Firewall Attacks and Mitigation Techniques

Griffin Higgins
Faculty of Computer Science
University of New Brunswick
Fredericton NB, Canada
griffin.higgins@unb.ca

Vaishnavi Modi
Faculty of Computer Science
University of New Brunswick
Fredericton NB, Canada
modi.vaishnavi15@unb.ca

***Abstract*—Firewalls represent the first line of defence in any network as they are designed to control the flow of information in order to promote security. However, it is all too easy to forget that just because firewalls serve a special security purpose that it does not mean they are any more secure than other computing systems. In many cases, firewalls themselves are susceptible to attacks that can prove to have devastating effects on a network. In our work, we survey several such attacks on firewalls and explore their proposed mitigation techniques. Additionally, as part of our contribution, we develop a working Distributed Oblivious Firewall model according to one of our surveyed works. Lastly, we propose a simple improvement to help preserve the privacy of the model.**

***Index Terms*—Firewall, Firewall Attacks, Denial of Firewall, Firewall Fingerprinting, SQLi, Inside Attackers**

I. Introduction

Attacks on computer networks are increasing each day, putting the confidentiality, integrity, and availability of networked machines and the data within them at increased risk of attack [1]. As such, it is vitally important to ensure that networks have a strong first line of defence to deter potential attackers. Firewalls help fill this role by acting as a protective barrier between a private network and the wider public Internet [2]. While firewalls are certainly not perfect, they do help filter access to networks and prevent a large range of attacks. If used intelligently firewalls prevent attacks by ensuring that inbound and outbound traffic, in the form of IP packets, meet specific sets of conditions before being forwarded into or outside of a given network [3]. Typically, the conditional evaluation of packets, known as packet classification, is done by inspecting the five attributes of received IP packets against a rule-based engine or firewall policy [2], [3]. The source and destination IPs and ports as well as the IP protocol of a packet are usually evaluated. However, because firewalls serve such an important role in protecting computer networks from attacks it is important to consider how firewalls themselves can be vulnerable to attack.

In our paper, we begin by elaborating on firewalls by describing their statefulness, types, and several attacks in section II. In section III, we review several research papers that focus on firewall attacks and mitigation techniques. Furthermore, we model the Distributed Oblivious Firewall scheme as proposed by Goss and Jiang [4] and suggest an improvement in the model to secure firewalls in section IV. Lastly, in section V we conclude our paper.

II. Detailed Description

In order to evaluate the attacks on firewalls and their mitigation techniques, it is important to develop a good understanding of firewalls and their statefulness, types, and the different kinds of attacks that can occur.

A. Firewall Statefulness

Firewalls can be further classified based on their statefulness, that is, whether or not they retain connection state. In a stateless firewall, packets are inspected with the help of rules held in a rule table containing the five aforementioned attributes [2]. Once a packet is found to match a given rule an action to forward or block the packet is taken. Since multiple rules can exist the potential for conflict can arise. As such, firewalls adopt a first-semantic match policy [2]. Comparatively, in a stateless firewall packet inspection concentrates on matching a given packet's Transmission Control Protocol (TCP) flags against sessions held in the session table. Packets that are found to match a record in the session table bypass the firewall and are removed from the session table when the connection is torn down. Additionally, caching can be used to help improve the effectiveness of a firewall regardless of its statefulness.

B. Firewall Type

Firewalls typically fall into two types. They can be either hardware or software based. An example of hardware firewalls are routers installed in most homes to access the home network. With the help of these hardware firewalls, that is, routers, unauthorized access of attackers can be easily mitigated [2]. Hardware firewalls do not have to be updated regularly. They work on protecting all the devices on the network instead of protecting a single device [5]. Similar to hardware firewalls, software firewalls secure networks from attacks by monitoring the incoming and outgoing traffic and can block suspicious activity. Software firewalls can be conveniently installed on computers and other devices [2]. This allows for tailored network access permissions depending on the user's choice without excessive effort. However, software firewalls have to be regularly updated and monitored to guarantee their smooth functioning [5]. It is recommended by researchers to incorporate both software and hardware firewalls for a wider range of protection from network attacks.

C. Firewall Attacks

While many types of firewall attacks exist, several prominent attacks found in the literature include Denial of Firewall (DoF), Firewall Fingerprinting, SQL injection (SQLi), and Insider attack. We survey and describe these specific attacks in our related works.

III. Related Works

Trabelsi *et al.* [6] proposed a classification of DoF attacks that seek to exhaust firewall resources leading to DoS Denial of Service (DoS) for users. The authors present attacks against firewalls as primarily targeting a firewalls filtering rules or its session table. Attacks against filtering rules exploit the linear search complexity of the firewall having to search through many rules to potentially drop a packet. Thus, attacks with specially crafted packets designed to be filtered at a high index require more resources to filter and degrade firewall performance. The session table is also a target of firewall attacks since it is a limited resource designed to push authenticated packets through a session if they are found to match an entry in the session table. Attacks against the session table attempt to flood the table with illegitimate requests such that valid session connections are refused. The authors further classify attacks on a firewalls session table as being either volumetric (flooding based) or low volumetric (BlackNurse). In the volumetric or flooding based attack FTP handshakes are established between a host and attacker where the attacker hangs the connection at the FTP login screen. This generates an entry session in the firewalls session table that occupies space. FTP requests are made by the attacker until the session table is flooded through a SYN-ACK-ACK food process. ICMP based flooding can also be used by an attacker with similar results to ICMP echo requests. Low-volumetric attacks or BlackNurse attacks are also considered in the classification where attackers utilize specially crafted ICMP requests sent at a low volume to attack a firewalls session table. The attack works by sending ICMP error messages [Type: 3 (Destination unreachable), Code: 3 (Port unreachable)]. The messages are special because they try and exhaust the firewalls CPU by generating meaningless work rather than taking over the session table. This is used in the BlackNurse attack, shown in Figure 1, where an attacker requests a specific port on a destination machine behind the firewall.

The firewall makes a session entry and then forwards the request to the machine. Since the port at the machine is inactive, by attack design, an ICMP port unreachable error message [Type: 3, Code: 3] is returned back to the firewall. This is problematic because now the firewall must work to resolve the failed message in the session table to deliver the error message back to the attacker. The process is computationally taxing and can overwhelm firewall resources without having to flood it with requests.

To demonstrate their work the authors conduct an experiment to generate the BlackNurse attack with various firewalls such as Juniper NetScreen SSG 20 and ASA 5540 firewalls. The experiment demonstrates that in some cases

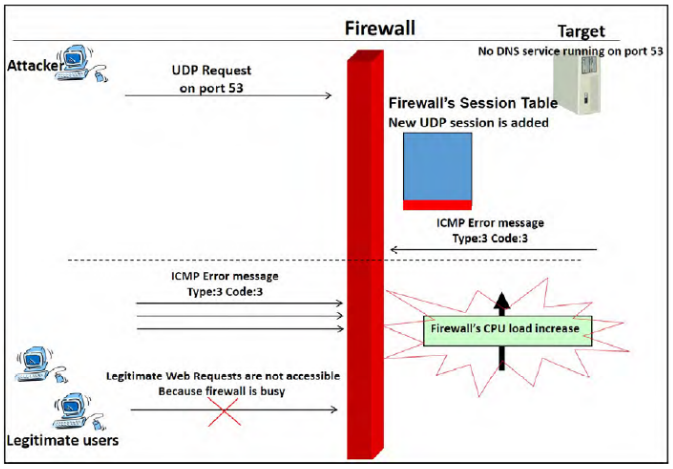
Firewall
Target
No DNS service running on port 53
Attacker
UDP Request on port 53
Firewall's Session Table
New UDP session is added
ICMP Error message Type:3 Code:3
ICMP Error message Type:3 Code:3
Firewall's CPU load increase
Legitimate Web Requests are not accessible Because firewall is busy
Legitimate users

Fig. 1: BlackNurse attack
Source: Adapted from [6, Fig. 11]

the BlackNurse attack can drive firewall CPU utilization to 89%, up from 2% in normal conditions, sixty seconds after commencing the attack. The authors propose that iptable-based firewalls should be used as an alternative since they rate limit ICMP [Type: 3, Code: 3] messages by default and thus are unaffected by the BlackNurse attack. Additionally, several mitigation strategies in the form of ICMP based firewall rules and general guidance on firewall flooding threshold configuration are provided. However, the authors mention that threshold configuration can still be the target of attack and are not always effective in the case of BlackNurse attacks. The main contributions of the paper address this issue and propose an adaptive early rejection rule with a dynamic time of activity or time to defend (TTD) depending on the statistics of attack history. Once a given minimum threshold of ICMP [Type: 3, Code: 3] is encountered then the rejection rule is activated for a dynamically calculated TTD. The rule triggering process is dependent on fake ICMP packets that are received and also not found to belong to an entry in the session table. The TTD is calculated based on the previous TTD and statistics of the rate of ICMP [Type: 3, Code: 3] messages that are used to predict the TTD needed for the firewall to regain control. The authors demonstrate that for a novice attacker an average TTD of 1.128s was required for the firewall to regain control and 1.284s for an experienced attacker (13% increase).

Hayawi *et al.* [7] expanded on their work in [6] to address a vulnerability in the early rejection rule designed to mitigate DoF attacks, specifically BlackNurse by blocking ICMP packets for a TTD duration. The authors identified that an attacker could use knowledge of the early rejection rule to increase the rule activity duration time and subsequent TTD continuously until all ICMP [Type: 3, Code: 3] packets are dropped. To mitigate against this vulnerability the authors construct a growth limit on the TTD, that dynamically updates according to an adjusted process after a certain upper bound TTD that keeps it from increasing indefinitely and blocking all ICMP [Type: 3, Code: 3]. The upper bound can be calculated based on a business model, positive increment, or attack probability.

Overall the work provides an approximate 25% reduction in legitimate packet loss, a large improvement on the previous work, and addresses a critical vulnerability.

Alex *et al.* [2] proposed firewall fingerprinting techniques that examine probe packet process timing variation under controlled conditions. In their experiment, shown in Figure 2, the authors construct an attack scenario where a compromised host resides behind several different kinds of software and hardware firewalls such that packet processing times (PPT) can be calculated and compared.

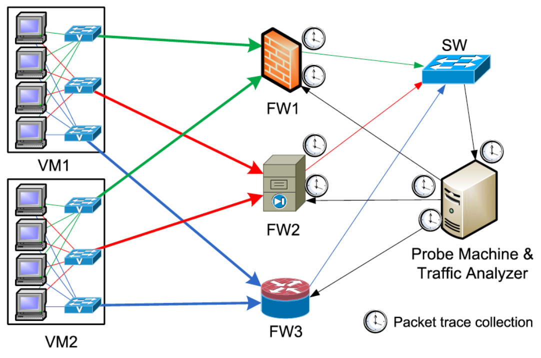

Fig. 2: Testbed Experiment
Source: Adapted from [2, Fig. 1]

The authors specifically attempt to fingerprint the packet classification algorithms, caching, and statefulness of various firewalls. To identify the packet classification algorithm used by a firewall the authors first try and discern if a sequential rule-based search is used. To accomplish this multiple packets are sent to the firewall that are known to match exactly one rule. If the PPT increases linearly then the firewall can be said to likely use a sequential-rule-based classification. However, if the PPT remains constant then some other classification algorithm is used. Performance sensitivity under heavy and light loads can also give an indication of a particular packet classification algorithm. The authors were able to clearly separate all firewalls (two software and one hardware based) on the load variation and average PPT alone. Additionally, the authors use a similar technique of sending multiple probe packets against a single filtering rule to fingerprint protocol specific firewall caching methods (i.e.: rule or flow based) to infer the statefulness of the firewall. Lastly, the benefit of using the PPT fingerprint technique as opposed to sending well-crafted TCP probe packets with various headers is that firewalls may change their TCP flag headers from the default causing false positive or false negative fingerprinting, whereas PPT does not. A major contribution of the work is that the authors demonstrate that should an attacker successfully fingerprint a firewall they can then choose specific attacks to degrade firewall performance. This is important since blind attacks on firewalls are much less effective, as demonstrated by the authors.

Appelt *et al.* [8] highlighted the importance of Web Application Firewalls (WAFs) in preventing SQLi attacks. A WAF inspects the input request supplied to it using rules such as regular expressions and verifies its permissibility. But for these firewalls to remain effective against the growing number of attacks, developers have to first keep themselves updated with those attacks and then customize WAFs in order to prevent them. To make this process of fine-tuning WAFs easy the authors proposed a testing technique that uses machine learning to find holes in WAFs that are responsible for SQLi attacks that are able to pass through WAFs and successfully attack web applications. For this technique, the authors developed SQLi grammar using the information of known SQLi attacks considering Boolean, Union, and Piggy as the three fundamental classifications of SQLi attacks. The authors also implement a Random (RAN) input generation approach with the aim of automatically generating attack payloads. Additionally, this technique uses machine learning to study the attributes of the SQLi attacks that have been already blocked by the firewall. It uses these attributes to generate more test payloads. To prove this machine learning based technique's effectiveness, the authors performed various experiments using this approach on popular WAFs such as ModSecurity. Upon performing the analysis, it was found that the authors' proposed machine learning technique produces numerous test cases that were found to bypass ModSecurity. These tests could conveniently be used by security professionals to customize WAFs to safeguard web applications from the growing number of SQLi attacks.

Mukhtar *et al.* [9] investigated the current solutions to prevent and correct SQLi attacks and evaluate the capability of Modsecurity to protect web applications against SQLi attacks, shown in Figure 3. The authors perform a survey where they talk about different approaches to identify and counteract SQLi attacks that they have researched.

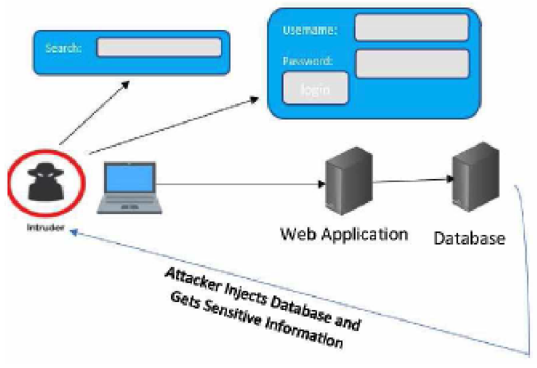

Fig. 3: SQLi Example
Source: Adapted from [9, Fig. 1]

The first approach that the authors discuss for preventing SQLi uses regular expressions and escape strings. However, it still requires large amounts of testing to tune the performance overhead. This disadvantage can be avoided by using hashing, syntax awareness, and signature matching approach's to prevent SQLi. Next, the authors discuss how signature matching has low performance overhead and is easy to implement. However, they note it is incapable of detecting zero-day attacks. Additionally, they discuss how using the SHA2 for hashing will make the web application vulnerable to attacks. Furthermore, the authors elaborate on machine learning techniques that also work well in mitigating SQLi attacks. Unfortunately, the problem with using machine learning for SQLi detection is that only a few datasets are available to train the model. Another technique to counteract SQLi is with the help of Database Embedded Tools. This technique is not only a great solution to mitigate SQLi but also to semantic mismatch problems. The authors argue that this approach has a lot of potential, but it is still a work in progress and needs to be tested in various databases. There is a way to prevent SQLi code from reaching the server that the authors include in their survey. This is by using Client-Side Controls that can be embedded inside different browsers. This technique is simple yet complicated since it requires rework on the browsers themselves. There are several techniques of detecting SQLi that require a significant amount of research before declaring them efficient. Two of those included in the survey are using Source Code Review, Static and Dynamic Analysis, and Blackbox Testing. After conducting the survey, the authors evaluate SQLi attacks against ModSecurity that they think will be a more productive approach to preventing SQLi attacks. In the assessment, they measure the effectiveness of ModSecurity in thwarting SQLi attacks by performing a three phases experiment where an application is first tested utilizing SQLMAP without using ModSecurity and then after installing ModSecurity. Lastly, the application was tested after adding new SQLMAP web application evasion attributes. The results showed that the ModSecurity firewall is efficient in preventing SQLMAP iterations. The authors suggest that the ModSecurity firewall is successful in thwarting SQLi attacks given that the rules of ModSecurity rules are updated regularly.

Salah *et al.* [3] demonstrated the importance of network firewalls in inspecting network traffic and their efficiency in preventing the Distributed Denial of Service (DDoS) attacks. The paper briefly elaborates on the inspection of packets by implementing a rule-based engine that the network firewalls use to examine the network traffic. Conditions that are present in rules are compared with the incoming packets and based on the comparison, specific actions are performed on them. According to the authors, the need for network firewalls has increased over the years and with this need, the attacks on firewalls have increased simultaneously. One very common and critical attack on firewalls is DDoS which puts the firewall implemented security at risk. Consequently, the authors think that an analytical queueing model that examines the performance of firewall would help security professionals and designers to study the execution of rule-based firewalls under different attacks including DDoS. In the proposed queueing model, network packets reach the firewall and queue up for multistage processing. After the packets are queued, they are compared to the rules in the firewall's rule-based engine one by one until it matches a rule. With the queueing model, the authors also suggest two analytical models, one acts as the rule-based engine of firewall, and the second one records firewall's performances after the rules are triggered in the first model. To analyze the models, the authors perform several experiments that involved sending DoS attack flows that target rules at the top and bottom of the ruleset. The authors observed that the firewall's prevention against DDoS attacks is acceptable only for the top ruleset. However, for the rules in the bottom of the ruleset, the firewall's performance is degraded in preventing DDoS attacks. As a result, the authors suggest keeping the size of the firewall's ruleset small. Another possible solution to this could be dynamically rearranging the rules so that the rules at the bottom can be brought to the top preventing attacks like DDoS.

Goss and Jiang [4] proposed a technique to mitigate inside attacks on firewalls. In their work, the authors consider both passive and active attacks on firewall rules. In the passive case, an inside attacker who wishes to learn firewall rules can use the knowledge to construct a precision fingerprint for further attacks without having to resort to other potentially noisy firewall fingerprinting techniques, previously discussed in [2]. Additionally, firewall rules could also be used by a passive inside attacker to infer sensitive relationships within the network and to identify potential targets via whitelist and blacklist rulesets. In comparison, if an active inside attacker were to insert malicious records into the firewall rules then they would essentially have complete control over the network. The authors discuss a naïve approach where firewall rules are simply encrypted using symmetric key cryptography in addition to other approaches like access control based policies. However, while these approaches are satisfactory for addressing outside attackers they do not address inside attackers. This is because inside attackers are presumed to have escalated privileges granting them access control and the ability to decrypt among other privileges.

To address these challenges the authors propose the use of Distributed Oblivious Firewalls, shown in Figure 4, that share a distributed set of information theoretic secure filtering rules. In this scheme, it is of critical importance to understand that no one entity owns, or can evaluate, firewall rules without cooperation from other parities. Additionally, once the scheme is constructed at no point can the firewall simply refer to a local copy of the rules since they are deleted after the initialization step in the scheme, thus no one entity ever holds a "plaintext" copy of them. To achieve this additive secret sharing, for passive inside attackers, or Shamir Secret Sharing, for both active and passive attackers, are used such that n or more nodes in a distributed setting could learn the filtering rules together, while a single node could not.

Therefore, the filtering rules are protected from passive and active inside attackers so long as the attackers do not control a minimal threshold of reconstruction shares. This has the advantage that in the case of a Shamir Secret Sharing approach the firewall would no longer have a single point of failure and could continue operation even if several nodes, not exceeding the minimal reconstruction threshold, were non-operational.

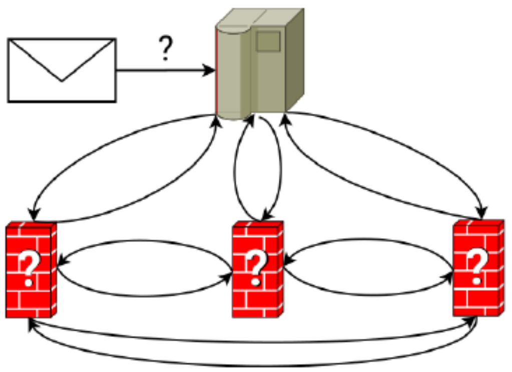

Fig. 4: Distributed Oblivious Firewall
Source: Adapted from [4, Fig. 1]

The secure multi-party computation aspects of the protocol rely heavily on secretly shared Bloom filters used to evaluate firewall rules in a distributed setting. A drawback of the scheme is that since Bloom filters generate a calculably small amount of false positives only blacklist rulesets can be used in the scheme. This is because a falsely dropped legitimate packet is tolerable compared to a falsely forwarded illegitimate packet. The major concern with the scheme, scalability, is considered by the authors who demonstrate that it only adds marginal overhead and can work in a truly decentralized manner such that any node could query the distributed firewall, not just the gateway.

IV. Distributed Oblivious Firewall Model

As part of our work, we model the Distributed Oblivious Firewall scheme as outlined in [4]. Here, the reader is expected to be familiar with Bloom Filters and Shamir Secret Sharing since it is not practical to explain them here in detail. However, the reader is encouraged to refer to [4], [10], [11] for further reading in these areas. In the remainder of this section, we briefly outline the *most important functions* developed from equations and algorithms in [4, Alg. 2 and 3] used in our model. Additionally, we also cover our proposed improvements to the model and future directions of our work.

Our model was developed entirely using Python 3. Our codebase implements a Shamir Secret Sharing scheme in *SSS.py*, Bloom filter in *BF.py*, Distributed Oblivious Firewall scheme in *firewall.py*, helper functions in *lib.py*, a test script in *test.py*, and a Distributed Oblivious Firewall scheme with our proposed changes in *firewall_mod.py* [12].

A. Firewall Functions

In the firewall initialization function, shown in Listing 1, a list of node objects and a blacklist of IPs are received as arguments to the function in line 1. In line 2 a Bloom filter is instantiated using B bits and k independent hash functions. Next, the IPs in the blacklist are hashed into the Bloom filter in lines 3 and 4. Afterward in lines 5-8 each bit in the Bloom filter is secret shared using Shamir Secret Sharing, bounded by a predefined minimum and maximum share reconstruction threshold, namely t and m respectively.

```
1  def firewall_init(self, nodes, blacklist):
2      bf = BF.BloomFilter(self.B)
3      for ip in blacklist:
4          bf.add(ip, self.B, self.q, self.uni_hash)
5      for bit in bf.arr:
6          bit_shares = SSS.gen_shares(self.N,
           ↪   self.m, self.t, bit)
7          for node, share in zip(nodes,
           ↪   bit_shares):
8              node.shares.append(share)
9              node.B = self.B
10             node.q = self.q
11             node.k = self.k
12             node.uni_hash = self.uni_hash
```

Listing 1: Firewall Initialization

Additional pieces of data are copied from the firewall to each node in lines 9-11. Lastly, it is important to note that the Bloom filter is not stored or used after the function completes.

Next, in the firewall evaluation function, shown in Listing 2, a list of node objects and a single IP are received as arguments to the function in line 1. In line 2, the IP is passed to each node, and a list of m shares is recovered.

```
1  def firewall_eval(self, nodes, ip):
2      shares = [node.recover_share(ip) for node in
       ↪   nodes]
3      sample = random.sample(shares, self.t)
4      result = SSS.recover_secret(sample, self.N)
5      return result == self.k
```

Listing 2: Firewall Evaluation

On line 3, a minimum threshold of t shares is randomly sampled from the m recovered shares. Next, the secret is reconstructed and the equality between the secret and number of hash functions k is returned in lines 4 and 5. Our model differs slightly in this regard from [4, Alg. 3], as only one sample is taken instead of multiple. The reason for taking multiple samples has to do with detecting and identifying attackers rather than determining outright correctness of the reconstructed secret. Regardless, the basic functionality remains the same.

In the recover share function, shown in Listing 3, the nodes return their shares for a given input IP, in line 1.

```
1  def recover_share(self, ip):
2      shares = [self.shares[i]
3                for i in lib.hash(ip,  self.B,
                 ↪   self.q, self.uni_hash)]
4      return (self.id, sum([i for _, i in shares]))
```

Listing 3: Firewall Recover Share

In line 2 and 3 the IP is hashed using k hash functions. All the values at the indices of the k digests in the shared

Bloom filter are returned into a list. In line 4, a tuple containing the node identifier and the sum of its shares are returned. The values are summed since Shamir Secret Sharing has an additive property that helps limit the communication complexity (i.e., one tuple is sent rather than *k* tuples).

B. Shamir Secret Sharing Functions

In the Shamir Secret Sharing share generation function, shown in Listing 4, several parameters are used representing the modulus *N*, maximum number of shares *m*, minimum number of shares *t*, and secret value *S* in line 1. In line 2, *t* - 1 random coefficients are chosen to construct the secret polynomial. In lines 3-8 shares of the secret polynomial are constructed and appended to a list that is then returned in line 9.

```
1  def gen_shares(N, m, t, S):
2      C = [random.randrange(0, N) for _ in range(t
       ↪  - 1)]
3      s = []
4      for x in range(1, m + 1):
5          p = 0
6          for t, c in enumerate(C):
7              p += c * x ** (t + 1)
8          s.append((x, (p + S) % N))
9      return s
```

Listing 4: Share Generation

In the Shamir Secret Sharing secret recovery function, shown in Listing 5, a list of *shares* and a modulus *N* are passed as parameters in line 1. Lagrange Polynomial Interpolation is used in lines 2-9, to reconstruct the secret value from a minimal threshold of *t* or more shares. One notable part of the algorithm occurs in line 8 where the modular inverse is needed since the calculations in our model are computed over a finite field.

```
1  def recover_secret(shares, N):
2      s = 0
3      for j, sj in enumerate(shares):
4          xj, yj = sj
5          for k, sk in enumerate(shares):
6              xk, _ = sk
7              if k != j:
8                  yj *= -xk * pow(xj - xk, -1, N)
9          s += yj
10     return s % N
```

Listing 5: Secret Recovery

C. Proposed Changes

A small theoretic issue with the current implementation is that it unnecessarily leaks information about the IPs being queried to the distributed nodes during firewall evaluation. While this is needed in the case of a truly distributed topology, where any node can act as a gateway, it is not needed in the case where a constrained topology is employed (i.e.: a master node overseeing several distributed nodes).

We propose two minor changes to the model to alleviate the information leakage and improve query privacy in the constrained topology setting. We propose that during the initialization phase, rather than hashing plaintext IPs into the Bloom filter, a keyed hash of the IP is instead hashed into the Bloom filter. In this case, the master node keeps the key used during the firewall initialization and thus is the only node that can request a valid query to evaluate an IP. Correspondingly, during the firewall evaluation when the master node wishes to evaluate an IP it must hash the IP in conjunction with its key before sending the digest to the distributed nodes for evaluation. These changes are shown in lines 5 and 18 of Listing 6 within their respective functions.

```
1  def firewall_init(self, nodes, blacklist):
2      bf = BF.BloomFilter(self.B)
3      for ip in blacklist:
4          # ==MODIFIED==
5          bf.add(lib.sha256(ip+self.key), self.B,
           ↪  self.q, self.uni_hash)
6          # ============
7      for bit in bf.arr:
8          bit_shares = SSS.gen_shares(self.N,
           ↪  self.m, self.t, bit)
9          for node, share in zip(nodes,
           ↪  bit_shares):
10             node.shares.append(share)
11             node.B = self.B
12             node.q = self.q
13             node.k = self.k
14             node.uni_hash = self.uni_hash
15
16 def firewall_eval(self, nodes, ip):
17     # ==MODIFIED==
18     shares =
       ↪  [node.recover_share(lib.sha256(ip+self.key))
       ↪  for node in nodes]
19     # ============
20     sample = random.sample(shares, self.t)
21     result = SSS.recover_secret(sample, self.N)
22     return result == self.k
```

Listing 6: Firewall Proposed Changes

Under this modified scheme the distributed nodes can no longer learn the IP being sent to them since they only receive a keyed hash from the master node that they cannot feasibly reverse. Therefore, the privacy of queries is substantially improved. However, a limitation of this proposal is that a node could theoretically match two received digests and discern that they are of the same plaintext or IP even if they don't know what the IP is or what its evaluation outcome is (i.e., blocked or forwarded).

D. Remarks

One of the most difficult parts of implementing this scheme, as noted by the authors, is selecting the appropriate parameters for the Bloom filter construction such that false positives are not excessively generated. We tested our model against 1,000,000 randomly selected IPs in the range 192.0.0.0 to 192.255.255.255. Additionally, we randomly selected ten random IPs in the stated range that we placed on our blacklist. Furthermore, at every 10,000th test iteration, we randomly selected a blacklisted IP for testing to ensure our model could correctly block it. However, we were primarily interested in false positives (i.e., falsely blocked IPs). We ran the experiment several times and found that by increasing the number of hash functions k and the size B of bits in the Bloom filter we could generate smaller and smaller error rates. With a k value of ten and a B value of 1,000,000, our model generated eleven false positives. With a k value of 20 and a B value of 4,000,000 our model instead generated three false positives. Depending on the application these false positives may or may not be acceptable. However, it is important to consider that this limitation is somewhat offset by the protection the scheme provides against inside attackers. Additionally, since the IPs on the blacklist are merely strings, we were able to extend our blacklist to include filtering rules with source and destination IP and port as well as IP protocol. We observed identical findings in this case and when testing our proposed changes.

As part of our future work, we are considering adding our to proposal where the master node sends both real and fake queries to different groups such that a node could only guess with some probability that it received a real query. Additionally, now that we have successfully developed and tested our model for correctness we are looking to fine-tune our codebase and test our model with larger blacklists, Bloom filters, and hash functions. Afterward our next step would be to implement the scheme in a real-world firewall under controlled conditions.

V. Conclusion

In our work, we introduce firewalls and how they are susceptible to various kinds of attacks. We survey several attacks in depth and outline their associated mitigation techniques. Additionally, we develop a working Distributed Oblivious Firewall model and propose a simple improvement to help improve query privacy. Lastly, we outline our future research plans and work in this area going forward.

References

[1] K. Nagendran, S. Balaji, B. A. Raj, P. Chanthrika, and R. Amirthaa, "Web application firewall evasion techniques," in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2020, pp. 194–199.

[2] A. X. Liu, A. R. Khakpour, J. W. Hulst, Z. Ge, D. Pei, and J. Wang, "Firewall fingerprinting and denial of firewalling attacks," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1699–1712, 2017.

[3] K. Salah, K. Elbadawi, and R. Boutaba, "Performance modeling and analysis of network firewalls," *IEEE Transactions on Network and Service Management*, vol. 9, no. 1, pp. 12–21, 2012.

[4] K. Goss and W. Jiang, "Distributing and obfuscating firewalls via oblivious bloom filter evaluation," *CoRR*, vol. abs/1810.01571, 2018. [Online]. Available: http://arxiv.org/abs/1810.01571

[5] D. K. Banwal, S. Kumar, and I. S. Rawat, "Firewall: Software and hardware implementations," 2013.

[6] Z. Trabelsi, S. Zeidan, and K. Hayawi, "Denial of firewalling attacks (dof): The case study of the emerging blacknurse attack," *IEEE Access*, vol. 7, pp. 61 596–61 609, 2019.

[7] K. Hayawi, Z. Trabelsi, S. Zeidan, and M. M. Masud, "Thwarting icmp low-rate attacks against firewalls while minimizing legitimate traffic loss," *IEEE Access*, vol. 8, pp. 78 029–78 043, 2020.

[8] D. Appelt, C. D. Nguyen, and L. Briand, "Behind an application firewall, are we safe from sql injection attacks?" in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 2015, pp. 1–10.

[9] B. I. Mukhtar and M. A. Azer, "Evaluating the modsecurity web application firewall against sql injection attacks," in *2020 15th International Conference on Computer Engineering and Systems (ICCES)*, 2020, pp. 1–6.

[10] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, p. 422–426, jul 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[12] G. Higgins and V. Modi, "Distributed oblivious firewall model," 4 2022. [Online]. Available: https://github.com/griffinhiggins/DOF