# Analysis of Vulnerabilities in Internet Firewalls

Seny Kamara, Sonia Fahmy, Eugene Schultz, Florian Kerschbaum, and Michael Frantzen
Center for Education and Research in Information Assurance and Security (CERIAS)
Purdue University
656 Oval Dr., West Lafayette, IN 47907–2039, USA
E-mail: seny@cs.jhu.edu, fahmy@cs.purdue.edu, eeschultz@lbl.gov, kerschf@cs.purdue.edu,
frantzen@nfr.net

## Abstract

Firewalls protect a trusted network from an untrusted network by filtering traffic according to a specified security policy. A diverse set of firewalls is being used today. As it is infeasible to examine and test each firewall for all possible potential problems, a taxonomy is needed to understand firewall vulnerabilities in the context of firewall operations. This paper describes a novel methodology for analyzing vulnerabilities in Internet firewalls. A firewall vulnerability is defined as an error made during firewall design, implementation, or configuration, that can be exploited to attack the trusted network that the firewall is supposed to protect. We examine firewall internals, and cross reference each firewall operation with causes and effects of weaknesses in that operation, analyzing twenty reported problems with available firewalls. The result of our analysis is a set of matrices that illustrate the distribution of firewall vulnerability causes and effects over firewall operations. These matrices are useful in avoiding and detecting unforeseen problems during both firewall implementation and firewall testing. Two case studies of Firewall-1 and Raptor illustrate our methodology.

## Keywords

Internet firewalls, firewall vulnerability analysis, firewall testing

## I. Introduction

FIREWALLS have become central fixtures in secure network infrastructures. Firewalls are software and hardware systems that protect intra-networks from attacks originating in the outside Internet, by filtering and managing Internet traffic. Despite their critical role, firewalls have traditionally been tested without well-defined and effective methodologies. Currently, a diverse set of firewalls is being used. Because it is infeasible to examine each firewall separately for all potential problems, a general mechanism is required to understand firewall vulnerabilities in the context of firewall operations.

The firewall data flow model we presented in [1] gives an overall description of firewalls by detailing the operations they perform (depicted in figure 1). When a packet is received by a firewall, it first undergoes link layer filtering. Then it is checked against a dynamic rule set. The packet then undergoes packet legality checks and IP and port filtering. Finally, network/port address translation is performed. Sophisticated firewalls also reassemble packets and perform application level analysis. After a routing decision is made on the packet, out-bound filtering may also be performed. Each of these operations is optional, and the order in which the packet traverses them may also differ in different firewalls. Our model, however, is flexible enough to describe the operations carried out by any firewall, regardless of its type (packet filter or application-level (as illustrated in figure 2) and its implementation.

A firewall vulnerability is an error, weakness, or an invalid assumption made during firewall design, implementation, or configuration, that can be exploited to attack the trusted network the firewall is supposed to protect. For example, common firewall vulnerabilities and misconfigurations include: (1) ICMP allowed, e.g., the firewall can be *ping*-ed; (2) *Denial* rather than *drop* of traffic to ports the firewall is blocking. This provides the attacker with additional information, or improves the speed of the attacker's port scan; (3) Misconfiguration that allows a TCP ping of internal hosts with Internet-routable IP addresses (e.g., in-bound TCP 80 is not restricted to the web server); (4) Trust of certain IP addresses; (5) Availability of extra/unnecessary services on the firewall; (6) Unnecessarily open TCP and UDP ports; and (7) Firewalls that resolve to "firewall.client.com," "gateway.client.com," or something equally conspicuous.

A number of taxonomies [2], [3], [4], [5] provide high level descriptions of the classes of weaknesses that result in software vulnerabilities. In this paper, we analyze several publicly disclosed firewall vulnerabilities by examining the

—-Seny Kamara is currently with Johns Hopkins University; Eugene Schultz is with Lawrence Berkeley Lab; and Michael Frantzen is with Network Flight Recorder (NFR) Security

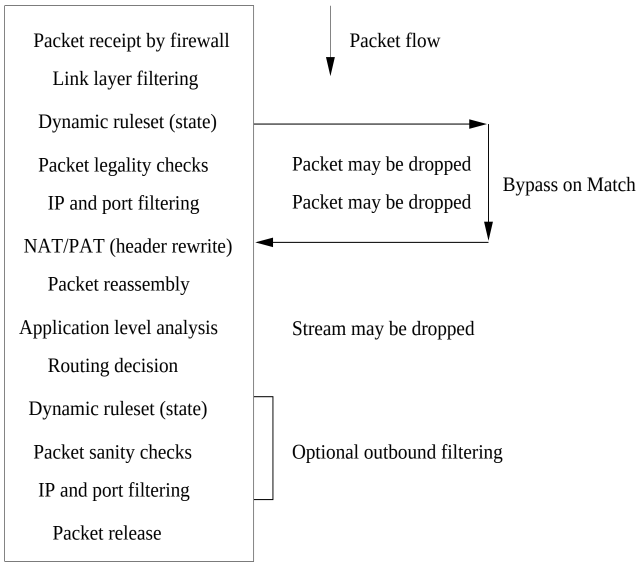

Fig. 1. Firewall operations and data flow

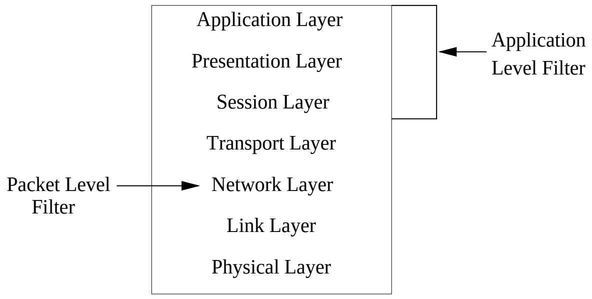

Fig. 2. Firewall layers

errors that caused them, the effect they have on the system, and the firewall operations in which they occur. We then use this analysis to tackle the following questions: (1) Where do certain types of vulnerabilities occur most often and what kind of effect do they usually have on the system? (2) Which operations are traditionally more vulnerable to which kinds of errors? and (3) What kinds of attacks usually result from a particular type of error being found in a a specific operation? We believe that a systematic and formal approach to answering such questions is important, and can be useful in both the development and testing of firewalls.

The remainder of this paper is organized as follows. Section II discusses related work. Section III explains the vulnerability taxonomy we used in our analysis of firewall vulnerabilities (the complete analysis is in the appendix). Section IV presents a set of vulnerability matrices that we constructed from our vulnerability analysis, and discusses some of the conclusions we can infer from them. Section V demonstrates how our matrices can be useful to firewall developers and testers, using two case studies. Finally, section VI summarizes our conclusions and discusses future work.

## II. Related Work

Firewall testing has different goals, including determining if the firewall is a correct implementation of the firewall security policy, how well the firewall resists particular types of attacks, if leakage occurs in the security perimeter created by a firewall, if the logging capability is adequate, if the firewall has the ability to send alarms, and if the firewall can hide information and addresses from the internal network it protects [6], [7]. In this section, we survey prior work that has examined firewalls and firewall testing. The basic design of a firewall and sample firewall algorithms are discussed in [8], [9], [10], [11], [12], [13]. Schuba [14] formalizes firewalls using hierarchical colored Petri nets. Bellovin [15] recently proposed a distributed approach to Internet firewalls. In this approach, personal firewalls are installed at the hosts themselves. The advantages of this distributed approach include detection of attacks within the internal network, detection of more application level attacks, and speeding up firewall functions.

In [16], Haeni describes a methodology to perform firewall penetration testing. The testing steps include indirect information collection, direct information collection, attacks from outside, and attacks from inside. Attack approaches are based on the type of firewall. For packet filtering firewalls, the attacks include blind IP spoofing, non-blind IP spoofing, source porting and source routing. For application level firewalls, the attacks are on bad security policy, poorly implemented policies, SOCKs incorrectly configured, brute force attacks, and enabled services/ports.

Another firewall testing methodology is presented in [17]. The motivation behind this work is that field testing is currently performed using simple checklists of vulnerabilities without taking into account the particular topology and configuration of the firewall target operational environment. A firewall testing methodology is proposed, based on a formal model of networks that allows the test engineer to model the network environment of the firewall system; to prove formally that the topology of the network verifies the sufficient conditions for protection against attacks; and to build test cases to verify that protections are actually in place.

In [18], firewall security and performance relationships are explored. Experiments are conducted to classify firewall security into seven different levels and to quantify their performance effects. These firewall security levels are formulated, designed, implemented and tested under an experimental environment in which all tests are evaluated and compared. Based on the test results, the effects of the various firewall security levels on system performance with respect to transaction time and latency are measured and analyzed. It is interesting to note that the intuitive belief that more security would result in degraded performance does not always hold.

In addition, the Internet engineering task force (IETF) has examined Internet firewalls. RFC 2647 [19] extends the terminology used for benchmarking routers and switches with definitions specific to firewalls. Forwarding rate and connection-oriented measurements are the primary metrics used in the RFC. RFC 2979 [20] defines behavioral characteristics and interoperability requirements for Internet firewalls. The RFC observes, as we will discuss in this paper, that firewall behavior is often either unspecified or under-specified, and this lack of specificity often causes problems in practice. Requirements specification makes the behavior of firewalls more consistent across implementations and in line with accepted protocol practices. This paper continues this effort.

## III. Vulnerability Classification

To analyze and classify firewall vulnerabilities, we need a vulnerability taxonomy that is suited for analyzing firewall systems. Several vulnerability taxonomies have been proposed in the literature, including [2], [21], [5], [3], [4]. Bishop's [5] and Du and Mathur's [3], [4] taxonomies are the most recent and succeed in avoiding some of the ambiguities in other taxonomies. Though Bishop's taxonomy [5] is well defined and exhaustive, it was developed for intrusion detection systems, and is difficult to use in the context of firewall vulnerabilities. On the other hand, Du and Mathur's taxonomy [3], [4] was designed to categorize vulnerabilities in any software system, and is flexible and adaptable to firewalls. We also found it more intuitive, and easier to work with in practice. For these reasons, we will use Du and Mathur's taxonomy in our analysis of vulnerabilities.

Du and Mathur argue that each vulnerability has a cause, an effect on the system, and a fix that corrects the problem. We categorize each firewall vulnerability according to causes and effects, but omit fixes. We make this omission because an analysis by fix requires access to the sections of source code where the vulnerabilities occur. Commercial developers are reluctant to disclose implementation details of their products, so the publicly available vulnerability descriptions omit any reference to specific implementation details.

We proceed to summarize the Du and Mathur taxonomy from [3], [4] (with minor changes), and provide some examples of how the categories can be interpreted in the context of an Internet firewall. The most common vulnerability causes include:

- **Validation error:** A validation error occurs when the program interacts with the environment without ensuring the correctness of environmental data. There are three types of environmental data that need validation: *input, origin,* and *target*. **Input** validation ensures that the input is as expected. This includes the number, type and format of each input field. **Origin** validation ensures that the origin of data is actually what it is claimed to be, e.g., checking the identity of the IP source. **Target** validation ensures that the information goes to the place it is supposed to. This includes ensuring that protected information does not go to an untrusted target.

- **Authorization error:** An authorization error (called authentication error in the origin Du and Mathur taxonomy) permits a protected operation to be invoked without sufficient checking of the authority of the invoking agent.

• **Serialization/Aliasing error:** A serialization error permits the asynchronous behavior of different system operations to be exploited to cause a security violation. Many time-of-check-to-time-of-use flaws fall into this category. An aliasing flaw occurs when two names for the same object can cause its contents to change unexpectedly, and, consequently, invalidate checks already applied to it.

• **Boundary checking error:** A boundary checking error is caused by failure to check boundaries and ensure constraints. Not checking against excessive values associated with table size, file allocation, or other resource consumption, leads to boundary checking errors. Buffer overflow is a result of a boundary checking error.

• **Domain error:** A domain error occurs when the intended boundaries between protection environments have "holes." This causes information to implicitly leak out.

• **Weak/Incorrect design error:** A design error occurs can be traced to the system design phase. For example, a weak encryption algorithm falls into this category.

• **Other error:** Any error that does not fall into any of the above categories.

Note that this is not a comprehensive or complete list– we only include the most common causes that lead to firewall vulnerabilities.

Vulnerability effects include:

• **Execution of code:** This occurs when a vulnerability can lead to code being illegitimately executed. This includes, but is not limited to, code written by an attacker.

• **Change of target resource:** This occurs when a vulnerability allows the state of a resource to be illegitimately changed by an attacker. A resource could be a host, a firewall rule table, or any entity that should be protected by the firewall.

• **Access to target resource:** This occurs when a vulnerability allows an attacker illegitimate access to a target resource. Again, a resource may be any entity that is protected by the firewall. Examples of this vulnerability effect include allowing an attacker to read the firewall rule tables, or to find out which services are available on a protected host.

• **Denial of service (DoS):** This occurs when a vulnerability is exploited to disrupt a service provided to legitimate users. Services in the context may range from packet forwarding or network address translation to administration.

Note that in some instances, one vulnerability effect may be implied by others. For example, changing the state of a target resource implies access to that resource. While this might be true, it does not necessarily imply *unauthorized* access to that resource, which is what we are interested in. Another difficulty is that vulnerabilities cannot always be accurately and exclusively described by a single category. For example, a vulnerability can be caused by a combination of an input validation error and an authorization error, or it can cause an unauthorized execution of code which, in turn, leads to a DoS attack. In these cases, we establish which category is the most immediate and classify the vulnerability accordingly. This means that, in the context of this vulnerability, if we can establish a causal relationship between the two categories, we will designate the cause as the most immediate. The reason we chose to do this is that, in many cases, the category caused by the other may or may not apply, and it is impossible to determine when it will apply. Using the previous examples, if the authorization error leads to an attacker being able to input invalid data that is not correctly checked, then the authorization error is more immediate. If the unauthorized execution of code leads to an attacker being able to carry out a DoS attack, the execution of code error is the most immediate.

## IV. Vulnerability Analysis

In the appendix, we analyze twenty reported firewall vulnerabilities from vulnerability databases and reports. These include the MITRE common vulnerabilities and exposures (CVE) and CVE candidates (CAN) databases [22], X-Force archives [23], bugtraq archives [24], and others [25], [26], [27]. We construct three matrices that offer different perspectives on firewall vulnerabilities. The first matrix, given in table I, cross references firewall operations (according to our firewall data flow model) with vulnerability causes. We simply insert each vulnerability tag/name in the matrix cell that corresponds to its classification by cause and its location in our data flow model. We will use this matrix to develop an intuition about which errors a firewall operation is most vulnerable to. For example, we find that the legality checks operations are susceptible to validation errors. On the other hand, we can also use the matrix to find the operations in which a certain error is most often found. For example, design errors are often found in the IP and port filtering operation.

The second matrix, given in table II, cross references each firewall operation with vulnerability effects. This matrix indicates which effects vulnerabilities in a certain operation tend to have. For example, we find that vulnerabilities found in the application level analysis operation tend to lead to the execution of code. We can also infer from this matrix the operation in which a certain type of effect is likely to appear. For example, vulnerabilities that lead to DoS attacks are typical in the legality checks operation.

The third matrix, given in table III, cross references vulnerability causes with vulnerability effects. This matrix can be used to find the effect that a certain kind of error leads to. For example, design errors typically lead to illegitimate access to target resources. On the other hand, we can also use the matrix to determine which errors often lead to a certain kind of effect. For example, DoS is typically the result of validation errors. The causes of a certain observed behavior can be extremely valuable during testing.

The three matrices provide a structured mechanism for expressing firewall vulnerabilities, and give an intuition about where and why they typically occur, and what effect they typically have. This intuition can then be used in several ways to improve the processes of firewall implementation and firewall testing (as discussed in the next section). We should note, however, that conclusions based on clusters in the matrices are based on our analysis of publicly disclosed vulnerabilities. While the matrices may show that an operation is most vulnerable to a certain kind of error, in actuality there could be another kind that is more prevalent but more difficult to find, which would explain why it does not appear in the public domain. Assuming that vulnerabilities found in the public domain are of average or below average difficulty in finding, then conclusions based on clusters in the matrices can only be reliably applied to vulnerabilities whose difficulty of finding are average and below average.

By examining the operation-versus-cause matrix (table I and the operation-versus-effect matrix (table II), it appears that the most vulnerable firewall operations are IP and port filtering, with six vulnerabilities, and legality checks, with five. These are followed by application level analysis, packet reassembly, and finally dynamic rule-set and NAT/PAT.

Vulnerabilities in IP and port filtering are due to validation errors (which are mostly due to invalid assumptions about the origin of an object) and design errors. The matrices also indicate that the vulnerabilities that appear at this level usually lead to the access to a resource (in this case, a machine on the internal subnet) or to a DoS attack. The legality checks operation checks for packets that are blatantly illegal and attacks that are commonly encountered. These checks are selected by firewall developers, so this operation does not have an underlying theme as the others do. All the vulnerabilities found in this operation are due to validation errors and lead to both access to target and DoS attacks. The most common vulnerability causes in application level operations are validation errors, followed by boundary checking and design errors. The consequences of these include access to target and execution of code. The packet reassembly layer includes fewer vulnerabilities than the previous two. The vulnerabilities at this level are due to validation and design errors. Packet reassembly vulnerabilities mostly lead to access to target resources.

Analyzing the cause-versus-effect matrix (table III), we see that most vulnerabilities are caused by validation errors and lead to an access to target. A smaller number of validation errors lead to DoS attacks and one leads to the execution of code. The next most prevalent vulnerability cause is design errors, which mostly lead to access to target. Finally, there is a boundary checking error that leads to execution of code.

TABLE I

OPERATION-VERSUS-CAUSE FIREWALL VULNERABILITY MATRIX: THIS MATRIX CROSS REFERENCES FIREWALL OPERATIONS (ACCORDING TO OUR FIREWALL DATA FLOW MODEL) WITH VULNERABILITY CAUSES IN THE VULNERABILITY TAXONOMY, AND DEMONSTRATES THE ANALYSIS OF 20 SAMPLE VULNERABILITIES, AS GIVEN IN THE APPENDIX.

| | NAT/PAT | Dynamic rule-set | Legality checks | IP/port filtering | Reassembly | Application level |
|---|---|---|---|---|---|---|
| Validation | | FreeBSD IPFW | Forwarding external packets; Forwarding reserved packets; Raptor DoS; Snapgear options DoS; WatchGuard SOHO options DoS | Gauntlet DoS; Firewall-1 RDP; Raptor zero-length UDP | IPFilter fragment bypass; Firewall-1 fast mode | Firewall-1 PASV; Firewall-1 script tags; WatchGuard SMTP |
| Authorization | | | | | | |
| Serialization/Aliasing | | | | | | |
| Boundary checking | | | | | | Norton firewall buffer overflow |
| Domain | | | | | | |
| Design | | | | ZoneAlarm port 67; Firewall-1 license DoS; Tiny personal firewall port 53 | PIX and CBAC fragmentation | PIX SMTP |

## V. RESOURCE ALLOCATION

When testing or developing software, it is important to make resource allocation decisions to minimize vulnerabilities in the system. Resources to be allocated may include time, money, expertise, and any other resource needed for system development. Traditionally, resource allocation has been performed in an ad-hoc manner, using both common sense and experience. In this section, we will outline a simple method in which the vulnerability matrices we developed in the previous section are used in allocating resources to different firewall operations.

The first step that must be performed is to map the firewall under implementation (or under test) to our data flow model. For example, a very basic packet filter typically only includes the IP and port filtering, and legality checks operations. The developer (or tester) can then use the operation-versus-cause matrix (table I) to determine which kinds of errors typically lead to vulnerabilities in this particular firewall operations. Using the operation-versus-effect matrix (table II), the developer can also determine the attacks and undesirable consequences that vulnerabilities in these operations are likely to lead to.

While this approach gives some intuition about where vulnerabilities occur and what effect they have, we can further refine the process. If a developer or tester can identify the types of attacks that they most wish to protect against, they can take the following steps to prioritize the kinds of errors to look for:

- Identify the most critical undesirable effects in table II, and obtain a list of the corresponding most important operations to check.
- Identify the same effects in table III to obtain a list of the causes of problems that usually lead to these attacks.

In order to better allocate their resources, developers and testers must look for the kinds of errors identified in the second step in the operations identified in the first step. In the following sections, we provide an in-depth description and case

TABLE II

OPERATION-VERSUS-EFFECT FIREWALL VULNERABILITY MATRIX: THIS MATRIX CROSS REFERENCES EACH FIREWALL OPERATION WITH VULNERABILITY EFFECTS IN THE VULNERABILITY TAXONOMY, AND DEMONSTRATES THE ANALYSIS OF 20 SAMPLE VULNERABILITIES, AS GIVEN IN THE APPENDIX.

| | NAT/PAT | Dynamic rule-set | Legality checks | IP/port filtering | Reassembly | Application level |
|---|---|---|---|---|---|---|
| Execution of code | | | | | | Firewall-1 script tags; Norton firewall buffer overflow |
| Change of target | | | | | | |
| Access to target | | FreeBSD IPFW | Forwarding external packets; Forwarding reserved packets | ZoneAlarm port 67; Firewall-1 RDP; Tiny personal firewall port 53 | Firewall-1 fast mode; IPFilter fragment bypass; PIX and CBAC fragmentation | Firewall-1 PASV; PIX SMTP; WatchGuard SMTP |
| DoS | | | Raptor DoS; Snapgear options DoS; WatchGuard SOHO options DoS | Firewall-1 license DoS; Gauntlet DoS; Raptor zero-length UDP | | |

studies of this approach.

## A. Operation Ranking

We rank firewall operations as a function of the kinds of attacks that we would most like to prevent. To do this, we use the effect matrix in table II to provide us with a means of quantifying the effect of vulnerabilities found in an operation. We start by attributing a weight to every effect. This weight depends on the particular situation. For example, in a certain context, DoS attacks may be more disastrous than a change of resource attack, so the weight of the first is assigned to be higher than the second. Next, we define the effect of an operation in the following manner. For each operation (column), we iterate over each effect (cell) and multiply the number of vulnerabilities associated with that effect (in the cell) by the weight of the effect. For example, from table II, assuming that all effects have equal weights (i.e., 1), we find that the application level analysis operation has an operation effect of 5. This allows us to rank the operations by their relative effects.

## B. Cause Ranking

We also rank vulnerability causes as a function of the types of attacks that we need to prevent the most. Instead of the operation-versus-effect matrix, we will use the cause-versus-effect matrix in table III. Again, we define a weight for each effect and compute the product of the number of vulnerabilities found in a cell, and the weight of the effect, summed over the entire column (cause). We can now rank the vulnerability causes as a function of their effects.

Once we have established a ranking of both firewall operations and vulnerability causes, we can combine the two in order to best allocate our efforts on specific errors in certain operations in order to prevent the most disruptive kinds of attacks. A developer or tester can then focus his/her resources on the highest ranked error in the highest ranked operation, then the second highest error in the highest operation, and so on.

TABLE III

CAUSE-VERSUS-EFFECT FIREWALL VULNERABILITY MATRIX: THIS MATRIX CROSS REFERENCES VULNERABILITY CAUSES WITH VULNERABILITY EFFECTS, AND DEMONSTRATES THE ANALYSIS OF 20 SAMPLE VULNERABILITIES, AS GIVEN IN THE APPENDIX.

| | Validation | Authen–tication | Serialization or Aliasing | Boundary checking | Domain | Design |
|---|---|---|---|---|---|---|
| Execution of code | Firewall-1 script tags | | | Norton firewall buffer overflow | | |
| Change of target | | | | | | |
| Access to target | Firewall-1 PASV; Forwarding external packets; Forwarding reserved packets; Firewall-1 RDP; Firewall-1 fast mode; WatchGuard SMTP; IPFilter fragment bypass; FreeBSD IPFW | | | | | PIX and CBAC fragmentation; PIX SMTP; ZoneAlarm port 67; Tiny personal firewall port 53 |
| DoS | Raptor DoS; Gauntlet DoS; Snapgear options DoS; WatchGuard SOHO options DoS; Raptor zero-length UDP | | | | | Firewall-1 license DoS |

### C. *Case Studies*

The proposed firewall testing methodology consists of two phases: the modeling phase and the ranking phase. The initial stage consists of mapping the firewall to the proposed firewall data flow model. This requires a basic functional understanding of the firewall to be tested, which can be acquired without much difficulty through technical manuals and/or testing. Once a simple functional model of the firewall has been conceived, we proceed to rank the operations and the vulnerability causes.

#### C.1 Firewall-1

This section provides a case study of how our methodology can be applied to CheckPoint's Firewall-1 [28]. (Our analysis was based on version 2.1, so the firewall operations may have been somewhat modified in more recent versions.) We begin by modeling Firewall-1, and then we use the matrices to guide us in allocating resources for testing it.

**Modeling.** Firewall-1 operations can be modeled as five main operations each corresponding to one of its primary functions. Figure 3 illustrates the Firewall-1 model. The first operation is packet reassembly. Firewall-1 reassembles every fragmented packet before it allows it to proceed further. Following that, comes the dynamic rule-set (state table) which is the core of stateful inspection. This is followed by IP and port filtering, which, as the name implies, is the packet filter (enforces the access lists). These last two operations work together to enable stateful inspection as follows. Upon receiving a SYN packet that is authorized by the filtering rules, the dynamic rule-set will allow the following packets belonging to that stream to bypass the IP and port filtering operation altogether. Next is header rewrite, which translates in-bound and out-bound network addresses, in order to hide the external and internal networks from each other. Finally, Firewall-1 can perform some basic application level analysis on packet data in order to filter out malicious content.

**Operation and Cause Ranking.** In this case study, we will assign the following weights to the vulnerability effects:

- Execution of code: 3
- Change of target: 2
- Access to target: 1

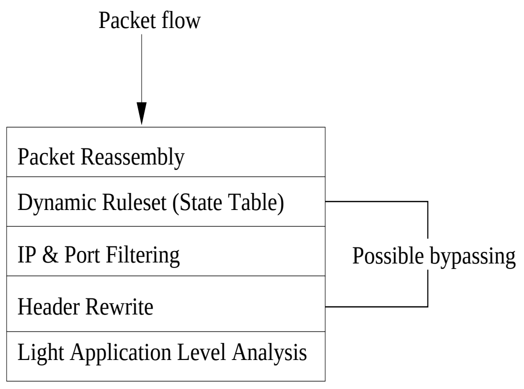

Fig. 3. Firewall-1 operations

• DoS: $4$

These weights are based on how disastrous the effect of each attack is, in a particular context. Given these weights, we use the operation-versus-effect matrix to compute a weighted sum for each operation. More specifically, for each column, we iterate over each cell and multiply the number of vulnerabilities associated with that effect (in the cell) by the previously assigned weight of the effect. The resulting weighted sums for each operation are:

• NAT/PAT: No vulnerabilities in this operation.
• Dynamic rule-set: 1 (vulnerability) $\times$ 1 (weight of “access to target”) yields a total of 1.
• Legality checks: 2 (vulnerabilities) $\times$ 1 (weight of “access to target”) $+$ 3 (vulnerabilities) $\times$ 4 (weight of “DoS”) yields a total of 14.
• IP and port filtering: 3 (vulnerabilities) $\times$ 1 (weight of “access to target”) $+$ 3 (vulnerabilities) $\times$ 4 (weight of “DoS”) yields a total of 15.
• Packet reassembly: 3 (vulnerabilities) $\times$ 1 (weight of “access to target”) yields a total of 3.
• Application level: 2 (vulnerabilities) $\times$ 3 (weight of “execution of code”) $+$ 3 (vulnerabilities) $\times$ 1 (weight of “access to target”) yields a total of 7.

Thus, we rank the operations in decreasing order of importance, as follows:
1. IP and port filtering
2. Legality checks
3. Application level
4. Packet reassembly
5. Dynamic rule-set
6. NAT/PAT

We also utilize the cause-versus-effect matrix to rank the vulnerability causes. We define the weighted sum for each cause as the product of the number of vulnerabilities found in a cell, and the weight of the effect, summed over the entire column. These computations are as follows:

• Validation: 1 (vulnerability) $\times$ 3 (weight of “execution of code”) $+$ 8 (vulnerabilities) $\times$ 1 (weight of “access to target”) $+$ 5 (vulnerabilities) $\times$ 4 (weight of “DoS”) yields a total of 31.
• Authorization: No vulnerabilities due to this cause.
• Serialization/aliasing: No vulnerabilities due to this cause.
• Boundary checking: 1 (vulnerability) $\times$ 3 (weight of “execution of code”) yields a total of 3.
• Domain: No vulnerabilities due to this cause.
• Design: 4 (vulnerabilities) $\times$ 1 (weight of “access to target”) $+$ 1 (vulnerability) $\times$ 4 weight of “DoS”) yields a total of 8.

We therefore rank the causes, in decreasing order of importance, as follows:
1. Validation
2. Design
3. Boundary checking
4. Authorization, serialization, aliasing, and domain

We can now combine both rankings and use them to establish an order in which to test for errors. The cause ranking prioritizes the vulnerability causes, while the operation ranking prioritizes the operations. We first look for validation, design, boundary, and authorization, serialization and domain errors in the IP and port filtering operation, then in the legality checks, followed by the application level analysis, packet reassembly, dynamic rule-set, and finally, the NAT/PAT operations.

### C.2 Raptor

Again, we divide our discussion into the modeling and ranking stages.

**Modeling.** The Raptor firewall [29] model is depicted in figure 4. (This model is for version 6 of Raptor, so the model may be somewhat different in more recent versions.) The first operation is an optional packet filter module that Raptor provides in case a network administrator needs to set up virtual private network (VPN) tunnels, or wants to add generic services to the firewall machine. This is followed by the packet reassembly operation which ensures that the following layers will only work with complete packets. Then, the legality checks test packets against known attacks and non-standard configurations (e.g., reserved bits, certain options). Following that is the header rewrite operation which hides network information and allows NAT/PAT. Finally, the most important operation, the application level analysis operation, rewrites every incoming and outgoing packet before forwarding it.

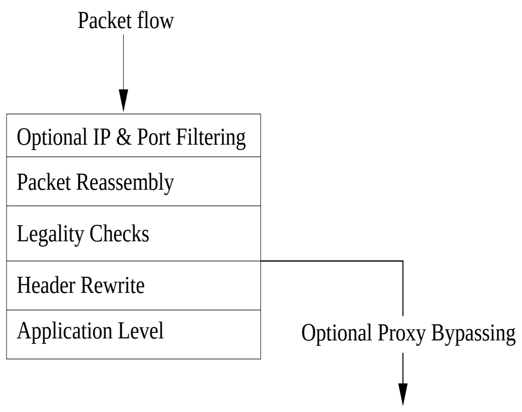

Fig. 4. Raptor firewall operations

**Operation and Cause Ranking.** For Raptor, we will use the following weights based on how severe the effects are, in a certain context:

- Execution of code: 3
- Change of target: 4
- Access to target: 4
- DoS: 1

We apply ranking on the operation-versus-effect matrix, by summing the weight-number of vulnerabilities products over every column (operation):

- NAT/PAT: No vulnerabilities in this operation.
- Dynamic rule-set: 1 (vulnerability) $\times$ 4 (weight of “access to target”) yields a total of 4.
- Legality checks: 2 (vulnerabilities) $\times$ 4 (weight of “access to target”) $+$ 3 (vulnerabilities) $\times$ 1 (weight of “DoS”) yields a total of 11.
- IP and port filtering: 3 (vulnerabilities) $\times$ 4 (weight of “access to target”) $+$ 3 (vulnerabilities) $\times$ 1 (weight of “DoS”) yields a total of 15.
- Packet reassembly: 3 (vulnerabilities) $\times$ 4 (weight of access to target) yields a total of 12.
- Application level: 2 (vulnerabilities) $\times$ 3 (weight of “execution of code”) $+$ 3 (vulnerabilities) $\times$ 4 (weight of “access to target”) yields a total of 18.

According to the computed numbers, we rank the operations in the following decreasing order of importance:

1. Application level
2. IP and port filtering
3. Packet reassembly
4. Legality checks
5. Dynamic rule-set

6. NAT/PAT

Applying ranking to the cause-versus-effect matrix, we sum the weight-number of vulnerabilities products over every column (cause), and obtain:

- Validation: 1 (vulnerability) $\times$ 3 (weight of "execution of code") + 8 (vulnerabilities) $\times$ 4 (weight of "access to target") + 5 (vulnerabilities) $\times$ 1 (weight of "DoS") yields a total of 40.
- Authorization: No vulnerabilities due to this cause.
- Serialization/aliasing: No vulnerabilities due to this cause.
- Boundary checking: 1 (vulnerability) $\times$ 3 (weight of "execution of code") yields a total of 3.
- Domain: No vulnerabilities due to this cause.
- Design: 4 (vulnerabilities) $\times$ 4 (weight of "access to target") + 1 (vulnerability) $\times$ 1 (weight of "DoS") yields a total of 17.

Therefore, we rank the causes in decreasing order of importance, as follows:

1. Validation
2. Design
3. Boundary checking
4. Authorization, serialization, aliasing and domain

## VI. Conclusions and Future Work

We have defined a method to describe firewall vulnerabilities and relate them to firewall operations in compact matrices. We have also reviewed twenty known firewall vulnerabilities and analyzed the constructed matrices. The matrices not only show vulnerability trends, but also aid in predicting the where and how of new unforeseen vulnerabilities. They are also useful– to both firewall developers and testers– in making resource allocation decisions.

Consider, for example, an organization that has purchased and deployed a new version of a firewall. Vulnerability postings on CVE and other sites will eventually appear, but the organization does not necessarily need to be entirely dependent on these postings (which invariably lag behind the discovery of each vulnerability and often also the development of exploitation tools) to maintain a consistent level of security on the firewall. Knowing that IP/port filtering, legality checks, packet reassembly, and application level vulnerabilities are most likely to occur, the organization could install an additional packet filter that weeds out the kinds of traffic most likely to exploit these vulnerabilities when the firewall is installed, thereby reducing the effect of the discovery of any new vulnerabilities related to these processing phases. This approach is proactive rather than reactive– something that most information security practices ardently strive to achieve.

Additionally, knowing the data flow processing operations in which vulnerabilities are most likely to surface can greatly facilitate an organization's estimation of threat and risk. Consider, for example, an organization that has many business critical servers and applications within its network. Suppose that this organization relies heavily on multiple firewalls, e.g., a firewall at the external gateway to its network and multiple internal firewalls, to screen traffic bound for subnets. This organization should realize that numerous IP/port filtering, legality checks, packet reassembly, and application level-related vulnerabilities are likely to surface over time, and should carefully analyze the effects upon its business interests (DoS, code execution, and so forth) if these vulnerabilities are successfully exploited. Assuming that each firewall completely or nearly completely will counter both externally- and internally-initiated threat is specious. The organization should instead elevate its estimate of residual risk resulting from partial effectiveness of deployed control measures– in this case, its firewalls.

We are currently extending this work in a number of ways. We are conducting (1) source code analysis; (2) attack tool analysis; and (3) extensive testing of a number of firewalls, to validate the clustering properties exhibited by our matrices. In addition, we observe that the original data flow model we developed in [1] is much more detailed than what we used in our matrices, and the vulnerability categories can be more detailed as well. In essence, the entire procedure can be detailed further, to yield more accurate results at the cost of more complex analysis. In addition, we can approximate the position of vulnerabilities in our matrices using functions. With these, we can estimate the probabilities of where vulnerabilities are likely to appear, in order to mathematically prove the optimal placement of sanity checks that prevent unforeseen vulnerabilities in firewalls. Finally, we are examining the complete automation of testing for use from the viewpoints of both programmer and tester.

## Acknowledgments

This work was supported by CERIAS. The authors would like to thank former members of the firewall group for their contribution to the discussions and the testbed setup. The authors would also like to thank the reviewers for their valuable comments that helped improve this paper.

## References

[1] M. Frantzen, F. Kerschbaum, E. Schultz, and S. Fahmy, “A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals,” *Computers and Security*, vol. 20, no. 3, pp. 263–270, May 2001.

[2] Ivan Krsul, *Software Vulnerability Analysis*, Ph.D. thesis, Department of Computer Sciences, Purdue University, 1998, https://www.cerias.purdue.edu/techreports-ssl/public/98-09.pdf.

[3] Wenliang Du and Aditya P. Mathur, “Testing for software vulnerability using environment perturbation,” in *Proceeding of the International Conference on Dependable Systems and Networks (DSN 2000), Workshop On Dependability Versus Malicious Faults*, June 2000, pp. 603–612, http://www.cerias.purdue.edu/homes/duw/ research/paper/ftcs30_workshop.ps.

[4] Wenliang Du and Aditya P. Mathur, “Categorization of software errors that led to security breaches,” in *Proceedings of the 21st National Information Systems Security Conference (NISSC’98)*, 1998, http://www.cerias.purdue.edu/homes/duw/research/paper/nissc98.ps.

[5] M. Bishop and D. Bailey, “A critical analysis of vulnerability taxonomies,” in *Proceedings of the NIST Invitational Workshop on Vulnerabilities*, July 1996, Also appears as Technical Report 96-11, Department of Computer Science, University of California at Davis (Sep. 1996) at http://seclab.cs.ucdavis.edu/projects/vulnerabilities/scriv/ucd-ecs-96-11.ps. Also see ”Classifying Vulnerabilities,” ”A Taxonomy of UNIX and Network Security Vulnerabilities” at http://seclab.cs.ucdavis.edu/projects/vulnerabilities/scriv/ucd-ecs-95-10.ps and ”Vulnerabilities Analysis” by the same author.

[6] E. Schultz, “How to perform effective firewall testing,” *Computer Security Journal*, vol. 12, no. 1, pp. 47–54, 1996.

[7] E. Schultz, “When firewalls fail: lessons learned from firewall testing,” *Network Security*, pp. 8–11, Feb 1997.

[8] William R. Cheswick and Steven M. Bellovin, *Firewalls and Internet Security: repelling the wily hacker*, Addison-Wesley, Reading, Massachusetts, 1994.

[9] Bill Cheswick, “The design of a secure Internet gateway,” in *USENIX*, Anaheim, California, June 1990, pp. 233–237.

[10] Kathryn M. Walker and Linda Croswhite Cavanaugh, *Computer Security Policies and SunScreen Firewalls*, Prentice Hall, Upper Saddle River, New Jersey, 1998.

[11] Linda McCarthy, *Intranet Security*, Prentice Hall, Upper Saddle River, New Jersey, 1998.

[12] Rita C. Summers, *Secure Computing: Threats and Safeguards*, McGraw-Hill, 1997.

[13] Gene Spafford and Simson Garfinkel, *Practical Unix and Internet Security*, O’Reilly & Associates, Inc, second edition, 1996.

[14] Christoph L. Schuba, *On the Modeling, Design and Implementation of Firewall Technology*, Ph.D. thesis, Department of Computer Sciences, Purdue University, December 1997, https://www.cerias.purdue.edu/techreports-ssl/public/97-07.pdf.

[15] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith, “Implementing a distributed firewall,” in *Proceedings of the ACM CCS*, November 2000, Also see http://www.research.att.com/∼smb/papers/distfw.html and www.DistributedFirewalls.com.

[16] Reto E. Haeni, “Firewall penetration testing,” http://www.seas.gwu.edu/∼reto/papers/firewall.pdf, 1997.

[17] Giovanni Vigna, “A formal model for firewall testing,” http://www2.elet.polimi.it/pub/data/ Giovanni.Vigna/www_docs/pub/fwtest.ps.gz.

[18] M. R. Lyu and L. K. Y. Lau, “Firewall security: policies, testing and performance evaluation,” in *Proceedings of the COMSAC*. IEEE Computer Society, 2000, pp. 116–21.

[19] D. Newman, “Benchmarking terminology for firewall performance,” Request for Comments 2647, ftp://ftp.isi.edu/in-notes/rfc2647.txt, Aug. 1999.

[20] N. Freed, “Behavior of and requirements for internet firewalls,” Request for Comments 2979, http://search.ietf.org/rfc/rfc2979.txt, Oct. 2000.

[21] Taimur Aslam, “A taxonomy of security faults in the UNIX operating system,” M.S. thesis, Purdue University, 1995, purdue.edu/pub/COAST/papers/taimur-aslam/aslam-taxonomy-msthesis.ps.Z. Also see: Use of a Taxonomy of Security Faults by Taimur Aslam, Ivan Krsul and Gene Spafford.

[22] “Vulnerability databases from the Common Vulnerabilities and Exposures (CVE) and Candidates (CAN),” http://cve.mitre.org/cve/, 2003.

[23] “X-Force,” http://xforce.iss.net/, 2003.

[24] “Bugtraq,” http://www.securityfocus.com/bugtraq/archive, 2003.

[25] “Computer Emergency Response Team (CERT) advisories,” http://www.cert.org/advisories/, 2003.

[26] “System Administration, Networking, and Security (SANS),” http://www.sans.org/, 2003.

[27] “Firewalls digest,” http://lists.gnac.net/firewalls/, 2003.

[28] “Check Point Products,” http://www.checkpoint.com, 2003.

[29] “Symantec products,” http://www.symantec.com, 2003.

## Appendix

### Vulnerability Analysis

In this appendix, we analyze reported firewall vulnerabilities from vulnerability databases and reports. Our sources include the MITRE common vulnerabilities and exposures (CVE) and CVE candidates (CAN) databases [22], X-Force archives [23], bugtraq archives [24], and others [25], [27], [26]. We summarize the reported problem description in our own words, then classify the problem according to the vulnerability cause and effect taxonomy, and according to our firewall operation data flow model:

1. **Firewall-1 PASV (CVE-2000-0150 and BID 979):** The FTP protocol can be executed in passive mode. In this mode, a client includes a PASV command when setting up its data connection. The server, in turn, responds with a "227" reply that indicates an IP address and an open port on which it will wait for a connection. On seeing an outgoing packet with the string "227" in the first four bytes, Firewall-1 extracts the IP address and port given by the server, checks that the IP address matches the packet source IP, and then proceeds to allow a connection from the client to the server. However, since there are ways (notably using the error handler) to fool the server into sending a packet with the "227" string in the first four bytes of its payload, an attacker can use the passive behavior of the FTP protocol to bypass the firewall rules.
**Cause:** Input validation error. The firewall simply checks that the string "227" occurs in the beginning of a packet payload. However, this check is insufficient, as it alone cannot determine the legitimacy of a "227" response packet. The firewall thus fails to properly validate its input, which, in this case, is the response packet.
**Effect:** Access to target resource. By allowing an illegitimate connection from the client to the FTP server, the firewall is granting an attacker access to a resource.
**Operation:** Application level analysis. The firewall performs its check on data provided by the FTP server, which is at the application layer (it only requires access to the application level).

2. **Firewall-1 Script Tags (CVE-2000-0116 and BID 954):** Firewall-1 has the ability to check HTML data for script tags and render scripts unexecutable by browsers. However, this feature can be bypassed by simply adding an extra open angled bracket "<" to the script tag. The firewall does not recognize the tag as a script tag, and forwards the HTML code as is. Browsers, on the other hand, still recognize the tag and execute the script.
**Cause:** Input validation error. Again, the firewall fails to properly check for script tags. An extra open angled bracket should not invalidate a script tag. Thus, the firewall fails to properly validate its input, which, in this case, is the HTML code.
**Effect:** Execution of code. By fooling the firewall into forwarding the HTML code, an attacker can cause illegitimate (in terms of the firewall security policy) code to be executed by a browser.
**Operation:** Application level analysis. The firewall parses data that is provided by an HTTP server, which is at the application layer (it only requires access to the application level).

3. **Forwarding External Packets Claiming to be Internal (CAN-1999-0528):** Some firewalls forward external packets that claim to come from the internal network.
**Cause:** Origin validation error. The firewall fails to check on which interface the packet came from. Since source addresses can be spoofed, the only trusted source of information for a packet origin is the incoming interface. Failure to (at least) check the interface on which a packet came is a failure to check its origin.
**Effect:** Access to target resource. By failing to check the interface, the firewall allows an attacker to send packets to hosts inside the protected network. (This may, in turn, result in other effects.)
**Operation:** Legality checks. Though logically we might place this in the IP and port filtering operation, it is traditionally done in the legality checks operation.

4. **Forwarding Packets From Reserved Addresses (CAN-1999-0529):** Some firewalls forward packets claiming to originate from IANA-reserved or private addresses (e.g., 10.x.x.x, 127.x.x.x or 217.x.x.x).
**Cause:** Origin validation error. The firewall fails to check whether the packet is coming from a reserved address or not.
**Effect:** Access to target resource. (See previous vulnerability.)
**Operation:** Legality checks. This is traditionally handled in the legality checks operation.

5. **Cisco PIX and CBAC Fragmentation Attack (CVE-1999-0157 and Cisco bug ID CSCdk36273):** When a packet is fragmented, the enclosed TCP header is usually only found in the first packet fragment. This means that an IP filtering firewall can only apply rules that pertain to the TCP header on the first fragment. Due to this, the Cisco PIX firewall and the IOS firewall feature set only apply filtering rules to the first fragment of a packet. The assumption here is that

if the first fragment of a packet is blocked, then even if a destination host receives all the other fragments, it will be unable to re-construct the original packet. While it is true that the destination host will not be able to re-construct the original packet, it can construct a packet that consists of all the remaining fragments it received, and this new packet can be malicious.
**Cause:** Design error. At first glance, this vulnerability could be seen as an origin validation error in the sense that the firewall fails to check the origin of the non-initial fragments. However, due to how IP fragments packets, this is impossible to perform. The only solution is to first reassemble the entire packet, and then check the origin and apply the rules. Therefore, the most immediate error is not the fragment origin validation, but the reassembly (or lack of) of the original packet. The fact that the firewall does not reassemble fragments (before applying the rules) also implies a design error.
**Effect:** Access to target resource. The firewall allows an attacker to send a malicious and fragmented packet to a protected host.
**Operation:** Reassembly.

6. **Raptor Denial of Service (CVE-1999-0905 and BID 736):** The IP header contains an options field. When parsing this field, the Raptor firewall iterates over the options as follows:

```
        int option_length, i;
        for(i = 20; i < header_length; i += option_length){
                /* execute parsing code */
                /* update option_length */
        }
```

During each iteration, $i$ points to the beginning of the next option. However, if an option length is $0$, the loop becomes infinite.
**Cause:** Input validation error. The firewall options parser fails to check if the current option is of length 0. Therefore, the firewall fails to properly check its input, which in this case is the packet IP options.
**Effect:** Denial of service. The infinite loop that results from zero-length options freezes the firewall, making it unavailable for processing incoming packets.
**Operation:** Legality checks. The parsing of IP options is traditionally found in the legality checks operation.

7. **Cisco PIX Firewall SMTP Content Filtering Evasion Vulnerability (CVE-2000-1022 and BID 1698):** Certain SMTP commands such as HELP, EXPN and VRFY are known to be exploitable. Due to this, the Cisco PIX firewall may filter out arbitrary SMTP commands. However, in order to avoid false positives, the firewall does not apply its SMTP filters to any text sent between a “DATA” and a “<CR><LF><CR><LF>.<CR><LF>” command, which delineate the beginning and the end of the email text. Thus, if an attacker sends a “DATA” command before sending any other commands, the SMTP server responds with an error message, but the firewall stops filtering content until it receives an “END” command. At this point, the attacker can send any command to the SMTP server and the firewall lets it through.
**Cause:** Design error. The Cisco PIX SMTP filter is stateless, so it cannot filter content according to its context. This is why it cannot keep track of whether a “DATA” command has been sent before the other commands or not.
**Effect:** Access to target resource. By fooling the firewall into a state where it looks at all incoming data as email text, an attacker illegitimately accesses a protected SMTP server.
**Operation:** Application level analysis. The firewall SMTP filter operates on application level data.

8. **ZoneAlarm Personal Firewall Port 67 Vulnerability (CVE-2000-0339 and BID 1137):** When an attacker wishes to know what services are running on a server, he/she can initiate a port scan on that server. This can be done in many different ways, but the simplest is to send a TCP SYN packet with a certain port number. If the attacker receives an ACK packet in response, then he/she knows that the port is open. If he/she receives an ICMP port unreachable packet, he/she knows that it is closed. By incrementing the port number, an attacker can scan all the ports on a server, and find out which ones are open or closed. Most firewalls can detect port scans and will generally log or block them, as they are usually a precursor to an attack. While the Zone Labs firewall can detect and log port scans, it fails to do so if they

originate from port 67.
**Cause:** Design error. Since port 67 is the DHCP/BOOTP port, a check was omitted in order to allow easier bootstrapping.
**Effect:** Access to target resource. This allows an attacker to illegitimately access the host.
**Operation:** IP and port filtering.

9. **Gauntlet Firewall Denial of Service Attack (CVE-1999-0683 and BID 556):**
All ICMP error packets encapsulate the first 64 bytes of the packet that caused the error. The Gauntlet firewall hangs when it has to forward such an ICMP error packet, when the encapsulated packet IP_ID field is random and certain IP options are set.
**Cause:** Input validation error. The firewall fails to check the form of its input, in this case the encapsulated packet.
**Effect:** Denial of service. If an attacker sends an ICMP error packet with an encapsulated packet with certain IP options and a random IP_ID, the firewall hangs.
**Operation:** IP and port filtering.

10. **CheckPoint Firewall-1 RDP Header Firewall Bypassing Vulnerability (BID 2952):** Firewall-1 implements a proprietary protocol called RDP (different than the Reliable Datagram Protocol defined in RFC 908), which is defined on top of UDP at port 259. Firewall-1 automatically forwards any RDP packet without checking the packet.
**Cause:** Origin validation error. The firewall fails to check the source of the packet as long as it is an RDP packet.
**Effect:** Access to target resource. The firewall allows an attacker to reach a protected host.
**Operation:** IP and port filtering.

11. **CheckPoint Firewall-1 4.1 License Denial of Service Vulnerability (BID 2238):** If Firewall-1 is only licensed to protect a limited address space, it keeps a running count of the number of different destination addresses it sees in incoming packets. Once it has reached the maximum number of allowed addresses, it outputs an error message to the console for every incoming packet with a new destination address (whether the address is behind the firewall or not). Each error message that is sent to the console generates a load on the system. If an attacker can generate a lot of traffic to internal hosts that are not covered by the license, he/she can make the console unusable.
**Cause:** Design error. The firewall should not be sending a message to the console for every packet outside of the license range.
**Effect:** Denial of service. The load of processing messages that are sent to the console makes the system unusable to a legitimate user or administrator.
**Operation:** IP and port filtering. The enforcement of the license is done at the IP filtering level, where the license code can have access to the destination IP.

12. **CheckPoint Firewall-1 Fast Mode TCP Fragment Vulnerability (BID 2143):** CheckPoint's Firewall-1 contains a vulnerability in its fragmentation engine that allows an attacker to bypass the firewall rules. The vulnerability stems from the fact that the firewall does not check and discard initial fragments that do not contain complete TCP headers.
**Cause:** Input validation error. Firewall-1 fails to check that the initial fragment does not contain a complete TCP header.
**Effect:** Access to target resource. The firewall allows an attacker to access a protected host.
**Operation:** Reassembly.

13. **WatchGuard Firebox SMTP Proxy Attachment Bypassing Vulnerability (BID 2855):** WatchGuard's firewall has the ability to check email content and attachments. However, if an attacker appends to the boundary name two dashes, the attachment is not be checked.
**Cause:** Input validation error. Under a certain condition, the firewall fails to check its input.
**Effect:** Access to target resource. An attacker is able to send, by way of attachment, illegitimate data or even executable code to a host. (This vulnerability may thus, in some cases, cause illegitimate code execution.)
**Operation:** Application level analysis. SMTP filtering takes place at the application level.

14. **IPFilter Fragment Rule Bypass Vulnerability (BID 2545):** When dealing with fragments, the IPFilter firewall

has the ability to cache the forwarding decision that was made on the initial fragment, and later apply it to the other fragments. This allows the firewall to safeguard against many fragmentation attacks since it can filter out an entire packet based on the header included in the initial fragment. Unfortunately, the way in which it associates non-initial fragments with their initial fragments (and hence the cached decision), is by the IP fragment identifier (ID). This means that an attacker can send a valid initial fragment, which causes the firewall to cache a "forward" decision for all fragments with the corresponding IP ID. The attacker can then send any fragment (even a new initial fragment) with the corresponding IP ID, and the firewall simply forwards it.
**Cause:** Input validation error. When an initial fragment has already been received and a decision has already been cached, the firewall fails to check if any of the incoming fragments with the corresponding IP ID are initial fragments (in which case it should check the new fragment headers against its rules and replace the cached decision with the new one).
**Effect:** Access to target resource. The firewall allows an attacker to send fragments of an illegitimate packet to a protected host, where they can possibly be reconstructed and processed.
**Operation:** Reassembly.

15. **FreeBSD IPFW Filtering Evasion Vulnerability (BID 2293):** IPFW can apply rules to packets that are part of an already established TCP connection. The problem is that it treats all packets that have the ECE bit (one of the reserved TCP flags) set, as part of such a stream. An attacker that sets the ECE bit on a packet is guaranteed that it will be processed as if it were part of an already established connection, even when it is not.
**Cause:** Input validation error. The firewall fails to correctly associate incoming packets with TCP connections.
**Effect:** Access to target resource. The firewall allows an attacker to possibly send illegitimate packets to a protected host.
**Operation:** Dynamic rule-set filtering. The association of packets with already established connections happens in the dynamic rule-set filtering operation.

16. **Symantec Norton Firewall Buffer Overflow (BID 5237):** The HTTP proxy of the Norton firewall cannot handle HTTP requests over a certain length. Such requests cause a buffer overflow in kernel memory and can enable an attacker to execute code.
**Cause:** Boundary checking error. The firewall fails to check the size of the request.
**Effect:** Execution of code. It is possible for the attacker to execute code illegitimately.
**Operation:** Application level analysis. The vulnerability is in the HTTP proxy, which is part of the application level analysis.

17. **Tiny Personal Firewall port 53 Vulnerability (BID 4725):** To facilitate DNS, the Tiny personal firewall automatically forwards all packets from and destined to port 53 (the DNS port).
**Cause:** Design error.
**Effect:** Access target resource. The firewall allows an attacker to access a host on port 53.
**Operation:** IP and port filtering. The application of rules based on port numbers is done in the IP and port filtering operation.

18. **Snapgear Firewall IP Options DoS (BID 4660):** The Snapgear Lite+ firewall crashes when it tries to parse many IP packets with malformed IP options.
**Cause:** Input validation error. The firewall fails to properly check the IP header and its options.
**Effect:** Denial of service. The firewall crashes.
**Operation:** Legality checks. IP Options parsing is typically performed in the legality check operation.

19. **WatchGuard SOHO Firewall IP Options DoS (BID 4447):** The WatchGuard SOHO firewall crashes when trying to forward a packet with illegitimate IP options.
**Cause:** Input validation error. The firewall fails to properly check the IP header and its options.
**Effect:** Denial of service. The firewall crashes.
**Operation:** Legality checks. IP Options parsing is traditionally done in the legality check operation.

20. **Raptor Firewall Zero Length UDP DoS (BID 3509):** The Raptor firewall crashes if it receives a UDP packet with a length field of 0.
**Cause:** Input validation. The firewall fails to check the length of UDP packets.
**Effect:** Denial of service. The firewall crashes.
**Operation:** IP and port filtering. The processing of UDP headers is done in the IP and port filtering operation.

## AUTHOR BIOGRAPHIES

**Seny Kamara** received his B.Sc. degree in computer science from Purdue University in 2001, and is currently a Ph.D. student in computer science at Johns Hopkins University in Baltimore, MD. His current research interests include cryptography (foundations, applied and quantum) and network security.

**Sonia Fahmy** is an assistant professor in the department of Computer Sciences at Purdue University, West Lafayette, IN. She completed her PhD degree at the Ohio State University in August 1999. Her research interests are in the design and evaluation of network architectures and protocols. She is currently investigating network-aware applications using Internet tomography; and heterogeneous sensor networks integrated with active control. She has been very active in the Traffic Management working group of the ATM Forum, and has participated in several IRTF and IETF working groups. Her work is published in over 40 papers, including publications in leading journals such as *IEEE/ACM Transactions on Networking* and *Computer Networks*, and in leading conferences and workshops such as IEEE INFOCOM and ICNP, and ACM NOSSDAV. She is a member of the ACM, IEEE, Phi Kappa Phi, Sigma Xi, and Upsilon Pi Epsilon. She has served on the technical program committees of IEEE INFOCOM, ICNP, ICC, ICDCS, GLOBECOM; chaired the tutorials for IEEE Hot Interconnects; and co-chaired the first SPIE conference on scalability and traffic control in IP networks. Please see http://www.cs.purdue.edu/homes/fahmy/ for more information.

**Eugene Schultz**, Ph.D., CISSP is a Principal Engineer with Lawrence Berkeley National Laboratory of the University of California and also teaches computer science courses there. He is the author/co-author of four books, and has written over 90 papers. Gene is the Editor-in-Chief of “Computers and Security,” and is the former Editor-in-Chief of “Information Security Bulletin.” He has received the NASA Technical Excellence Award and the Information Systems Security Association (ISSA) Professional Achievement and Honor Roll Awards, and was recently selected to the ISSA Hall of Fame. While at Lawrence Livermore National Laboratory, he was the founder and original project manager of the U.S. Department of Energy’s Computer Incident Advisory Capability (CIAC) and also a co-founder of FIRST, the Forum of Incident Response and Security Teams. He has provided expert testimony before committees within the U.S. Senate and House of Representatives on various security-related issues, and has served as an expert witness in legal cases.

**Florian Kerschbaum** is a PhD student at Purdue University, West Lafayette, IN. He received his MS from Purdue University in 2000. His current research interests include software watermarking and network security.

**Michael Frantzen** graduated from Purdue in 2001 with a Bachelors degree in Computer Engineering. He has authored several firewall testing suites. As an undergraduate student, he wrote the Mongoose proxying firewall and worked as a developer on the Sunscreen firewall. Now he is employed by NFR Security as a developer on their intrusion detection system. He is also a developer of the OpenBSD PF firewall.