IJDCST @ Sep-Oct, Issue- V-2, I-8, SW-22
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)

Dynamic Security Applications for Detecting DDOS Attacks

Subhani Sayyed[1], Smt.SK.M.Almas [2]

[1] **M.Tech(CSE), Vasireddy Venkatadri Institute Of Technology,Guntur, A.P., India.**

[2]**Asst. Professor, Vasireddy Venkatadri Institute Of Technology,Guntur, A.P., India.**

Abstract: Distributed denial of service (DDoS) attack works to shut down a particular victim web server with packet flooding. DDoS attacks evolved from relatively humble megabit beginnings in 2000 to the largest recent DDoS attacks breaking the 100 Gb/s barrier, for which the majority of ISPs (Internet Service Provider) today lack an appropriate infrastructure to mitigate them. The sudden increase in traffic can cause the server to offer degraded performance. My Doom devastation on micro soft, wiki leaks encounter with DDoS barrages is some examples to highlight the impact. And other major Internet players like Amazon, CNN, and Yahoo are no exception. Early discovery of these attacks, although challenging, is necessary to protect victim server's network infrastructure resources. Previous intrusion prevention systems like FireCol although efficient in thwarting DDoS, its architecture is based on ISP collaboration and virtual protection rings. We propose to use an IPS rules(Snort rules) driven DDoS detection approach that checks various parts of a data packet and not just the header. This enables the detection system to eliminate other forms DoS attacks such as Slow Read DoS attack. Its effectiveness and low overhead, as well as its support for incremental deployment in real networks is demonstrated

I. INTRODUCTION

DDoS attacks are mainly used for flooding a particular victim with massive traffic and paralyzing its services [4]. Recent works aim at countering DDoS attacks by fighting the underlying vector, which is usually the use of botnet. A botnet is a large network of compromised machines (bots) controlled by one entity (the master). The master can launch synchronized attacks, such as DDoS, by sending orders to the bots via a Command & Control channel [2][3] . Unfortunately, detecting a botnet is hard, and efficient solutions require scanning entities to participate actively in the botnet itself unlike entities scanning from a safe distance. [6] A single intrusion prevention system (IPS) or intrusion detection system (IDS) can hardly detect such DDoS attacks, unless they are located very close to the victim. However, even in that latter case, the IDS/IPS may crash because it needs to deal with an overwhelming volume of packets (some flooding attacks reach 10–100 GB/s). In addition, allowing such huge traffic to transit through the Internet and only detect/block it at the host IDS/IPS may severely strain [5][7] Internet resources. So a collaborated system is required that can empower the single host based detection and blocking procedures for an efficient prevention of DDoS.

To overcome such problems, a new collaborative system called FireCol was proposed that detects flooding DDoS attacks as far as possible from the victim host and as close as possible to the attack source(s) at the Internet service provider (ISP) level. [3][6] FireCol relies on a distributed architecture composed of multiple ISPs forming overlay networks of protection rings around subscribed customers. The virtual rings use horizontal communication when the degree of a potential attack is high. [2] In this way, the threat is measured based on the overall traffic bandwidth directed to the customer compared to the maximum bandwidth it supports. FireCol Components

- Packet Processor
- Metrics Manager
- Selection Manager
- Score Manager

- Collaboration Manager

FireCol architecture uses the following algorithms: Packet rate computation using rule frequencies(collaboration manager) and Mitigation Shields Deployment. In addition to detecting flooding DDoS attacks, FireCol also helps in detecting other flooding scenarios, such as flash crowds, and other botnet-based DDoS attacks thus offering a better performance. [14] But, FireCol's defense procedures requires different ISP's collaboration to form virtual protection rings which has real time implementation issues involving total revamp of the architecture. FireCol's defense procedures (virtual protection rings notion) are not based on IPS rule structures (Snort Rules).

In this paper, the proposed system extending FireCol to support different IPS rule structures will help FireCol thwart other forms of DoS attacks especially the latest entrant Slow Read DoS attack. Proposed system was Snort's detection system which is based on rules. Like viruses, most intruder activity has some sort of signature. Information about these signatures is used to create Snort rules. These rules in turn are based on intruder signatures. Snort rules can be used to check various parts of a data packet not just the header scanning adapted by prior approaches. A rule may be used to generate an alert message, log a message, or, in terms of Snort, pass the data packet, i.e., drop it silently. Thus enabling a detection system eliminating other forms DoS attacks such as Slow Read DoS attack. Snort Based DoS detection system can be a real time efficient and feasible implementation that can counter varying DoS attack forms.

II. RELATED WORK

High bandwidth DDoS attacks consume more resources with ISP level in DDOs attacks to graceful degradation of network and being undetectable [12][13]. Most number of detection schemes was proposed for current requirement to detection of DDoS attacks. We propose earlier technique i.e. false alarm rate by varying tolerance factors in real time [11]. In this technique we describe the simulation results using some NS-2 simulations techniques present in networks. This technique main advantage is that variable rate attack detection and minimum false alarms. But False alarms have significant results in detection of DDOS attacks [12]. We introduce the network under provisioning in cloud infrastructure for detecting and avoiding new form of DDOS attacks. The above comparison techniques are worked for detection of DDOS attacks. The primary goal of an attack is to deny in Victim's access in particular resources. We provide the framework detecting the attack and dropping the snooped attacks. [13] It will forge the attack in IP packet but we cannot control the hop count in that attack. This technique can be reduced by identifying the attackers in learning state. Finally we describe the scalable solution for detection for DDOS attacks [14]. It is performed as close to attack sources as possible, providing a protection to subscribed customers and saving valuable network resources. Experiments showed good performance and robustness of *FireCol* and highlighted good practices for its configuration. But FireCol was designed in single IPS Rule structure. In this paper we introduce the SNORT rule structure for original source code is available to anyone at no change. Snort Based DoS detection system can be a real time efficient and feasible implementation that can counter varying DoS attack forms.

III. BACKGROUND

Intrusion detection is a set of techniques and methods that are used to detect suspicious [2][3] activity both at the network and host level. Usually an intrusion detection system captures data from the network and applies its rules to that data or detects anomalies in it. Snort is primarily a rule-based IDS, however input plug-ins are present to detect anomalies in protocol headers. Snort uses rules stored in text files that can be modified by a text editor. Rules are grouped in categories. [6][8] Rules belonging to each category are stored in separate files. Snort reads these rules at the start-up time and builds internal data structures or chains to apply these

rules to captured data. [4] Finding signatures and using them in rules is a tricky job, since the more rules use, the more processing power is required to process captured data in real time. [2] It is important to implement as many signatures as it can using as few rules as possible. Snort comes with a rich set of pre-defined rules to detect intrusion activity and it is free to add own rules at will. To avoid false alarms, built-in rules can also remove.

IV. PROPOSED SYSTEM

SNORT is one of the most popular NIDS. SNORT is Open Source, which means that the original program source code is available to anyone at no charge, and this has allowed many people to contribute to and analyze the programs construction. SNORT uses the most common open-source license known as the GNU General Public License. Snort is logically divided into multiple components. These components work together to detect particular attacks and to generate output in a required format from the detection system. Snort's architecture consists of four basic components:

■ The sniffer

■ The preprocessor

■ The detection engine

■ The output

Packet Sniffer

A packet sniffer is a device (either hardware or software) used to tap into networks. It works in a similar fashion to a telephone wiretap, but it's used for data networks instead of voice networks. A network sniffer allows an application or a hardware device to eavesdrop on data network traffic. In the case of the Internet, this usually consists of IP traffic, but in local LANs and legacy networks, it can be other protocol suites, such as IPX and AppleTalk traffic. Packet sniffers have various uses:

■ Network analysis and troubleshooting

■ Performance analysis and benchmarking

■ Eaves dropping for clear-text passwords and other interesting tidbits of data.

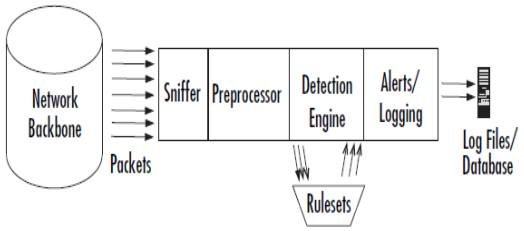

Fig 1: Snort Architecture

Preprocessor

A preprocessor takes the raw packets and checks them against certain plug-ins (like an RPC plug-in, an HTTP plug-in, and a port scanner plug-in).These plug-ins check for a certain type of behavior from the packet. Once the packet is determined to have a particular type of "behavior," it is then sent to the detection engine. Snort supports many kinds of preprocessors and their attendant plug-ins, covering many commonly used protocols as well as larger-view protocol issues such as IP fragmentation handling, port scanning and flow control, and deep inspection of richly featured protocols.

Step 1: Extend the Original Rule set{R}

Step 2: Initially Extended Rule Set E=ϕ, then E= Insert (R_i, E).

Step 3: For all Rule structure E_r from E

Step 4: Calculating the each matching rules in Original Rule Set Matching Rules i.e. Mi = Mx∪ Mi; where Mi is super rule set and Mx is sub rule set.

Step 5: We repeat the above 2,3,4 steps for each client present in network.

Algorithm1: Detection of rule structure algorithm.

As shown in the above algorithm1, detection with matching rule structure working procedure as follows. Initially we are taking original rule set R={R1,R2,…….Ri} as input. Each rule set associated with match list with index provided by our original rule set. Then extended rule set scans each rule E_i in E and check the matching relations between original rule set structures with generated rule set. If matching is done in this relation then we are adding that client into network. If any rule structures are not matching with original rule set then we are assigning that particular client may be act as attacker.

Detection Engine

Once packets have been handled by all enabled preprocessors, they are handed off to the detection engine. The detection engine is the meat of the signature-based IDS in Snort. The detection engine takes the data that comes from the preprocessor and its plug-ins, and that data is checked through a set of rules. If the rules match the data in the packet, they are sent to the alert processor. The signature-based IDS function is accomplished by using various rule sets. The rule sets are grouped by category (Trojan horses, buffer overflows, access to various applications) and are updated regularly.

The rules themselves consist of two parts:

■ **The rule header** The rule header is basically the action to take (log or alert), type of network packet (TCP, UDP, ICMP, and so forth), source and destination IP addresses, and ports

■ **The rule option** The option is the content in the packet that should make the packet match the rule.

The detection engine and its rules are the largest portion (and steepest learning curve) of new information to learn and understand with Snort. Snort has a particular syntax that it uses with its rules. Rule syntax can involve the type of protocol, the content, the length, the header, and other various elements, including garbage characters for defining butter overflow rules. If we want to generate new rules from existing rules it is known as generalizing SNORT rules.

Alerting/Logging Component

After the Snort data goes through the detection engine, it needs to go out somewhere. If the data matches a rule in the detection engine, an alert is triggered. Depending upon what the detection engine finds inside a packet, the packet may be used to log the activity or generate an alert. Logs are kept in simple text files, tcp dump- style files or some other form. Alerts can be sent to a log file, through a network connection, through UNIX sockets or Windows Popup (SMB), or SNMP traps. The alerts can also be stored in an SQL database such as MySQL and Postgress.

V. PERFORMANCE

Consider an Internet packet that contains a variation of a known attack, there should be some automated way to identify the packet as nearly matching a NIDS attack signature. If a particular

statement has a set of conditions against it, an item may match some of the conditions. Whereas Boolean logic would give the value false to the query 'does this item match the conditions', our logic could allow the item to match to a lesser extent rather than not at all. This principle can be applied when comparing an Internet packet against a set of conditions in a SNORT rule. Our hypothesis is that if all but one of the conditions are met, an alert with a lower priority can be issued against the Internet packet, as the packet may contain a variation of a known attack. While implementation, generalization in the case of matching network packets against rules, involves allowing a packet to generate an alert if:

• The conditions in the rule do not all match, yet most of them do;

• The only conditions that do not match exactly nearly match.

When implementing generalized rules, the execution time was 1 second to process and convert the original 1,325 rules into a total of 6,975 rules. The generalized Content execution time was 2 seconds to process and convert the same 1,325 original rules, into a total of 18,265 rules. These execution times would easily be acceptable for most potential uses, such as each time the SNORT rules were downloaded for signature updates. The increase in the number of rules affected the time spent processing network traffic data as follows:

• Using the original rules, Snort took approx 100 seconds to process 1,635,267 packets;

• Using the generalized (inverted) rules, Snort took approx 400 seconds to process the same packets;

• Using the generalized content rules, Snort took approx 1,000 seconds to process the packets. The change in SNORT's processing time is an increase of around four to ten times and roughly in line with the increase in the number of rules.

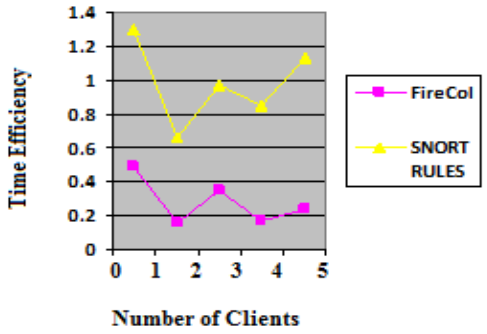

Figure 2: Time comparison results for FireCol and Snort Rule detection systems.

As shown in the above figure, it distinguishes the comparison results between both existing and proposed approaches developed in our application. In our existing approach we have to develop FireCol technique for detection of Denial-of-Service attacks in network communication. In this technique we are not providing any rule structure process for detection of those attacks present in the network communication. In this technique we was developed Intrusion detection system rules structure for developing network performance with equal priority values of each node present in the network.

In this section we describe the network performance results when we are using different rule structure for detection of Denial-of-Service attacks in network communication process. For this process we are developing different Snort rule structures like DOS, DDOS, Web-Attack, and SCAN. In our proposed approach we are developing different classification structure for each node present in network, and then they are calculating individual classification time establishing connection for detecting attacks. Those results were taking more time when compare to FireCol detection system. Because FireCol doesn't provide classification structure for each client in network.

VI. CONCLUSION & FUTURE WORK

In this paper, the proposed system extending FireCol to support different IPS rule structures will

help FireCol thwart other forms of DoS attacks especially the latest entrant Slow Read DoS attack. As further future work of FireCol, We Propose Snort's detection system which is based on rules. Like viruses, most intruder activity has some sort of signature. Information about these signatures is used to create Snort rules. These rules in turn are based on intruder signatures. Snort based detection system consists of several components: Sniffer, preprocessor, the detection engine, the output/ alert component. The detection engine makes use of snort rules. Snort rules can be used to check various parts of a data packet not just the header scanning adapted by prior approaches. A rule may be used to generate an alert message, log a message, or, in terms of Snort, pass the data packet, i.e., drop it silently. Thus enabling a detection system eliminating other forms DoS attacks such as Slow Read DoS attack. Snort Based DoS detection system can be a real time efficient and feasible implementation that can counter varying DoS attack forms. As further improvement of our proposed work we are developing IDS rule structure with limited access only, in this efficient results are generated according to presented rules only. In future we are developing our what are the rule presented in IDS we are developing all those rules and organize DDOS attacks efficiently.

VII. REFERENCES

[1] S Axelsson (2000) 'Intrusion Detection Systems: A Survey and Taxonomy', Chalmers University Tech Report, 99-15.

[2] Proctor, Paul E. The Practical Intrusion Detection Handbook. New Jersey: Prentice Hall PTR, 2001.

[3] Northcutt, Steven. Network Intrusion Detection, An Analyst's Handbook. Indianapolis: New Riders, 1999.

[4] Bace, Rebecca. "An Introduction to Intrusion Detection and Assessment: for System and Network Security Management." ICSA White Paper, 1998.

[5] G. Badishi, A. Herzberg, and I. Keidar, "Keeping denial-of-service attackers in the dark," *IEEE Trans. Depend. Secure Comput.*, vol. 4, no. 3, pp. 191–204, Jul.–Sep. 2007.

[6] T. Peng, C. Leckie, and K. Ramamohanarao, "Detecting distributed denial of service attacks by sharing distributed beliefs," in *Proc. 8th ACISP*, Wollongong, Australia, Jul. 2003, pp. 214–225.

[7] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney, "The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware," in *Proc. 10th RAID*, Sep. 2007, pp. 107–126.

[8] Sourcefire Inc, M Roesch and C Green (2006) 'SNORT Users Manual - SNORT Release: 2.6.0', http://www.snort.org

[9] J Hoagland and S Staniford (2003) 'Viewing IDS alerts: Lessons from SnortSnarf',http://www.silicondefense.com/research/whitepapers/index.php.

[10] D. Das, U. Sharma, and D. K. Bhattacharyya, "Detection of HTTP flooding attacks in multiple scenarios," in *Proc. ACM Int. Conf. Commun., Comput. Security*, 2011, pp. 517–522.

[11] H. Liu, "A new form of DOS attack in a cloud and its avoidance mechanism," in *Proc. ACM Workshop Cloud Comput. Security*, 2010, pp. 65–76.

[12] A. Sardana, R. Joshi, and T. hoon Kim, "Deciding optimal entropic thresholds to calibrate the detection mechanism for variable rate DDoS attacks in ISP domain," in *Proc. ISA*, Apr. 2008, pp. 270–275.

[13] I. B.Mopari, S. G. Pukale, and M. L. Dhore, "Detection of DDoS attack and defense against IP spoofing," in *Proc. ACM ICAC3*, 2009, pp.489–493.

[14] Jéerôme François, Issam Aib, Raouf Boutaba*," FireCol*: A Collaborative Protection Network for the Detection of Flooding DDoS Attacks*",* IEEE 2012 Transaction on Networking,Volume :PP, Issue:99.