Linux Application Firewall: Towards a Modern Per-Application Packet Filtering Implementation

Peter Maynard
Uncharted Security Research, UK
p.maynard@unchartedsecurity.com

***Abstract*—Current Linux Application Firewall (LAF) implementations, do not take full advantage of the Linux kernel. Which results in a bad experience for end users, ultimately, leaving the idea dead in the water. Background chatter on the Internet, shows there is an interest in a desktop application firewall, that can provide the average user with an additional layer of security. While expert users advocate the use of complex Mandatory Access Control (MAC) systems, such as SELinux and AppArmor to provide the same level of protection. This work attempts to find a middle ground between the two.**

I. Introduction

A Linux Application Firewall (LAF), is a host based packet filter with rules generated on a per-application bases, instead of IP/Port. Similar to servers' Web Application Firewall (WAF), which filters the content for a specific web application. However, a LAF differs, in that it is for desktop use, and filters content for applications found in a typical Linux distribution (e.g. calculator, word processor, or web browser).

Linux desktop users enjoy an open, secure, and privacy focused operating system. Yet, the average user is still missing one important feature, which other operating systems provide. That is, the fine grained option of allowing or denying network access for specific applications. Often, egress filtering is disabled, allowing any application to transmit on the network. While one has to place a certain amount of trust in the host operating system, the same amount of trust can not be afforded for third party applications. While current systems[1], such as Mandatory Access Control (MAC), SELinux, and application whitelisting are appropriate for a server environment, these are not easily operated by the average user since it requires an in depth understanding of the underlying system.

II. Existing Work

For an application firewall to operate, (a) it requires the PID of the application requesting network access as well as a (b) related rule, and the ability to (c) filter packets. Existing implementations do this either with the NetLink interface in the kernel, as a kernel module. Or, by using the proc process information to map the PID to the network socket being opened (`/proc/net/{tcp/udp}`), then queue the packet on the Netfilter queue for a verdict, i.e. allow or deny.

Both of these methods have disadvantages. The kernel approach, while fast and efficient, requires untrusted code to be loaded into the kernel, which can reduce the robustness of the overall system. While the procs method, provides more assurances, since runs in userspace. Yet, it incurs additional overhead and performs slower than the kernel method.

III. Proposed Approaches

As discussed above, there are currently two approaches, use the process information mapping or develop a kernel module. However, one may employ alternative approaches, such as eBFP or application sandboxing via CGroups and namespaces. Fig. 1 shows how these approaches align against each other in terms of userland or kernel space.

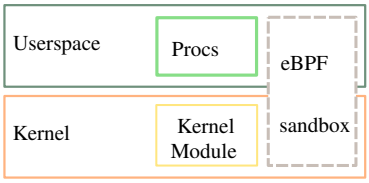

Fig. 1. Linux Application Firewall (LAF) implementation approaches.

eBPF is a new feature of the Linux kernel, which allows verified, secure, and lightweight VMs to be attached to a code path within the kernel, which it takes over. This allows extending the kernel without compromising the robustness of the whole system. By implementing a LAF using eBFP, it would be possible to achieve the same results as current implementations, but faster and more secure.

Alternatively, one might place all applications into isolated network namespaces, thereby providing each application with a networking stack (or without), and firewall rule sets.

IV. Future Work

Both of these proposed approaches will provide granular application firewall, superior in terms of security, and performance of existing implementations. A LAF implementation may form one part of a ZeroTrust network implementation. Further work may also be performed to investigate what behaviours would be required for an average desktop user, e.g. temporary allow application when on a public network.

References

[1] Theogene Hakiza Bucuti, Ram Dantu, and Kirill Morozov. "CMCAP: Ephemeral Sandboxes for Adaptive Access Control". In: *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*. SACMAT '19. Toronto ON, Canada: Association for Computing Machinery, May 28, 2019, pp. 207–212.