Smart Cooperative Firewalls

An aid to a safer and secure cyber world

Thomas Graves
Computer Science
Truman State University
Kirksville Missouri USA
tcg6531@truman.edu

Chetan Jaiswal
Computer Science
Truman State University
Kirksville, Missouri, USA
cjaiswal@truman.edu

***Abstract* – A Firewall is a necessity in the cyber world today. Cyber-attacks have become a guarantee for any large business or any network containing useful information. Firewalls can be very effective when intelligently designed and run for a single machine; however, they have an issue when they are part of an interconnected firewall network. Administrators are responsible for updating or changing each firewall individually, which has been shown to lead to many errors. In this paper we propose a solution: smart cooperative firewalls. Firewalls that can talk to each other, don't need to be manually updated, and therefore have faster and more accurate attack response procedures. Our proposal, named DSN (Distributed Secured Network), turns multiple, but separate, firewalls into a dynamic network of firewalls all working together to combat cyber-attacks.**

***Keywords* – Firewall; Intrusion Detection System; Security; Network Security; Cyber-Attack**

I. INTRODUCTION

Firewalls are the first line of defense and the most widely used tool in cyber security. They can be very powerful if applied correctly, but they are limited in their scope. Packets are analyzed as they come in and compared to a set of priorities. The limitation is that the priorities can only read and act on the header information of the packet: this includes data like the type, source IP address and port number, and the destination IP address and port number. Therefore the ability to act on incoming packets is quick but may miss attacks that are based on packet content. For example, a cross-site scripting attack could come from any IP; a firewall cannot predict where this attack will come from, so a new tool is needed to analyze packet content.

Intrusion Detection Systems (IDS) serve exactly that purpose. IDS' can be very powerful due to their ability to look inside packets at all the information. Consider the scenario: an IDS could look at a packet attempting a cross-site scripting and notice the exact text attempting the attack and compare that to a known cross-site scripting signature. A limitation for an IDS is that it can be slower. When taking into account its ability to analyze more data, a slower analysis time is certainly reasonable.

Traditional and current firewall systems are failing. The increase in both the number of attacks and the complexity of attacks has put the conventional system out of date. The Hiscox Cyber Readiness Report 2017 [1] interviewed 3000 companies across the U.S., U.K., and Germany. According to the report 53% of companies were poorly prepared and only 30% were very prepared for cyber-attacks. This lack of preperation is concerning as cyber-attacks have such an effect on both the economy and people's personal lives. The total cost of cybercrime was $450 billion, and 2 billion personal records were stolen in 2016 [2]. The most recent WannaCry ransomware attack in May of 2017 infected more than 230,000 computers in 150 countries. It encrypted all the files on a system and demanded a $300 payment to unencrypt them. This attack was especially harmful in Europe where it infected multiple hospitals' systems [3]. Not only as computer users do we have to be concerned about our own security, but the security of companies and organizations that we rely.

It has been suggested that through the year 2020, 99% of all firewall breaches will be due to firewall misconfiguration, assuming the same systems are implemented [4]. Administrators not only have to stay up to date with all the potential cyber threats, but they have to manually update their firewalls with rules that use a syntax that is easy to confuse. And then they have to apply that rule individually to each machine on the network. This process can create issues as it creates a window of time in which certain machines are up to date and others are not. It only takes one machine, one entry point, being out of date to potentially compromise the network.

Our goal and the purpose of this paper are to combine firewalls and IDS' dynamically. Additionally apply that system across a network, creating a Distributed Secured Network (DSN). The DSN will use an IDS to detect an attack, take that information, analyze it and then feed it to the firewall so that the

firewall can deal with it. It will send that information to each firewall on the network creating the exact same rules across the network. This setup creates a system in which all the firewalls are updated automatically and will always remain consistent, eliminating that vulnerability. Referring to the same example as before, the IDS will detect a cross-cite scripting attack, it will take that information, included the source of the attack and give that to the firewall. Now the firewalls (the local firewall and the others on the network) could block the attack using the source of the attack.

Throughout the paper, the terms machine, computer, and entry point are all used interchangeable, meaning the computers at the entry points of a cloud network, data center, etc.

The rest of the paper is organized as follows. Section II covers the key philosophies and features for the DSN. Section III introduces the design of the DSN prototype and its evaluation. Section IV considers multiple scenarios. Section V presents similar works and Section VI is the conclusion and future works.

II. Key Features and Philosophies

Before we cover the actual design of the prototype there are a couple of key ideas that drove the design and were inspiration for the whole prototype. While other tools applying a Distributed Intrusion Detection System are similar in some ways, the following features make our tool unique because they differ in either ideology or design.

The first feature is for the tool to be totally decentralized. While other tools have been proposed for this problem they usually have a centralized machine that does the data analysis and rule management [5]. One thing our system is designed to do is to keep running even if any of the machines go down. This stops the possibility of the whole system crashing because a central unit crashed. Having a centralized data analyzer of tool manager does have its advantages. It could potentially increase speed and ease to work with but the potential for it to compromise the security was not worth it. The tool we designed shares information directly to other machines instead of with a central one and the analysis is done on each machine. While no direct work has been done comparing these two methodologies, our system would potentially only compromise on speed a small amount if any. However that compromise in speed for a single machine is countered by a faster overall network attacks response delay. Our system is also designed to have minimal human interaction due to its automatic and dynamic nature.

The second feature is that the IDS do not deal with the attack directly but feeds information to the firewall which then deals with the attack. Instead of implementing an Intrusion Prevention System, which is similar to an IDS except that it stops the acts instead of simply alerting the system, we wanted a system that automatically reads those alerts from the IDS and gives the firewall information to act on. This feature has a couple of desired outcomes. It is easier to manage, since all the rules actually affecting incoming traffic are from the firewall. Also it would potentially make the whole system more efficient as the IDS will not have to be checked for any traffic that has already been added to the firewall.

The final feature is the implementation of a time to live (TTL) for the rules. A TTL is the process of applying a rule only for a certain amount of time and then removing it. Without this the firewall rules could become cluttered, slow, and also hurt management ability. A time to live is also implemented because often rules only need to be applied once. If someone attempts a cross-site scripting attack and is blocked for a short amount of time, they probably will move to the next target instead of waiting an unknown amount of time. Attackers sometimes just probe a network to see if it vulnerable. If they probe fails, they move on, maybe to a different type of probe or to a completely different target. Either way there is no need to have that original rule in place anymore.

III. Prototype Design and Evaluation

The design of the DSN is separated into 4 threads (or processes), each running continuously on every machine in the network. Along with those threads, each machine is running an IDS, for our prototype we used snort [6]; and a firewall. We used the Linux firewall: iptables [7]. Our prototype is written in Java.

The prototype was built using a Toshiba Satellite model laptop. The CPU is an Intel i7 with 4 cores and running at 2.50 GHz. The computer also has 8 GB of RAM. For software; the operating system is Ubuntu 16.04 LTS. In order to create the network the application VirtualBox was used. Four virtual machines were created and designated to all run on the same network. Each machine is also running Ubuntu 16.04 LTS and was assigned 1 GB of RAM from the host computer.

Figure 1 shows the layout of a basic cloud network. Within the cloud there is a network of machines. In order to get access to this network, an IP packet must pass through one of the multiple entry points. Each of these entry points has a firewall associated with it. Traditionally these firewalls would act on their own, but the DSN connects them all, as shown in the image. Now each of these entry point firewalls has each of the four threads (only listed at entry point 1 to avoid clutter). Table 1 is a basic diagram of what happens at each machine and thread.

The four threads are the Client, Server, TTL, and Rule Updater threads. Each of these threads is explained in more detail, followed by a simple algorithm demonstrating its activity.

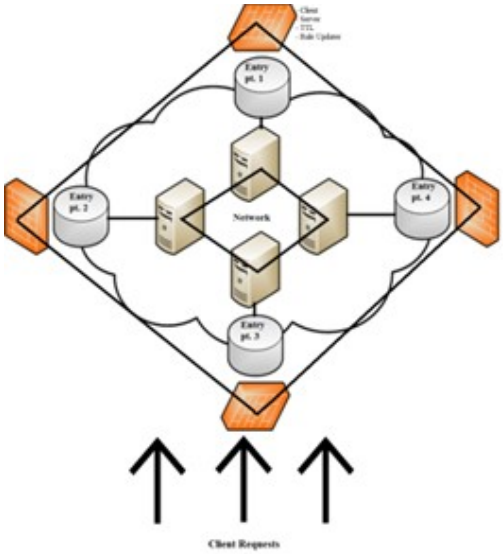

Figure 1. DSN on a network

Table 1. DSN processes

ALL MACHINES	
• Running an IDS [snort] • Running a Firewall [iptables]	
CLIENTS	**SERVERS**
• Read Data from IDS logs •Send Data to all servers	•Read data form all clients • Convert data to firewall rule • apply rule
TTL • Repeatedly check time to live of rules **Rule Updater** • Updated newly added rules	

A. Client

The Client is a program continuously running at every entry point. It is responsible for detecting an alert sent by the IDS. This is done first of all by ensuring that IDS is alerting and logging that alert to a known directory. The Client must continuously watch this directory for any changes. If a change is detected then the Client takes action. First, it analyzes the log of the alert picking out the critical information like the type, the source IP address and port, the destination IP address and port, and an ID associated with that IDS rule. Then multiple different sockets are created for each machine on the network and that information is sent to them all. Following is the pseudo code for the client:

```
While (True)
    Check snort log directory
    If new snort log detected
        Convert log to apply a rule to own firewall
        Send rule update to all other machines to do
        the same
```

B. Server

The Server is a program running on each machine. It has two responsibilities, listening for any communication from any of the Clients and acting on that information. A socket is created to listen to incoming data and that data is stored. When the Server receives data that indicates that an attack was attempted on another machine in the network, now, this machine should prepare for that attack. The information that was received will include the data needed to act, generally blocking the source at the firewall. For our tool this involved using the Java Runtime class in order to write an iptable rule at the terminal.

Finally, data is written to a "Rulebook" file. This file is used as a list of active rules for the DSN. Each "Rulebook" file will be exactly the same so this can be used as a tool to analyze which rules are active on the system. The file was written to insure readability and it also serves a couple of functions for the DSN system. It makes sure the rules are not duplicated or need to be renewed and it applies the time to live feature (described more later). The file has a format as follows:

```
"Rule that was just applied | Current timestamp |
Time to live for the rule"
```

Following is the pseudo code for the server:

```
While (true)
    Listen for incoming message
    If new message detected
        Apply data in message to create new firewall
        rule
        Write data to Rulebook file
```

C. TTL

The TTL (Time to Live) thread is used implement the feature discussed in section II. This thread allows the newly created rules to only be active for a given amount of time. The TTL is its own thread and not part of Server because it needs to continuously check the "Rulebook" file. This would cause the Server class to block and could mean it loses incoming data, and therefore not prepared for attacks. The "Rulebook" file contains all the rules, the timestamp when each rule was created and the TTL for how long the rule should be active. The TTL continuously checks if any of the rules have expired by applying this simple formula to every rule:

$$Current\ time - timestamp > ttl.$$

If this equation returns true then the rules must be removed. Then the rule will no longer exist in iptables or in the "Rulebook" file. For every rule in which this formula holds true, the rule that the Server

previously added needs to be removed. For iptables this simply means using the Runtime class again but changing the "-I" to a "-D" and everything else remaining the same. This changes it from "input" to "delete". In the "Rulebook" file that rule must be found and deleted.

```
While (true)
   Check rule book file
   If current time – timestamp > ttl for any rule
      Remove the rule from iptables and Rulebook
```

D. Rule Updater

The Rule Updater thread is running on each machine. It has a single job to make sure that all the rules across the machines are the same. As discussed in the introduction, administrator errors and firewall inconsistences are two major factors in firewall vulnerabilities. This thread eliminates those issues. Any new rule created by the DSN will be applied to every machine automatically, so there is no worry of inconsistency there. However, any new rule created by an administrator needs to be applied to all the machines. Of course, an administrator could apply the rule manually to each machine, but that would increase the likelihood of an error. The DSN only requires updates on one machine, and it will upload it to all the machines. IDS or firewall rules can be updated, and, similarly to the Client thread, those can be monitored for change, and then simply the changes can be sent to the other machines. Following is the pseudo code for the rule updater:

```
While (true)
   If a new rule has been added to this machine
      Update all other machines
```

Using this prototype, the DSN is robust, lightweight, and adaptable. The DSN will continue to function even if some machines go down. Since the analysis is done at every point and there is no central part, the rest of the machines will continue to check for breaches and share that with operating machines. The DSN is lightweight because it largely uses tools that are already in place: snort and iptables. The rest of the new code is sharing data between the machines, converting certain information to create rules, and reading through files to check for changes. None of these are demanding tasks. This system functions buy running the exact same code on each machine. Then the code just has to be edited once and uploaded to each machine. The DSN is not a complex system to run on a system. There is no worry about what has to be uploaded where. Every machine performs every function.

IV. Attack Scenarios

Throughout the paper the scenario of cross-site scripting has been used a few times to assist in the explanation. Here we present a few more scenarios to demonstrate how the DSN will work.

An SQL injection is a simple attack that will illustrate the basic way the DSN functions. An SQL injection attack works by attempting to force SQL code into a form so as to disrupt or gain access to information that the attacker should not have. The IDS Snort will detect the attack by comparing it to certain known signatures. The DSN will then create a firewall iptable rule to block that IP address. It will then record that rule in the "Rulebook" file. The DSN will share that information to all other active machines on the network. Those machines will also create rules and update their own "Rulebook" files. Every machine will have its TTL thread running and go and remove that rule after a predetermined about of time. This process can be repeated for any simple rule that an IDS can alert on.

Now consider that an administrator wants to block that SQL injection permanently. The administrator can go and write a permanent rule at either the firewall or IDS. The IDS thread will update all machines.

A distributed denial of service (DDOS) attack is one of the most common attacks facing both larger and smaller organizations today. There are few advantages to using our tool to combat this attack. First and most simply it should be able to detect and stop it efficiently. As long as there is an appropriate IDS rule written to detect a DDOS attack, the DSN will alert for that attack, creating a new firewall rule to stop it. There is a good chance that since the attack is now failing, it will just halt. The TTL will remove that rule from the firewall rule set to avoid congestion and improve readability. If the attack were to continue then the DSN would trigger again, blocking that attack, but this time it will assign a longer time to live for the rule and so on until the attack needs to be blocked permanently.

The DSN also be applies these rules and TTL's across its own network, so if the attacker tried to move to a new entry point they would face the same issue. The attackers are no longer able to try and race the administrator's ability to update all the firewalls.

Now suppose that DDOS attack was more successful and crashed that entry point. While this is obviously not a desired outcome, the DSN will still be able to function as normal. It will still alert that a DDOS attack was occurring and inform the other entry points. Now the rest of the machines will be protected from that kind of attack.

Consider instead that the computers on our network have become part of a botnet and are being used in a DDOS attack. Again, an IDS like snort can not only rule check for incoming packets but for outgoing packets as well. Therefore the DSN can work here too. Not only will it stop that one machine from proceeding with the DDOS, but it will stop

them all. It will also alert you as being part of a botnet then which can be fixed.

The DSN is adaptable as well. Features can easily be added. An attack scenario that has multiple sources or occurs over a length of time has traditionally been difficult for security systems to detect. The solution for these kinds of attacks is to log information. Then to periodically check those logs to detect attacks. While the DSN is not currently structured to detect these kinds of attacks, it can be modified to do so. Every entry point can log information and share that information with the other entry points. That way the system remains decentralized, but still combats this kind of attack. Not only if attacks come from different sources can they be detected, but even if they have different destination. If part of an attack comes to one entry point and the other part of the attack comes to another entry point the system would then be able to detect it. All entry points will have all information and then will be able to respond quickly and effectively.

Finally a simple test was run using the DSN and comparing it to a more conventional firewall and IDS system. The test was to check how long it took our 4 systems to deny an attack. First this was done by having the system just run on each machine individually, without any communication. Then we wanted to see how much the communication between machines would speed up the process. The times where calculated when snort sent an alert, which has a time associated with it, and after the ip address was blocked, which was hardcoded into the program. These times were then subtracted from one another to get the length of time it took to block the ip address. The Reaction times are listed below in Table 2.

Table 2. Conventional Firewall Vs. DSN: Reaction Time

Firewall Reaction Times		
Times	Traditional System	DSN
For each computer to block	.401 seconds	Not applicable
For whole Network to block	1.604 seconds	.502 seconds

As shown in table 2, the traditional system is much slower blocking the whole network. This is because each time the entirety of the blocking process is run, both the IDS and firewall, for each machine. For the DSN the code really only has to run once, then that information is shared and each machine has to run much less intensive code to block the IP address. The traditional systems time to block at one computer is similar to the DSN's ability to block at the whole network level.

V. Similar Works

In this section we will present the review of similar works.

There are multiple other works that try to address similar problems. Others have implemented a distributed firewall system or IDS. These are similar in the reasoning: such as firewalls inability to act on certain kinds of information. However, they differ in approach or features in their systems, one example being the use of a centralized data analyzer.

The work presented in "Implementing a Distributed Firewall" [8] has almost identical motivation as our work. The authors saw the issues with current firewall systems. They proposed a solution is which a security policy was defined centrally and then that policy was enforced at each endpoint. While this does address all the problems stated by the authors, it does introduce some potential problems as well. We wanted to create a robust system that was decentralized, so as to avoid the possibility of a central point of failure. The DSN's ability to be dynamic is important to us as well. Rules are created for the network as needed, then removed when not needed.

Another work presented in [5] is a Distributed Intrusion Detection prototype. The authors had similar motivation and again they differ in the idea of centralization. They were driven by the need to aggregate information and use a central expert system. Their system was designed to work against a doorknob attack, in which a user tries only a few username/password combinations but on many different machines, making it hard to track since the source changes. A central expert system creates an opportunity for an attack. An attacker does not need to try and defeat the whole network, just that one system.

"A summary of the Autonomic Distributed Firewalls (ADF) Project" [9] has similar problem statements and differs in centralization. As stated in section 2.1 in the paper "The ADF architecture is based upon centralized policy management and distributed enforcement". We felt that a centralized policy manager had potential security failure. If that one manager goes offline, the whole system does. A major focus of this work is providing protection on both sides of the network. The authors' state that it is not simply bad guys outside the network trying to get in, but people already inside the network doing damage; and these people have traditionally not been secured against. This was not a focus for our work but the proper use of an IDS can detect outgoing attacks and the DSN can then respond appropriately.

Other works were not concerned about the creation of distributed systems, but rather the testing of firewall systems.

The work on "Conflict Classification and Analysis of Distributed Firewall Policies" [10] is used to simplify firewall management. The authors know firewall policies can have many anomalies, such as multiple rules blocking the same packet, or separate firewalls having different policies. Their research is to design a tool that would make addition, deletion, and modification of rules easier. Our research might very well benefit from this. However our tool in the way it is designed does naturally avoid some anomalies. Multiple rules should not be blocking the same packet since rules aren't created until needed and since the machines share the rules, they should always be consistent. However this does not mean that the rule set is necessarily optimized.

The study "Firewall Security: Policies, Testing and Performance Evaluation" [11] created a system with multiple different security levels and analyzed the idea that more security means less performance. They found this to not always be true. This method would be interesting to test with our tool, and also compare it to other systems using this method to see how it compares.

VI. Conclusion/Future Work

The DSN tool is a prototype, but has potential to solve many security problems currently faced by organizations running large or small networks. This tool has the potential to ease workloads of administrators, decreasing potential for errors, and has the potential to increase security. The DSN gives every machine on the network, except the one where the initial attack happens, foresight. They already know an attack may be coming. If an attack crashes one machine, all the others will know the attack is coming and drop it before it gets there. Additionally the DSN tool will continue to work after any single machine crashes.

Future work requires extensive testing of this tool, and optimization of the code. The DSN has shown to be effective on our virtual network setup, but we would like to test it on a real network against real attacks and compare the results to systems in use today. Other research can also be applied to truly explore the security and efficiency of the DSN. [10] [11].

Having an interconnected system with many machines has become a common practice for many organizations. All of these networks need security. Unfortunately current security systems cannot insure safety. They have too many flaws. Our goal is to create a polished, lightweight tool that is used to create and protect a safer cyber-world.

VII. Bibliography

[1] "The Hiscox Cyber Readiness Report 2017," London, 2017.

[2] L. Graham, "Cybercrime costs the global economy $450 billion: CEO," cnbc, 7 February 2017. [Online]. Available: https://www.cnbc.com/2017/02/07/cybercrime-costs-the-global-economy-450-billion-ceo.html.

[3] B. Chappell, "WannaCry Ransomware: What We Know Monday," 15 May 2017. [Online]. Available: http://www.npr.org/sections/thetwo-way/2017/05/15/528451534/wannacry-ransomware-what-we-know-monday.

[4] J. DiPietro, "To Err Is Human; To Automate, Divine," InfoSecurity, 14 April 2016. [Online]. Available: https://www.infosecurity-magazine.com/opinions/to-err-is-human-to-automate-divine/.

[5] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, H. L. Todd, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal and D. Mansur, "DIDS (Distributed Intrusion Detection System) - Motivation,".

[6] "Snort," Cisco, [Online]. Available: https://www.snort.org/.

[7] "IPTABLES," 2015. [Online]. Available: http://ipset.netfilter.org/iptables.man.html.

[8] S. Ioannidis, A. D. Keromytis, S. M. Bellovin and J. M. Smith, "Implementing a Distributed Firewall".

[9] L. M. Meredith, "A summary of the autonomic distributed firewalls (ADF) project.," in *DARPA Information Survivability Conference and Exposition*, 2003.

[10] E. Al-Shaer, H. Hamed, R. Boutaba and M. Hasan, "Conflict Classification and Analysis of Distributed Firewall Policies," in *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, 2005.

[11] M. R. Lyu and L. K. Y. Lau, "Firewall Security: Policies, Testing and Performance Evaluation".