 10th USENIX Security Symposium*, Washington, D.C., USA, Aug 2001. [Online]. Available: http://www.usenix.org/publications/library/proceedings/sec01/full_paper%s/wool/wool.pdf

[5] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," in *Proc. 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, Copenhagen, Denmark, Nov 2001. [Online]. Available: http://www.niksula.hut.fi/~peronen/publications/nordsec_2001.pdf

[6] D. Valois and C. Llorens, "Identification of security holes in router configurations," in *Proc. 14th Annual Forum of Incident Response and Security Teams Conference (FIRST 2002)*, Hawaii, USA, Nov 2002.

[7] E. S. Al-Shaer and H. H. Hamed, "Firewall Policy Advisor for Anomaly Detection and Rule Editing," in *Proc. IEEE/IFIP 8th Intl. Symp. Integrated Network Management (IM 2003)*, Mar 2003, pp. 17–30. [Online]. Available: http://www.mnlab.cs.depaul.edu/~ehab/papers/im03-cr.pdf

[8] S. Hazelhurst, "Algorithms for analysing firewall and router access lists," University of Witwatersrand, Johannesburg, South Africa, Tech. Rep. TR-Wits-CS-1999-5, Jul 1999. [Online]. Available: ftp://ftp.cs.wits.ac.za/pub/research/reports/TR-Wits-CS-1999-5.ps.gz

[9] A. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *Proc. 21th IEEE Symp. on Security and Privacy*, Oakland, CA, May 2000. [Online]. Available: http://www.eng.tau.ac.il/~yash/sp00.ps

[10] M. Prandini, "A multi-platform toolkit for the configuration of packet-filtering firewalls," in *Proceeding (499) IASTED Intl. Conference on Communication, Network, and Information Security (CNIS 2005)*, Phoenix, AZ, Nov 2005, pp. 146–153.

[11] B. Stewart, *Router Audit Tool: Securing Cisco Routers Made Easy!*, 1st ed., The SANS (SysAdmin, Audit, Network, Security) Institute, Bethesda, MD, Mar 2002. [Online]. Available: http://www.sans.org/rr/netdevices/cisco_RAT.php

[12] The Center for Internet Security, "Router Audit Tool," Webpage, Oct 2003. [Online]. Available: http://www.cisecurity.org

[13] Algorithmic Security Inc., "AlgoSec Firewall Analyzer," Webpage, 2003-2006. [Online]. Available: http://www.algosec.com/Products/FA/