

# A Stateful Inspection of FireWall-1

Thomas Lopatic, John McDonald  
*TUV data protect GmbH*  
{tljm}@dataprotect.com

Dug Song  
*Center for Information Technology Integration*  
*University of Michigan*  
dugsong@monkey.org

August 15, 2000

## 1 Introduction

At the Black Hat Briefings 2000, we presented an analysis of Check Point FireWall-1 vulnerabilities resulting from protocol design flaws, problems in stateful inspection, common or default misconfigurations, and minor implementation errors discovered over the past few months in the lab, and verified in real-world penetration tests.

In this advisory, we summarize our findings and provide some recommendations for safe configuration. It is intended to complement our presentation slides and source code, available at:

<http://www.dataprotect.com/bh2000/>

Our research was conducted against a Nokia IP440 running FireWall-1 version 4.0 Service Pack 5. Check Point later provided us with version 4.1 Service Pack 1, which was still vulnerable to most of our attacks, but came with a much safer default configuration. Check Point has since released service packs for all supported versions of FireWall-1 to specifically address the vulnerabilities documented here.

## 2 Authentication

In the examination of any security product, the administrative control channels are often the weakest link – and the most security critical. Indeed, we found that FireWall-1, in a typical distributed deployment, was susceptible to several trivial attacks against its inter-module authentication protocols.

Our attacks assume that the attacker has access to port 256/tcp on the filter module, and knows the

IP address of either a management module or any peer for which a shared authentication secret has been configured using “fw putkey.” In version 4.0, this port is exposed to the world by default, as it is also used by SecuRemote clients. In version 4.1, however, SecuRemote clients use port 264/tcp, with access to 256/tcp restricted.

While only the management console is able to issue unload commands, the binary load (bload) command is enabled by default for any IP address that the filter module knows an authentication secret for. We were able to verify this under version 4.0 by using binary load to upload an “allow all” policy. Our attacks bypass authentication to issue an unload command, which is easier to implement, and more portable between versions.

### 2.1 IP Address Verification

In the inter-module authentication protocol, a filter module does not verify the source IP address of an authenticating management module by checking the peer address of a connection – it examines a list of IP addresses handed to it. A common misconfiguration is to put

```
127.0.0.1: */none
```

in the control.map file, to ease local administration. An attacker only needs to send 127.0.0.1 as her IP address to bypass authentication completely, as demonstrated by our “fw1none” program.

After exchanging version and IP address information, the filter module replies with the authentication it wants to be performed. Upon successful authentication, the management module transmits the

arguments for the command to be executed. Finally the filter module returns a status code to indicate whether or not command execution was successful.

The protocol is in general not strictly synchronous. In particular, the exchange of IP addresses happens asynchronously, i.e. the management module does not have to send its IP addresses first. It could just wait for the filter module to send all IP addresses and then transmit its own. In this way, an attacker acting as a management module may learn all IP addresses of the firewall without actually authenticating. This is useful for the encapsulation attacks discussed later, or in discovery of a management module's IP address.

If we present an IP address to the filter module that does not correspond to a management module – or more generally speaking, an IP address that it does not have an authentication secret for – it will simply refuse to authenticate. Hence we can scan for the correct IP address of the management module by connecting repeatedly, giving a different IP address (or list of IP addresses) each time.

## 2.2 S/Key

S/Key authentication [Ha95] is used when communicating with embedded FireWall-1 devices. It is also the fallback for version 4.0 filter modules that do not have an encryption license and thus do not support FWA1.

The problem with Check Point's S/Key implementation is that after a shared secret has been used for 99 iterations, the management module generates a new authentication secret using `time()`. The `time()` function has a resolution of seconds. Therefore, there are only  $24 * 3600 = 86400$  different possibilities for secrets generated on a single day. Even worse, if the status of the filter module is polled periodically every 10 seconds, 99 authentications will be performed in a maximum 990 seconds. Working back from current time, we just have to try each of the 990 possible values in the period from now to 990 seconds ago, because we can be sure that a new shared secret has been generated at least once during this period.

With our S/Key brute forcing program “fw1bf,” we are able to perform about 50 guesses per second against FireWall-1 4.0 by using parallel connections, covering a whole day of possible authentication secrets in less than half an hour. Version 4.1 does not support as many parallel connections, reducing our effective rate by an order of magnitude.

The S/Key authentication secret, once recovered, may be used with our “fw1skey” tool to issue an

unload command to the filter module.

There is no encryption with S/Key authentication. Neither are there any provisions for data integrity. Internal threats like TCP connection hijacking apply [Be89].

## 2.3 FWN1

FWN1 is an alternative to S/Key, sometimes used at sites lacking an encryption license for version 4.0 filter modules. It is the default authentication method for OPSEC in version 4.0 and version 4.1.

Authentication is based on a shared secret  $K$ . This secret is needed to sign data using a keyed hash.  $K$  is appended to the data and the result is fed to a cryptographically secure hash function. The resulting digest is used as a signature. The peer, who also knows  $K$ , is able to verify the signature by calculating the same keyed hash.

The original value for  $K$  is set using “fwputkey.” However, after the first authentication, a Diffie-Hellman key exchange is initiated, resulting in a new shared 1024 bit secret  $K$  for authentication.

The FWN1 protocol works as follows:

1. The filter module generates a random number  $R1$ .
2. The filter module signs  $R1$ . The signature  $S1 = \text{Hash}(R1 + K)$ , where '+' denotes concatenation.
3.  $R1$  and  $S1$  are sent to the management module.
4. The management module verifies  $S1$ .
5. The management module generates a random number  $R2$ .
6. The management module calculates the corresponding signature  $S2 = \text{Hash}(R2 + K)$ , where '+' is concatenation again.
7.  $R2$  and  $S2$  are sent to the filter module.
8. The filter module verifies  $S2$ .

This protocol is trivially defeated by a simple replay attack. Instead of generating our own  $R2$ , we can just reuse the value  $R1$  sent by the filter module. Then, we can also reuse the signature  $S1$ , instead of having to generate our own  $S2$ .

Our “fw1fwn” utility implements the unload command with FWN1 authentication.

There is no encryption with FWN1 authentication. Neither are there any provisions for data integrity. Internal threats like TCP connection hijacking apply [Be89].

## 2.4 FWA1

FWA1 is similar to FWN1, with the addition of encryption to the communication between the modules. This is the default mechanism used in version 4.1, and for most commands in version 4.0 when an encryption license is present.

Again, there is a shared secret  $K$ , initially set using “fw putkey.” After the first successful authentication, it is replaced by a 1024 bit value resulting from a Diffie-Hellman key exchange.

The FWA1 protocol works as follows:

1. The filter module generates a random number  $R1$ .
2. The filter module signs  $R1$ . The signature  $S1 = \text{Hash}(R1 + K)$ , where ‘+’ denotes concatenation.
3.  $R1$  and  $S1$  are sent to the management module.
4. The management module verifies  $S1$ .
5. The management module generates a random number  $R2$ .
6. The management module calculates  $R3 = R1 \hat{\ } R2$ , where ‘^’ denotes XOR.
7. The management module calculates the corresponding signature  $S2 = \text{Hash}(R3 + K)$ , where ‘+’ is concatenation again.
8.  $R2$  and  $S2$  are sent to the filter module.
9. The filter module verifies  $S2$ .

Hence, the only change from FWN1 authentication is that instead of signing  $R2$ , the management module signs  $R3 = R1 \hat{\ } R2$ . Again, this protocol is trivially defeated with a simple replay attack. We just choose  $R2$  to be zero. In this way  $R3 = R1 \hat{\ } R2$  will yield  $R3 = R1$  and we can again reuse the filter module’s signature.

The “fw1fwa” utility implements FWA1 authentication. However, the unload command cannot be issued, as encryption is used to protect the communication channel.

We have simplified our presentation of the FWN1 and FWA1 authentication algorithms a bit for the sake of clarity. In practice, only 64 of the 128 bits of the signature are transmitted. In FWA1 the remaining 64 bits of  $S1$  and  $S2$  are concatenated to form the 128 bit encryption key. Since  $S1 = S2$  in our attack, we would have to guess the 64 bit

encryption key to successfully issue an unload command. There may still be a more sophisticated and practical way which has yet to be investigated.

If the shared 1024 bit secret generated by the Diffie-Hellman key exchange turned out to be guessable by a practical brute force attack because of a lack of entropy, we would perhaps be able to find a relatively small set of candidate values for  $K$  by brute forcing based on  $R1$  and  $S1$ . Then we could try all candidate values for  $K$  to find the real  $K$ . To identify the real  $K$ , we could use the known plaintext of the “authentication successful” reply that we receive, which is already encrypted.

To date, we have not come up with a practical attack on the complete FWA1 protocol, including encryption. Still, the authentication mechanism is seriously broken.

FWA1 uses the proprietary stream cipher FWZ1, recently posted anonymously to the Usenet sci.crypt newsgroup. Unfortunately, even a simple command like unload needs at least 18 bytes of arguments. Moreover, a different key is used in either direction, confounding any known plaintext attack against the “authentication successful” reply.

There are no provisions for data integrity in FWA1. Internal attackers are able to flip bits in a connection between two modules by simply XORing the ciphertext  $C$  with a corresponding bit mask  $X$ , since the decrypted plaintext  $P = C \hat{\ } K$ , where  $K$  is the key stream and ‘^’ denotes XOR. Decryption of  $C \hat{\ } X$  would thus yield  $(C \hat{\ } X) \hat{\ } K = (C \hat{\ } K) \hat{\ } X = P \hat{\ } X$ .

## 3 Packet Filtering

Static packet filtering is perhaps the simplest mechanism for network access control, but also deceptively hard to implement correctly [Ch92]. We found FireWall-1 to be susceptible to a number of attacks based on incomplete input verification, including some that allowed for attacks against stateful inspection as well.

### 3.1 TCP Fastmode

FireWall-1 provides a mechanism known as “fastmode” to allow an administrator to designate certain services as being performance critical. The FireWall-1 kernel module simply passes packets that have a source or destination port of a fastmode service without any additional connection or rule base checking.

Additionally, in fastmode, only SYN packets are verified. This allows an attacker to pass non-SYN TCP packets through the firewall to map the network by setting the source port of her packets to that of the fastmode service. We demonstrated mapping a network using the `-g` and `-sF` options in `nmap` [Fy97].

### 3.2 FWZ Encapsulation

FireWall-1's simple tunneling protocol for SecuRemote connections proves extremely useful in bootstrapping insertion attacks against stateful inspection. This protocol (FWZ tunnelling via IP protocol 94) is decapsulated at the very beginning of pre-inspection, before any real analysis is performed. The decapsulated packet is then fed through the inspection process. It is not necessary to authenticate or encrypt packets in order to use this encapsulation protocol. Furthermore, we could find no way to disable this functionality, short of filtering IP protocol 94 with another device.

An ordinary IP packet is FWZ encapsulated as follows: the original destination IP address and protocol are appended as a trailer to the end of the packet, and replaced in the original header by the target IP address (one of the firewall's interfaces) and IP protocol 94. The trailer, five bytes in length, is obfuscated via XOR with a keyed hash on the IP ID. While we didn't recover the key used in the hash, our tools cheat by fixing the IP ID to one and XORING against a static hash.

In this manner, FWZ encapsulation may be used to send packets through the firewall that would otherwise be unroutable, such as packets from the DMZ or intranet to the Internet, packets destined for a multicast address, or packets to addresses behind NAT or on RFC 1918 private networks.

### 3.3 IP Spoofing Protection

IP spoofing protection on FireWall-1 is configured per network object at the interface level. Several options are possible, but the typical configuration looks something like this:

1. DMZ and intranet interfaces set to "This Net" or perhaps "This Net +", restricting valid source IP addresses on the interface to those directly on its network, or routable to its network.
2. External interface set to "Others", disallowing packets purporting to originate from any of the DMZ or intranet networks.

While this configuration seems simple enough, it leaves out spoofing protection for one important IP address – the external interface of the firewall. Denial of spoofed packets coming from firewall interfaces is apparently enabled by default in all versions of FireWall-1 other than version 4.1 Service Pack 1. Coupled with the default rule to allow ISAKMP packets, this hole allows an attacker to send any UDP datagram to the external firewall interface.

Another possibility for evading IP spoofing protection is to use the all-hosts multicast address (224.0.0.1) as a mechanism for delivering packets to the underlying operating system of the firewall. For our demonstration, we used FWZ encapsulation to spoof a packet from the multicast address to our attack host, allowing us to respond with a packet sent to the multicast address, passed on to the firewall itself. This attack can also be performed with broadcast addresses.

## 4 Stateful Inspection

Stateful inspection firewalls try to enforce certain semantics in the traffic they relay. The mechanisms used to accomplish this, however, are often incomplete or subject to simple insertion or evasion, bearing some similarity to the problems encountered in passive network monitoring [PN98].

FireWall-1's stateful inspection is vulnerable in a number of ways related to layering violations in inspection, and ambiguity in end-to-end semantics. Many of these attacks rely on misconfiguration of IP spoofing protection, or leveraging other mechanisms, such as FWZ encapsulation, for insertion.

### 4.1 FTP Data Connections

#### 4.1.1 FTP PORT

FireWall-1's FTP processing tries to restrict the destination of FTP data connections to the FTP client associated with the control connection, preventing the classic FTP bounce attack. In previous versions of FireWall-1, this was trivially bypassed by splitting up the PORT command over multiple packets, or even by spelling "PoRT" in mixed case for version 3.0. However, with the latest checks imposed by FireWall-1 to restrict one FTP command to a packet, bypassing this becomes a little more difficult.

In our presentation, we demonstrated an ambiguity in the parsing of the PORT command that allows us to bypass this restriction. FireWall-1 takes each octet of the presented IP address as a long integer,

shifts it the appropriate number of bits and then adds it to its notion of the IP address. However, the FTP server we targeted on Solaris 2.6 interprets each octet of the IP address modulo 256. This difference allows us to trick the firewall into believing the connection is destined for the correct client IP, while the FTP server believes the connection is destined for itself.

The attack we demonstrated was against the ToolTalk RPC daemon. We uploaded two files to the FTP server in a publicly writable directory, and then instructed the FTP daemon to upload those files to its own ToolTalk daemon port with a carefully crafted PORT command. The first file was necessary to kill the daemon, and the second file was the buffer overflow, as generated by the ToolTalk exploit by apk.

We also demonstrated utilizing FWZ encapsulation to spoof a FTP control connection packet from our victim machine to our attacking computer. The payload of this packet was a PORT command, which the firewall's FTP stateful inspection module interpreted as a legitimate data connection that needed to be passed through the firewall.

#### 4.1.2 FTP PASV

We originally intended to disclose this attack at our presentation, but it was independently discovered and published by Mikael Olsson to VULN-DEV. This vulnerability can be referenced under CVE-2000-0150 [CBHM99].

FireWall-1 violates a basic layering abstraction in its inspection of TCP application data within discrete IP packets, allowing for a simple insertion attack. We can leverage the quoting of client input in an FTP server's error messages and the client-side configuration of TCP connection parameters to trick the firewall into allowing arbitrary data connections for PASV replies we generate ourselves.

To demonstrate, we used our "ftp-ozone" tool to open up a direct connection to the ToolTalk daemon on the FTP server, allowing us to execute a buffer overflow attack.

## 4.2 Simplex TCP Connections

A connection in FireWall-1's connection table may be flagged to restrict data flow to one direction (One-Way versus Either-Way). The first packet containing TCP data establishes the direction of the connection, and any data travelling in the other direction will cause the connection to be dropped from the connections table. In versions of FireWall-

1 prior to 4.1, the check for data only considered the first fragment in a series of fragmented packets. Thus, it was possible to bypass this restriction by sending a TCP data packet in two IP fragments – the first containing the TCP header, the second containing data (a variation on the tiny fragment attack against static packet filters [ZRT95]).

Check Point corrected this in the latest version, but there is still another way to bypass it. It is important to note that the firewall does not attempt to tear down the connection; it only removes its entry in the connection table and drops any subsequent packets. As FireWall-1 doesn't verify TCP sequence numbers [Ro00], it is possible to re-establish the connection with the same source and destination parameters, allowing for retransmission of the previously dropped TCP data to set a new direction for the connection, passing the data back through the firewall. To exploit this, a small program running in an infinite loop that constantly re-opens the FTP data connection can be used to leak the return data through, although somewhat inefficiently.

## 4.3 RSH stderr

RSH is another protocol that permits an insertion attack to allow back-connections through the firewall, in the same manner as the previous FTP PORT exploit. This requires a legitimate RSH client inside the firewall, with the firewall configured to allow RSH/REXEC stderr connections (a simple policy box check, without any rules required for RSH in the rulebase).

RSH stderr connection handling also presents a problem in its management of connection state. Although we are unable to exploit this vulnerability in versions 4.1 and later, we decided to leave discussion of it in our presentation as it nicely illustrates some of the state-tracking mechanisms FireWall-1 uses internally.

When FireWall-1 sees the first SYN of an RSH or REXEC connection, it records the source address, source port, destination address, a magic number, and the TCP sequence number + 1 into the pending table. It records this information so that it can recognize the first packet of the connection that contains data. When the first data packet of the RSH connection is passed by the firewall, its entry in the pending table is replaced with a new one, containing the source IP address, the error port (extracted from the TCP payload), the destination IP address, the same magic number, and the IP protocol. Upon intercepting the first SYN for an error connection, the firewall checks its parameters against the entry

in the pending table, and adds the connection to the connections table.

An exploitable collision exists in the use of the pending table. If we provide an initial sequence number of five, it is incremented to six in the pending table – identical to the entry created by the first-packet handling code. Thus, by spoofing a SYN from an intranet or DMZ machine with an ISN of five, with the source port of the host that we wish to access, the destination IP address of our attacking machine, and the destination port of the RSH service, an entry is added to the connections table allowing us to connect from our machine to the “RSH client” on whatever port we specify.

In version 4.1, the problem is not exploitable due to the fact that the initial SYN packet is not passed, as certain additional flags are missing from our manufactured pending table entry.

## 4.4 UDP Replies

A default installation of FireWall-1 4.0 allows DNS traffic between all hosts. When combined with lax IP spoofing protection, this rule allows us to spoof UDP datagrams from the DMZ or intranet to our attacking machine. If UDP responses are allowed, we can reply to those packets – in effect allowing us to talk to any UDP service behind the firewall.

Our demonstration consisted of an attack against the SNMP daemon on the target Solaris 2.6 machine. Using the “tun” tool, we sent an FWZ encapsulated datagram to the external interface of the firewall, which upon decapsulation, appeared to originate from the SNMP service on the internal victim, destined for our attacking host’s port 53/udp.

This allowed us to respond to our “DNS request” with arbitrary packets destined for the target’s port 161/udp. It would follow that our source port should be 53/udp, but this isn’t actually necessary because of the way UDP replies are implemented.

## 5 Recommendations

### 5.1 Non-reliance on NAT

As mentioned above, FWZ encapsulation can be used to send packets through the firewall that would otherwise be unroutable. This includes packets destined to addresses behind NAT or on RFC 1918 private networks. For instance, in FireWall-1’s default configuration, an attacker can access DNS servers in the intranet or DMZ directly, due to the “allow all port 53” rule. If the ICMP “allow all” rule is

enabled, an attacker can map any intranet or DMZ subnets.

NAT and non-routable addresses do not necessarily provide any inherent security. While FWZ encapsulation is specific to FireWall-1, similar attacks using other VPN encapsulation techniques may be more widely applicable – GRE, IP protocols reserved for IPsec, and even IPv6 tunnelling in IPv4 [Ha00].

## 5.2 Configuration

A few simple rules of thumb:

1. Be sure to run the latest version of FireWall-1 possible.
2. Disable implicit rules, including those for DNS, administrative control connections, and ICMP.
3. Be as specific as possible in defining rules – no “any” source or destination IP addresses, deny multicast / broadcast addresses, etc.
4. Enable IP spoofing protection on all interfaces.
5. Pre-filter IP protocol 94 (e.g. with IP Filter or some other mechanism).
6. Separate virtual IP addresses for public services.
7. Use only FWA1 authentication for now.

## 5.3 Patches

Check Point released patches to address these problems in time for our presentation at the Black Hat Briefings. Current service packs, hotfixes, and alerts are available at:

<http://www.checkpoint.com/techsupport/>

## 6 Acknowledgments

We would like to thank Check Point for their timely, thorough response to our independent security review, and for their exceptional professionalism in dealing with these issues.

## References

- [Be89] S. Bellovin, "Security Problems in the TCP/IP Protocol Suite". *Computer Communications Review* 2:19, pp. 32-48, April 1989.
- [Ch92] D. Brent Chapman. "Network (In)Security Through IP Packet Filtering". 3rd USENIX Security Symposium, September 1992.
- [CBHM99] S. Christey, D. Baker, W. Hill, D. Mann. "The Development of a Common Vulnerabilities and Exposures List". 2nd International Workshop on Recent Advances in Intrusion Detection, September 1999.
- [Ha00] J. Hagino. "Possible Abuse Against IPv6 Transition Technologies". Internet Draft draft-itojun-ipv6-transition-abuse-00a.txt (work-in-progress), Internet Engineering Task Force, March 2000.
- [Fy97] Fyodor, "The Art of Port Scanning". *Phrack Magazine* 7:51, article 11, September 1997.
- [Ha95] N. Haller. "The S/KEY One-Time Password System". RFC 1760, Internet Engineering Task Force, February 1995.
- [ZRT95] G. Ziemba, D. Reed, P. Traina. "Security Considerations for IP Fragment Filtering". RFC 1858, Internet Engineering Task Force, October 1995.
- [PN98] T. Ptacek, T. Newsham. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". Secure Networks, Inc., January 1998.
- [Ro00] G. van Rooij. "Real Stateful TCP Packet Filtering in IP Filter". 2nd International SANE Conference, March 2000.