Network forensics on open-source firewalls through analysis of linux core dumps (revised April 2019)

Jay Chow, *Graduate Student Researcher*, *Johns Hopkins University Information Security Institute*

Abstract— A firewall acts as a barrier between a trusted network and an untrusted network, controlling both incoming and outgoing network traffic based on rules and traffic content. Forensics is the process of using scientific knowledge for collecting, analyzing, and presenting evidence to the courts. It is the use of science and technology to investigate and establish facts in criminal or civil courts of law. Another important yet non-traditional source of forensic data is the contents of volatile memory. By examining the contents of RAM, an investigator can determine a great deal about the state of the firewall when the memory dump is collected. The paper seeks to bring these two areas of research together by allowing investigators to apply forensics on core dumps of open-source firewalls such as IPFire, OPNSense and Smoothwall when firewalls crash. Why did it crash? How can we better understand the crash forensically?

1 Introduction

A firewall is an important security technology that is widely deployed in networks, restricting network traffic from one side of the network to the other side. A firewall is part of a computer system or network that is designed to stop unauthorized traffic from flowing from one network to another. Firewalls can be deployed anywhere, and are most commonly seen separating trusted and untrusted components of a network. Firewalls are also useful in differentiating networks within a trusted network. It can create a clear distinction between various divisions in an organization. Firewalls can be implemented in either hardware or software, or both. The main functionalities include filtering data, redirecting traffic, protecting against network attacks.

A well-designed firewall has:

- All traffic between 2 trust zones should pass through the firewall. This is a tough challenge with network spawned by a large enterprise where people can create back-doors to the network
- Only authorized traffic, as defined by the security policy, should be allowed to passthrough
- The firewall itself must be immune to penetration, which implies using a hardened system with secured OS

1.1 Firewall policy

A firewall is only as effective as the rules that are being enforced by it. A firewall policy defines different kinds of rule that should be enforced. The rules are normally defined to provide the controls for the traffic on the network:

- User control: Controls access to the data based on the role of the user who is attempting to access it. This is applied to users inside the firewall perimeter.
- Service control: Access is controlled by the type of service provided by the host that is being protected by the firewall. This control is enforced on the basis of network address, port numbers and the protocol of connection.
- Direction control: This determines the direction in which requests may be initiated and are allowed to flow through the firewall. The direction of flow signifies whether data are flowing from the network into the firewall(inbound) or vice versa(outbound).

1.2 Firewall Actions

Network packets flowing through the firewall can result in:

- Accepted: Allowed to enter the connected network or host through the firewall
- Denied: Not permitted to enter the other side of the firewall
- Rejected: Similar to denied but an attempt will be made to tell the source of the packet about this decision through a specifically crafted ICMP packet

Firewalls can inspect network traffic from both directions.

Ingress filetering is safeguarding an internal network and prevent potential attacks from outside, it inspects incoming network traffic. Egress filtering is preventing users in the internal network from reaching out to certain destinations or send a certain type of data out.

For example, a large-scale egress firewall is the Great Firewall of China, which blocks the access of many sites such as Facebook, Youtube and Google from within.

1.3 Types of Firewalls

1)Packet Filter Firewall(stateless): Make decisions based on each individual packet. The decision is based on the information in the packet headers, without looking into the payload that contains application data. It does not pay attention to whether the packet is part of an existing stream of traffic. The advantage is speed as it does not maintain the states about the packets. This type of firewalls must permit traffic on a large range of port numbers for this to function properly.

2)Stateful Firewall: Make decisions based on state information formed by multiple packets that are related. A stateful firewall tracks the state of traffic by monitoring all connection interactions until it is closed. This firewall retains packets until enough information is available to make an informed decision about the state of the connection. A connection state table is maintained to understand the context of the packets.

3)Application/Proxy Firewall: An application firewall controls inputs, outputs and access from/to an application or service. This inspects network traffic up to the application layer. A typical implementation of application firewall is proxy. This acts as an intermediary by impersonation. The client's connection terminates at the proxy, and a corresponding connection is separately initiated from the proxy to the destination host. Data is analyzed up to the application layer. This protects the internal host from risk of direct interaction, providing a higher level of security. An example of this is to prevent sensitive information from being leaked to the outside. The disadvantage is the need to implement new proxies to handle new protocols. And the advantage is to authenticate users directly rather than depending on network addresses of the system. This reduces IP spoofing attacks.

2 Open-Source Firewalls

2.1 Review Stage

Since the source code of most top proprietry firewalls from Checkpoint, Fortinet, Cisco, Palo Alto are not available, I will look at open-source firewalls such as IPFire, OPNSense and Smoothwall, and apply forensics analysis and techniques on these three open-source firewalls with source code available.

2.2 IPFire

IPFire is built on top of netfilter and trusted by thousands of companies worldwide. With customization flexibility, it can be used a firewall, proxy server or VPN gateway, depending on the configurations.

IPFire focuses on security. Its easy to configure firewall engine and Intrusion Detection System prevent any attackers from breaking into your network. In the default configuration, the network is split into various zones with different security policies such as a LAN and DMZ to manage risks inside the network and have custom configuration for the specific needs of each segment of the network. IPFire is built from scratch and not based on any other distribution. This allows the developers to harden IPFire better than any other server operating system and build all components specifically for use as a firewall. Frequent updates keep IPFire strong against security vulnerabilities and new attack vectors.

IPFire uses a stateful packet inspection firewall, built on top of Netfilter, the Linux packet filtering framework, filtering packets fast and achieves throughputs of up to multiple tens of Gigabit/s. Its web interface allows the user to create groups of hosts and networks which can be used to keep large set of rules short and tidy, which is important in complex environments with strict access control. Logging and graphical reports give good insight. For example, settings are available to mitigate and block DoS attacks by filtering them directly at the firewall and not allowing them to take down the servers. A main use case of IPFire is a fully-fledged web proxy, delivering and filtering web content and can allow internet access only to some users.

2.3 OPNSense

OPNSense is a fork of pfSense and m0n0wall. The graphical user interface is available in multiple languages like French, Chinese, Japanese, Italian and Russian. OPNSense got many enterprise levels of security and firewall features like IPSec, VPN, 2FA, QoS, IDPS, Netflow, Proxy, Webfilter, etc. It is compatible with 32bit or 64bit system architecture and available to download as ISO image and USB installer. Some of the core features include:

- Traffic Shaper
- Two-factor Authentication throughout the system
- Captive portal
- Forward Caching Proxy (transparent) with Blacklist support

- Virtual Private Network (site to site & road warrior, IPsec, OpenVPN & legacy PPTP support)
- High Availability & Hardware Failover (with configuration synchronization & synchronized state tables)
- Intrusion Detection and Prevention
- Build-in reporting and monitoring tools including RRD Graphs
- Netflow Exporter
- Network Flow Monitoring
- Support for plugins
- DNS Server & DNS Forwarder
- DHCP Server and Relay
- Dynamic DNS
- Encrypted configuration backup to Google Drive
- Stateful inspection firewall
- Granular control over state table
- 802.1Q VLAN support

2.4 Smoothwall Express

The Smoothwall Open Source Project was set up in 2000 to develop and maintain Smoothwall Express - a Free firewall that includes its own security-hardened GNU/Linux operating system and an easy-to-use web interface. Its features include:

Hardware Support:
- Any Pentium class CPU and above – with a recommended minimum of 128MB RAM
- 64bit build for Core 2 systems

Firewalling:
- Supports LAN, DMZ, and Wireless networks, plus Extrnal.
- External connectivity via: Static Ethernet, DHCP Ethernet, PPPoE, PPPoA using various USB and PCI DSL modems.
- Portforwards, DMZ pin-holes
- Outbound filtering
- Timed access
- Simple to use Quality-of-Service (QoS)
- Traffic stats, including per interface and per IP totals for weeks and months
- IDS via automatically updated Snort rules
- UPnP support
- List of bad IP addresses to block

Proxies:
- Web proxy for accelerated browsing
- POP3 email proxy with Anti-Virus
- IM proxy with realtime log viewing

UI:
- Responsive web interface using AJAX techniques to provide realtime information
- Realtime traffic graphs
- All rules have an optional Comment field for ease of use
- Log viewers for all major sub-systems and firewall activity

Maintenance:
- Backup config
- Easy single-click application of all pending update
- Shutdown and Reboot from UI

3 Project Execution

My paper provides a starting point to understand the reasons for the crash of firewalls by using core dumps. Why did the firewalls crash? How did the firewall crash? How can we apply digital forensics to better understand the crash of the firewall software? Since the source code of each firewall is available, I could generate a core dump to analyze the contents of the RAM when the firewall software is running on a machine. A core dump, or memory dump, is a file of a computer's memory of when a program is crashed. The file consists of the recorded status of the working memory at an explicit time, usually close to when the system crashed or when the program ended atypically.

A core dump file may include additional information such as the processor's state, the processor register's contents, memory management information, the program's counter and stack pointer, operating system and processor information and flags. RAM is flat, and on production systems, some traffic that passes through a firewall may cause it to crash. By analyzing the core dump due to a crash, we could view the data structures and variables that existed at the time of the crash. However, this is a very tedious task. To unserialize the RAM to make the core dump more human-readable for forensic analysis, I could use gdb scripts to anayze what went wrong in the firewalls from a forensic viewpoint. For example, I could use gbd to open a core dump file and get a backtrace to know what the stack was when the firewall program crashed. Alternatively, I could write a python script that takes the core dump file as input and run certain functions on the core dump. This will give a networks forensics examiner more information to understand what caused the crash in the firewall. To be more specific, a python function could be written to iterate the linked list data structure in the data section of the core dump to figure if some data were corrupted or modified unintentionally. Note that the code section is also part of the memory dump but in this paper, I will be focusing on the data section of the memory dump.

Other low-level foresnics tools include running the firewall program and while it is running before the crash, I could dump the virtual memory tables by first identifying the PID of the firewall process, then running sudo cat/proc/<pid>/map on the command line. Aditionally, I could run strace on the firewall process. The strace command traces system calls and signals. In the simplest

case, strace runs the specified command until it exits. It intercepts and records the system calls which are called by a process and the signals which are received by a process. The name of each system call, its arguments and it's return value are printed. It is a useful diagnostic, instructional and debugging tool. System administrators and trouble shooters will find this useful with programs for which the source is not available. Since system calls and signals are event that happen at the user/kernel interface, the close examination of this boundary is useful for bug isolation, sanity checking and attempting to capture race conditions.

4 Conclusion

A concrete implementation of this digital forensic analysis on core dumps using gdb scripts or a python program can be written and executed to better understand a firewall crash.

5 Acknowledgement

The author wishes to thank Professor Timothy Leschke for his guidance and mentorship for his courses on Computer Forensics(EN.650.656) in the Fall of 2018 and Advanced Computer Forensics(EN.650.757) in the Spring of 2019 at Johns Hopkins University Information Security Institute.

6 References

[1] C. Kumar, https://geekflare.com/best-open-source-firewall'/
[2] J. Chow, H. Luo, "Linux Firewall Exploration", Security and Privacy in Computer Lab 6
[3] J. Chow, K. Hamilton, Y. Wang, "Reverse Engineering for Lwan, a high-performance and scalable web server"/
[4] H. Bensefia, N. Ghoualmi, "A Multi-Agent System for Firewall Forensics Analysis"
[5] IPFire's features, https://www.ipfire.org/features
[6] OPNSense, https://opnsense.org
[7] Smoothwall, http://www.smoothwall.org