Fighting with Firewalld

Architecting a Host-based Firewall Policy and the Challenges of Implementation

Erick McGhee
Stephen Milton
Adam Howard
Gary Rogers
Tom Krobatsch
emcghee5@utk.edu
smilton1@utk.edu
ahoward@utk.edu
grogers3@utk.edu
tkrobats@utk.edu
University of Tennessee
National Institute for Computational Sciences
Knoxville, Tennessee, USA

ABSTRACT

Host-based firewalling is an important part of a comprehensive security plan. This paper explores a robust solution that can be adapted and used at any institution, along with the challenges involved in its development. The solution implements custom host-based firewall policies via configuration management software, accounting for external networks sanctioned by the United States while striving to maintain adaptability and portability.

CCS CONCEPTS

• **Security and privacy → Firewalls**; **Distributed systems security**; • **Networks → Firewalls**.

KEYWORDS

firewall, security, network, networking, linux, firewalld, Ansible, policy, zones

ACM Reference Format:
Erick McGhee, Stephen Milton, Adam Howard, Gary Rogers, and Tom Krobatsch. 2024. Fighting with Firewalld: Architecting a Host-based Firewall Policy and the Challenges of Implementation. In *Practice and Experience in Advanced Research Computing (PEARC '24), July 21–25, 2024, Providence, RI, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3626203.3670523

1 INTRODUCTION

The National Institute for Computational Sciences (NICS) at the University of Tennessee, Knoxville (UTK) is a vital research arm of the university with an expansive operational range. NICS maintains clusters in partnership with various industrial entities. Operations are not limited to computational resources but also extend to web interfaces, user documentation sites, several private GitLab[3] instances, and a Kubernetes[2] cluster. Center staff also maintain all of the supporting infrastructure necessary to ensure these tools function optimally and reliably.

As a high-performance computing (HPC) center, NICS is exposed to numerous potential attack vectors that malicious actors could exploit. These potential vulnerabilities stem from the diverse array of services and platforms that could potentially serve as entry points for cyber threats. The situation is further compounded by the center's standing within a higher education institution - a sector increasingly targeted by cyber attackers[4].

NICS places paramount importance on cybersecurity. The center strives to minimize attack vectors and prevent any potential security breaches. Over the past year, the operations team has embarked on an initiative to strengthen security across all facets of the center. This endeavor involved numerous measures and projects to bolster its defenses and mitigate risks. These efforts have resulted in approximately 40% of all incoming network traffic to the center being identified as threats or suspicious activities and getting blocked or dropped. Notably, around half of this blocked traffic originates from countries identified by the US Office of Foreign Assets Control (OFAC) as severely sanctioned due to their association with significant cybersecurity risks[23].

One significant aspect of this initiative was revising host-based firewall configurations. While a network-based firewall provides robust defense mechanisms, hosts in a Demilitarized Zone (DMZ) lack this protection layer. Moreover, should an attacker breach the system's initial defenses, host-based firewalls are crucial in preventing further spread beyond the original attack vector.

2 MOTIVATION

Work on this project began as support for CentOS 7[26] was ending. This necessitated migrating to a newer operating system for infrastructure systems and the center's research computing platform. It was decided that the center would adopt Red Hat Enterprise

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PEARC '24, July 21–25, 2024, Providence, RI, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0419-2/24/07
https://doi.org/10.1145/3626203.3670523

Linux® (RHEL) 8 for the compute platform and RHEL 9 for infrastructure systems[19]. With the release of RHEL 8 came a shift in the low-level firewall technology from iptables[21] to nftables[22]. In many cases, this change would have been non-impacting due to the expectation that most users of RHEL would manipulate the firewall via the higher-level firewalld[9] toolset. However, the team at NICS extensively used firewalld's direct rules interface to continue implementing the firewall policy using iptables, which would no longer be supported on RHEL 8 and beyond. This was seen as an opportunity to review and consolidate existing firewall policy and usage, and a task force was formed to evaluate the current usage patterns and plot a path forward.

Ansible is the preferred configuration management software at NICS[17]. The first step was a review of the existing code for how the firewall policy was being implemented. As previously stated, many rules were still being written in direct rules for iptables. This was partially due to a misunderstanding of how many of the rules in place could be implemented using features of firewalld. Additionally, many Ansible roles for setting up various software packages added a service to the default firewall zone, allowing traffic to that service from any source in that zone.

In most cases, the `drop` zone was set as the default, and all rules were applied to it. However, there were some cases where the default zone had been manually changed for functional reasons and was mainly used to support dual-homed hosts. This configuration drift resulted in errors when running Ansible against those hosts. It also made it difficult to trace problems. There wasn't a simple way to determine what the correct configuration should be.

One significant difficulty was a lack of understanding within the team of how the zones, sources, interfaces, services, and other firewalld primitives came together to form a ruleset. Initially, this resulted in an attempt to abandon firewalld altogether and directly utilize nftables as detailed in Section 5.1. This proved to be a task that would require significantly more time and effort than expected. The team's focus was shifted onto firewalld cross-training, followed by a significant re-evaluation and re-implementation of firewall policy throughout the existing Ansible code base. The results of this effort were a much more concise and consistent implementation of firewalld. The team gained a better understanding of the technology and how it was to be used. This allowed for the creation of a more robust security posture that is focused on limiting vectors for attack and aims to prevent a situation where one compromised host may spread to others.

3 RELATED WORK

NICS started with the intention of migrating from iptables to nftables. With that in mind, NICS investigated possible existing projects that may have eased the transition. The most popular related project on Ansible Galaxy[18] at the time was ipr-cnrs.nftables[12], a collection of roles designed to manage nftables from one central role. That approach conflicted with the team's desire to build a simple base configuration and add additional rules from separate roles that defined individual services. Another work considered was ansibleguy.infra_nftables[1]. At the time, the project was still in early development and lacked most of the core functionality that was sought.

The team had experience with using the existing Ansible Community firewalld module to interact with the service, but found it limited[28]. The module was designed in a way that allowed for the direct editing of existing firewalld zones. It granted the ability to set a zone's target, add and remove sources, allow and deny ports and services, assign interfaces, modify masquerade settings, configure the behavior of ICMP blocks, and add rich rules. It also provided the ability to make the changes immediate and permanent. While useful, the design of this module was targeted at making incremental changes to an existing firewalld configuration, whereas NICS was searching for a way to architect policies from the ground up. NICS eventually used this module to add additional rules from other roles as detailed in section 5; however, the lack of ability to implement custom firewalld objects such as zones and ipsets limited its usefulness for creating a custom solution.

Once it was decided to shift focus from nftables to firewalld, other implementations were sought. One of the most mature projects found was gnfzdz.network[5]. This project applied its configurations utilizing a mix of the functionality provided by the default ansible.posix.firewalld module and sending firewalld-cmd commands via ansible. This meant that each firewalld object created would need to be modified multiple times during the configuration process, drastically increasing processing overhead and possibly causing unneeded reapplication of the same changes.

4 POLICY CONSIDERATIONS

NICS implements a robust host-based firewall system using firewalld. The NICS local firewall policy starts with the concept of least privilege. If a host doesn't need access, it is blocked from the starting configuration. For example, only the bastion and configuration management hosts can SSH into every node. The only exception is ICMP, as pings are allowed from any host to any other internal hosts. In the past, NICS had the opposite configuration where only monitoring hosts could ping another internal host. This, however, made troubleshooting and diagnosing networking issues difficult.

4.1 Zones

NICS uses a system that classifies hosts into five distinct zones, each with unique characteristics and functional roles in network security.

4.1.1 Internal Zone. The Internal Zone is at the heart of the NICS network. It encompasses all hosts residing within the subnets of the center's infrastructure or DMZ areas. These hosts are integral to operations and thus necessitate stringent security measures to safeguard against external threats.

4.1.2 Cluster Zone. Next is the Cluster Zone, which houses all compute nodes. This is an area of potential vulnerability due to less restrictive policies that permit end users to run their own code. The inherent risks associated with this flexibility mean access from this zone to other parts of NICS is more restricted than from the Internal Zone.

4.1.3 Campus Zone. Moving beyond NICS's immediate environment is the Campus Zone, which represents the wider university community. This zone facilitates smoother collaboration across

partners by leveraging campus virtual private networks (VPNs) for certain activities, thus balancing accessibility with security.

4.1.4 External Zone. The External Zone captures everything outside the confines of campus and essentially serves as a catch-all for rules meant for open Internet connections. This zone handles interactions with external entities while maintaining appropriate safeguards against potential cyber threats.

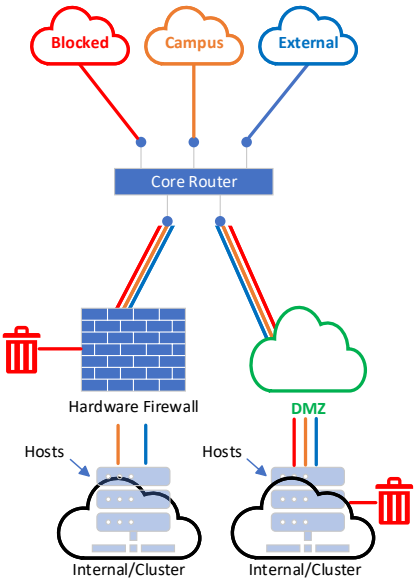


Fig. 1: Example network diagram of firewalld zones.

4.1.5 Blocked Zone. Finally, there's the Blocked Zone, primarily responsible for managing country sanctions lists based on recommendations from authoritative bodies due to a significant number of malicious attempted connections to the center's network[23]. These blocks are also implemented at the center's network-based firewall level, so they are currently only applied to the host-based firewalls of hosts that are not behind the network-based firewall.

Going forward, NICS aims to enhance its security posture by implementing a list of repeat offenders to block repeated failed connection attempts across all center resources rather than just one host. This proactive measure will serve as another layer in a multi-zone defense approach toward securing the computational resources and classifying different access levels into these five zones: Internal, Cluster, Campus, External, and Blocked, as seen in Figure 1. This ensures robust protection against varying degrees of cyber threats while allowing necessary access where needed.

4.2 Standardizing Firewall Rules

Before this initiative, rules were often not applied correctly as admins used different methods to configure hosts. To alleviate these problems, the team established a standard for creating rules. All rules associated with a service should reside in the configuration management role for that service.

To simplify rule reading, the team preferred configuring services instead of using ports. For instance, a web server role would add rules to hosts for ports 80 and 443 using HTTP and HTTPS services.

Furthermore, if a rule requires access from specific hosts only, it must have an ipset associated with it, and a policy should be created instead of merely applying the service to the zone in which the host resides. This means that in the web server role mentioned earlier, the services would need to be added to the External, Campus, and Internal zones for it to be accessible over the open Internet. If it were intended to be accessible only to specific internal servers, however, then an ipset would need to be created for those hosts, and a policy should be established utilizing that ipset along with the appropriate services.

Significantly, all interfaces are added to the external zone. While the team strives to avoid applying rules and policies directly to interfaces when possible, placing them in this zone is crucial in preventing accidental application of other policies.

4.3 Consistency

The new policies standardize firewall rule management to address past inconsistencies in applying rules due to varied host configurations. They stipulate guidelines for rule creation, promote using services over ports, and define specific procedures when rules are required to restrict host access including the strategic use of zone assignments. With added precautions against accidental policy applications and a reduction in necessary zone-wide policies, this approach significantly enhances the efficiency of firewall rule management, and ensures consistency across the center.

5 IMPLEMENTATION

The project went through several iterations before landing on the current implementation. The team initially attempted to migrate from iptables to nftables before settling on firewalld. This realignment resulted in a robust and portable solution that could be implemented at other institutions.

5.1 nftables Versus firewalld

The initial impetus for this project was the migration from iptables to nftables as the underlying technology behind firewalld on RHEL 8. Some members of the team expressed a preference to work directly with the underlying software. As a result, the first attempt of the project was to do away with firewalld entirely in favor of working with nftables directly. The team hypothesized that the resulting configurations would be more straightforward and easier to implement. This proved to be incorrect, and led to many issues, starting with a general lack of ready information and tooling support for nftables. Configuration management frameworks lacked sufficient support for manipulating nftables from the module level, and the community-supported modules lacked key features and testing. This made them unfit for use in a production environment, especially for such a crucial security component. The final option the team explored was the internal development of a nftables module within the center's chosen configuration management tool. Given the effort required to develop and maintain such a tool, this proved a significant challenge. The deciding factor was the discovery that certain services such as Docker[20] and KVM[27] create and maintain separate firewalld zones, making it a hard requirement on those systems. At this point, the team made the difficult decision to abandon efforts to utilize nftables without firewalld.

Essentially, having to start from scratch, the team's efforts were refocused on a deep dive into and cross-training on firewalld. This represented a significant realignment of the project but resulted

in a much richer understanding of the tool, how it worked, and the quirks of the various versions the team would have to support. After changing course, the necessary implementation steps and their execution came together remarkably quick.

5.2 Replacing Default Configuration

The first step in implementing the policy in firewalld was removing as many of the default zones as possible and replacing them with the zones outlined within the policy. This was straightforward by removing most of the files defined in `/usr/lib/firewalld/zones` and replacing them with custom zone definition files located in `/etc/firewalld/zones`. It was desirable to remove all of the out-of-the-box zones; however, in doing so, it was discovered that firewalld contained hard-coded assumptions about the existence of the pre-packaged `drop`, `blocked`, and `trusted` zones, that caused the service to fail if they weren't defined. The team worked around this by re-purposing the `drop` zone to fit within the policy and ignoring the other two as much as possible.

5.3 Ansible

Configuration management at NICS is performed via Ansible. This is accomplished by incorporating or developing roles that operate at the unit of a single service or system component, which are then collected into a playbook that pairs hosts with the roles necessary to implement their function. The roles themselves are generally intended to be generic and site-agnostic. Site-specific behavior is encoded within Host and Group Variable files, effectively decoupling them from the core configuration and allowing portability of the solution.

To implement the firewall policy, it was decided that a central role would be responsible for establishing the Zones, Default Policies, and ipsets while the establishment of individual rules would remain within the service roles to which they were required. For example, the role for installing and configuring a database server would include creating rich rules to allow only specified hosts to access it. This required a review of 133 existing roles to ensure their behavior was compatible with the new assumed firewall behavior, resulting in small to moderate changes in twenty-four. Support was added to the central firewalld role to provide additional miscellaneous rules that did not align with a specific service-providing role, however, efforts are made to limit the use of this function. Figure 2 describes an example of this.

Ansible tasks should be designed to be idempotent, meaning that they should be able to be rerun an arbitrary number of times without attempting to re-implement changes. To accomplish this, the decision was made to forgo the use of the `firewall-cmd` command line tool in favor of writing the Zone, Policy, and ipset definition files in raw XML using Ansible's Jinja 2[24] template engine. This introduced some issues, as firewalld's XML syntax is very strict, and while the grammar is described in the documentation, it lacks all but the most basic examples.

5.4 Blocking Sanctioned Countries

To comply with the organization's security policy, blocking specific IP ranges based on country of origin was necessary if that country was subject to United States sanctions. This was implemented using

```
portal_dev__nics_firewalld_rules__to_merge:
  - rich_rule: rule family=ipv4 priority=2 port
    port="8000-8010" protocol=tcp accept
    zone: internal
  - rich_rule: rule family=ipv4 priority=2 port
    port="8000-8010" protocol=tcp accept
    zone: campus
```

Fig. 2: This snippet is taken from the host variables for a host `portal-dev`, which is used as a software development sandbox. Ports from 8000 to 8010 are available for web development test servers. When the `nics_firewalld` role is run against a host, it searches for any variables with the suffix `_nics_firewalld_rules__to_merge`, consolidates them into a single list of dictionaries, and then applies them with a task. The supported keys are just those supported by the `ansible.posix.firewalld` module.[28]

an ipset of network ranges associated with the countries in question via the IPDeny Project [25]. While these lists are kept up to date quite frequently, the decision was made to pull snapshots into the center's Ansible role repository and maintain them within the project's history. This would allow the auditing of changes to blocked ranges. The last change could then be identified and rolled back or modified if an issue was encountered.

The necessary country associations accounted for approximately 50,000 network segments, which made it essential to increase the size and hash length of firewalld's internal lookup table by setting additional options on the ipset definition. These changes were sufficient on RHEL 9, but issues remained on RHEL 8-based systems. Additionally, it was discovered that the size of such an ipset caused a significant increase in firewall reload times from a second to up to 30 seconds and an increase in memory usage from approximately 60MB to 130MB. Due to these issues, and this being a redundant measure for hosts behind the edge firewall that provides similar measures, it was decided to reserve this particular functionality for DMZ nodes or other hosts that needed to exist outside the protection of the edge firewall.

6 LESSONS LEARNED

During implementation, the team learned a lot about firewalld and how the software has evolved since its inception. NICS maintains servers that run different versions of multiple operating systems, including CentOS 7, RHEL 8, and RHEL 9. Each of these operating systems includes a different version of firewalld, and the team quickly discovered that features and configurations in the latest version were sometimes different or entirely missing from older versions. Unfortunately, many of these changes weren't reflected in the firewalld documentation. As a result, the team had to search through firewalld's official blog posts and the project's releases within its code repository to distinguish changes in behavior between versions.

One of the first problems the team encountered was the changes to how packets are filtered through the firewall with the addition

of policies and priority settings for zones and rules. Firewalld's current processing order is shown in Figure 3. When a packet comes through the firewall, the first things applied are direct rules or policies with a priority of less than zero. Policies were added in version 0.9.0[14], and the direct interface became deprecated in version 1.0.0[15]. Next, packets are filtered through zones. As of version 2.0.0, zones are processed in order of priority[16]. Zones with a priority less than zero are first, followed by zones with a priority equal to zero, and finally, zones with a priority greater than zero. In previous versions, or if two zones have the same priority in the current version, the zones are processed in alphabetical order[11]. This change has yet to make it into the firewalld documentation for zone options as of writing[8] but can be found in the manual pages for firewalld.zone[7]. As a packet is filtered through a zone, rules within the zone are applied in the order of rich rules, services, and, finally, default zone rules. Priority support for rich rules was added in version 0.7.0[13]. Rich rules are processed in the same order as zone priorities. Finally, any policies with a priority greater than zero are applied.

Another challenge the team faced was implementing the blocked networks list provided by the IPDeny Project. The team created the denied networks as an ipset. Firewalld defines ipsets within XML files that have a strictly defined structure[6]. The team discovered that having any whitespace between entry tags will cause parsing errors when starting the firewalld service. This was corrected by updating the Jinja template used to render the object. The next problem was regarding the number of ipsets that were being imported. The hashtable created to hold the rules in memory was not large enough to contain all entries. The team attempted to remedy this problem by increasing the size of the hash table, however, that was only successful on RHEL 9. As a result, the team decided to limit the deny lists to only be deployed on servers running RHEL 9 that are exposed to the open internet, such as those in the DMZ.

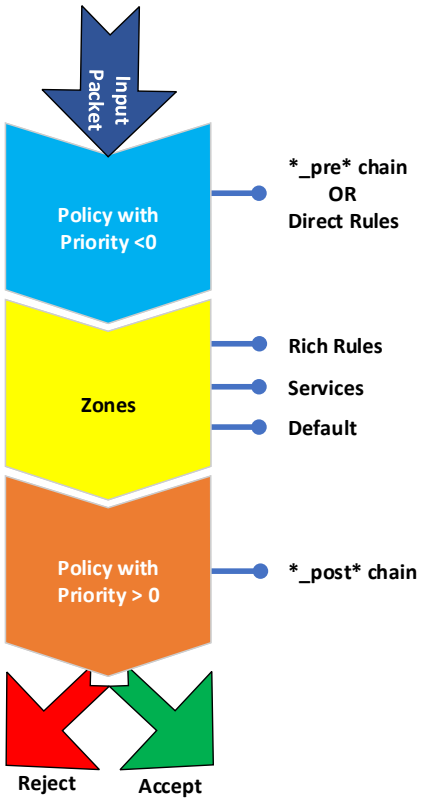


Fig. 3: Incoming packets are filtered through chains from policies and zones. Direct rules are deprecated as of version 1.0.0.

7 FUTURE WORK

The team's solution is designed to serve the needs of infrastructure servers. While considered feature-complete for this purpose, some work remains to be done. The team would like to extend the functionality to compute nodes within an HPC cluster. This would require optimizing the configuration to use as few system resources as possible while providing adequate protection that meets the center's security policy.

Another related goal is to develop a custom configuration for Kubernetes nodes. It would be preferable to develop a method for Kubernetes to have the ability to actively modify the configuration as needed for containers without the need for administrator intervention while being mindful of security requirements. This will require extensive testing to ensure the node and its running containers can allow the necessary connections without compromising security.

Finally, there is a need for additional documentation. During the development process, the team compiled an internal cookbook with examples of adding firewall rules to other Ansible roles. There is, however, little external documentation on the underlying firewalld role and the base configuration. Additional documentation would be helpful for new admins to quickly learn how the center's security policy is implemented. Firewalld's official documentation reflects the state of the project's current release. Changes of functionality between versions are spread across blog posts on the website. It would be helpful to create a centralized list of how functionality and features have changed across versions. It would also be useful to contribute a diagram and explanation of how packets are filtered through policies and zones along with the order in which rules are applied therein.

8 CONCLUSION

Host-based firewalls are an essential part of a comprehensive security strategy. The team at NICS has developed a dynamic solution that addresses the need to secure infrastructure while allowing for future needs to extend functionality and efficiently address changes to the underlying technologies. The solution presented here meets the center's needs but could be easily modified to work within the infrastructure of other institutions. The team also discovered and accounted for several discrepancies across versions of firewalld that may be deployed across varying operating systems. The solution is open-source and is available for public access on the NICS GitLab instance.[10]

REFERENCES

[1] AnsibleGuy. 2024. Ansible Galaxy - ansibleguy.infra_nftables. Retrieved March 5, 2024 from https://galaxy.ansible.com/ui/standalone/roles/ansibleguy/infra_nftables/
[2] Kubernetes Authors and The Linux Foundation®. 2024. Ansible Galaxy. Retrieved March 5, 2024 from https://kubenetes.io
[3] GitLab B.V. 2024. GitLab Community Edition. Retrieved March 5, 2024 from https://gitlab.com/rluna-gitlab/gitlab-ce
[4] Abrael Deldado. 2024. Who is the Prime Target for Cyber Attacks? Retrieved December 5, 2023 from https://www.compuquip.com/blog/prime-target-for-cyber-attacks-and-to-look-out-for
[5] Greg F. 2024. Ansible Galaxy - gnfzdz.network. Retrieved March 5, 2024 from https://galaxy.ansible.com/ui/repo/published/gnfzdz/network/
[6] firewalld. 2024. Documentation - IPSet - Examples | firewalld. Retrieved March 6, 2024 from https://firewalld.org/documentation/ipset/examples.html

[7] firewalld. 2024. Documentation - Manual Pages - firewalld.zones | firewalld. Retrieved March 6, 2024 from https://firewalld.org/documentation/man-pages/firewalld.zones.html
[8] firewalld. 2024. Documentation - Zone - Options | firewalld. Retrieved March 6, 2024 from https://firewalld.org/documentation/zone/options.html
[9] firewalld. 2024. firewalld. Retrieved March 5, 2024 from https://firewalld.org/
[10] National Institute for Computational Sciences. 2024. nics_firewalld_role. Retrieved June 12, 2024 from https://gitlab.nics.utk.edu/ops-public/ansible-code/nics_firewalld_role
[11] Andrew Galdes. 2019. Firewalld Zone Order/Priority And How To Restrict all but one address. Retrieved March 6, 2024 from https://agix.com.au/firewalld-zone-order-priority-and-how-to-restrict-all-but-one-address/
[12] Jérémy Gardais. 2024. Ansible Galaxy - ipr-cnrs.nftables. Retrieved March 5, 2024 from https://galaxy.ansible.com/ui/standalone/roles/ipr-cnrs/nftables/documentation/
[13] Eric Garver. 2019. Release firewalld-0.7.0 · firewalld/firewalld · GitHub. Retrieved March 6, 2024 from https://github.com/firewalld/firewalld/releases/tag/v0.7.0
[14] Eric Garver. 2020. Policy Objects: Introduction. Retrieved March 6, 2024 from https://firewalld.org/2020/09/policy-objects-introduction
[15] Eric Garver. 2021. The Upcoming v1.0.0. Retrieved March 6, 2024 from https://firewalld.org/2021/06/the-upcoming-1-0-0
[16] Eric Garver. 2023. firewalld 2.0.0 release. Retrieved March 6, 2024 from https://firewalld.org/2023/06/firewalld-2-0-0-release
[17] Red Hat®. 2024. Ansible. Retrieved March 5, 2024 from https://ansible.com
[18] Red Hat®. 2024. Ansible Galaxy. Retrieved March 5, 2024 from https://galaxy.ansible.com
[19] Red Hat®. 2024. Red Hat Enterprise Linux®. Retrieved March 5, 2024 from https://access.redhat.com/products/red-hat-enterprise-linux
[20] Docker Inc. 2024. Docker. Retrieved March 7, 2024 from https://www.docker.com/
[21] Netfilter. 2024. iptables. Retrieved March 5, 2024 from https://www.netfilter.org/projects/iptables/index.html
[22] Netfilter. 2024. nftables. Retrieved March 5, 2024 from https://www.netfilter.org/projects/nftables/index.html
[23] Office of Foreign Assets Control. 2024. Sanctions Programs and Country Information. Retrieved March 5, 2024 from https://ofac.treasury.gov/sanctions-programs-and-country-information
[24] Pallets. 2019. Jinja. Retrieved March 5, 2024 from https://jinja.palletsprojects.com/en/2.10.x/
[25] IPdeny.com Project. 2013. Jinja. Retrieved March 5, 2024 from https://www.ipdeny.com/about.php
[26] The CentOS Project. 2024. CentOS Linux. Retrieved March 5, 2024 from https://www.centos.org/centos-linux/#centos-linux-7-2009
[27] The KVM Project. 2024. KVM. Retrieved March 7, 2024 from https://linux-kvm.org/page/Main_Page
[28] Ansible project contributors. 2024. ansible.posix.firewalld module – Manage arbitrary ports/services with firewalld. Retrieved March 5, 2024 from https://docs.ansible.com/ansible/latest/collections/ansible/posix/firewalld_module.html