DoS: Attack and Defense

Vincent Tai
Sayantan Sengupta

COEN 233 Term Project - Prof. M. Wang
# Table of Contents

1. Introduction  
   1.1. Objective  
   1.2. Problem  
   1.3. Relation to the class  
   1.4. Other approaches  
   1.5. Our approach  
   1.6. Statement of the problem  
   1.7. Area of investigation  

2. Theoretical Bases and Literature Review  
   2.1. Definition of the problem  
   2.2. Theoretical background of the problem  
   2.3. Related research to solve the problem  
   2.4. Advantage/disadvantage of those research  
   2.5. Our solution to solve this problem  
   2.6. Where our solution different from others  
   2.7. Why our solution is better  

3. Goals  

4. Methodology  
   4.1. Basic Architecture  
   4.2. How to generate/collect input data  
   4.3. How to solve the problem  
   4.4. Algorithm design:  
   4.5. Tools and Language used  
   4.6. How to generate output  
   4.7. How to test against hypothesis  

5. Implementation  
   5.1. Code  
   5.2. Flowchart  

6. Data analysis and discussion  
   6.1. Output generation  
   6.2. Output analysis  
   6.3. Compare output against hypothesis
6.4. Abnormal case explanation
6.5. Discussion

7. Conclusions and recommendations
   7.1. Summary and conclusions
   7.2. Recommendations for future studies

8. Bibliography
9. Appendix
Introduction

Objective
To simulate denial-of-service attacks against an Internet web server; then to successfully defend against those attacks.

Problem
Denial of service is an ongoing problem on the Internet today. From discontented Internet users to governments and corporations, DoS is becoming a popular way for groups of people to suppress each other. Successful DoS can result in effective censorship or major loss of revenue for those who depend on web presence for business. In addition, new methods of DoS have been developed in recent years, and an average of 27.9 attacks have been occurring every hour -- this problem is not going away; it is escalating.

Relation to the class
Denial of service is a problem that occurs over networks. A successful DoS can disrupt the usability of computer networks. By sending data in special patterns over the network, or by simply flooding packets, it is often possible for attackers to take down remote services. On the other hand, by configuring a network well, it may be possible to mitigate DoS attacks.

Other approaches
Other approaches to mitigating denial of service attacks often involve connection rate-limiting at the server. The server is still vulnerable to other attacks, such as bandwidth flooding, since its IP address is known.

Our approach
Our approach is to set up shield servers -- essentially proxies -- that forward connection requests to the actual server. The connection filtering is to be done at the shields themselves, not at the server. This makes the whole defense more scalable. If a shield gets taken down, the server is still alive, since attackers cannot access the actual server directly. This method is better since the server gets an extra layer of defense -- it is protected by shields, instead of being directly exposed to the world.

Statement of the problem
To mitigate DOS attack by adding an extra layer of protection (using shield nodes), so that the attack can be detected and mitigated even before the attack reaches the origin server.
Area of investigation
Although a number of methods already exist for DoS detection and mitigation, DoS attacks still frequently occur. Our area of investigation is the configuration of a network that is resilient to DoS attacks. We will be implementing another layer of security in addition to the existing firewall setup.

Theoretical Bases and Literature Review

Definition of the problem
Transport networks are the core elements of today’s communications systems. Network operators are able to interconnect a large number of users at the same time because of their capability to transmit data extremely quickly and securely. However, the increasing number of attacks against such networks raises the challenge to deploy effective safeguards to guarantee their availability.

Theoretical background of the problem
Denial-of-Service (DoS) attacks have become a major threat for communication networks during the past few years. With the rapid growth in the internet, users are opting for online trading, shopping and other critical online activities. These resources have to be protected from various types of attacks. The main purpose of the DOS attack is usually to degrade – or even disrupt – the normal operation of the attacked network, its constituting components, or provided services. At present, the vast majority of DoS attacks is directed against individual network components and the services they are offering. Their targets predominantly comprise network components that provide end-user services, such as Web servers or other kinds of service access points.

As the types and number of users of the internet increases, the requirement of an effective system to detect these attacks also increases. Our project deals with focusing on transport resources aiming to prevent users from setting up transport connections. With respect to the aspects being relevant for Denial-of-Service attacks against transport resources and potential attack schemes and scenarios, we present methods for the detection of such attacks and subsequent blocking of those attacks, which contribute to a comprehensive detection framework for auditing the transport network for potential attacks.

Related research to solve the problem
With the size of the internet increasing everyday, Denial-of-Service attacks have also been increasing at a high rate. Related research projects in this field provides a theoretical and statistical base which provides ample scope of detecting an attack. With the statistical data
analysis recorded over a period of time a signature set of rules is set for the detection of an attack.

**Advantage/disadvantage of those research**

With the statistical data observed for a traffic pattern over a substantial amount of time we can recognize significant deviations which will subsequently indicate an attack. The attack is identified and appropriate steps are taken to safeguard the available resources.

However, the detection happens in an origin server where the critical data is present. Also, the detection of an attack only happens once the server has already been partly compromised. Hence, most attacks are often detected once the attack has already taken place.

**Our solution to solve this problem**

- Attacker has no access to the origin server.
- Data is cached on nodal servers.
- Implementation of shield nodes.
- Setting up of firewalls for rate-limiting and bandwidth based attacks on the nodes and not in the destination servers.

**Where our solution different from others**

- We provide the detection mechanism even before the traffic reaches the destination server.
- As the attack is detected before the attacker could reach the destination server, the attack can be blocked/mitigated and we will be able to guarantee availability on the destination server.

**Why our solution is better**

In our implementation, the attacker gets no access to the origin server. The attacker accesses the shield nodes who in turn fetch and cache the required data from the origin server. The origin server connects only to the shield nodes, not the whole internet. Also, each of the virtual servers are rate limited and firewalled for any abnormal traffic.
Goals
Our goal is to simulate a Denial-of-Service attack and to completely secure availability of the destination server by using shield nodes which will detect and mitigate connection- and bandwidth-based DoS attacks.

Methodology

Basic Architecture

How to generate/collect input data
A Denial of Service attack is mainly based on sending the victim an overwhelming amount of traffic that it is not able to handle. Some of the methods include Internet Control Message Protocol (ICMP) flood, TCP SYN flood, resource utilization attacks, etc. We will receive the traffic from these end users and detect if these connections are legitimate at the shield nodes, even before the traffic reaches the main destination server.

How to solve the problem
We will use shield nodes as proxies to our main server. The shield nodes will have a custom HTTP proxy implementation. This custom HTTP proxy will normally accept client requests and
forward them to the server. However, it will also reject requests it thinks are part of a DoS attack. We plan to implement the shield nodes in front of the destination server which will detect and block the traffic even before it reaches the server. These shield servers adds a layer of security protection while still ensuring that content is delivered to the end user.

Key benefits of the Shield node include:

- Enhance destination server security and mitigate risk by protecting the application origin.
- Ensure that traffic passes through the nodes, where attacks can be detected and mitigated

**Algorithm design:**
The shield nodes will have an algorithm for determining whether to accept or reject connections; here is one example of such an algorithm:

```
wait for connection

does the connection's source exceed its connection limit?

no

does the connection's source exceed its bandwidth limit?

no

forward request to server

yes

receive server response

send response to client

close connection
```
Tools and Language used
We used Mininet to emulate a network containing our virtual hosts. We configured Mininet using Python; the software running on the shield nodes and attacker nodes will be implemented in C. The web server running on the origin server is Apache 2.

How to generate output
We will simulate a Denial of Service attack from the client end. Initially, without the implementation of our framework the attacker will be able to completely take down the destination server. Now, when we introduce our defense mechanism and re-introduce the attack, the shield nodes will successfully be able to detect and block traffic which are not legitimate while still serving legitimate traffic simultaneously.

How to test against hypothesis
As we mentioned earlier, our goal is to simulate a Denial-of-Service attack and to completely secure availability of the destination server. The server should be able to successfully serve legitimate client traffic while simultaneously detecting and blocking non-legitimate traffic coming from an attacker. Hence we would need to execute a DoS attack and at the same time send legitimate requests to the server and check for successful response.

Implementation
Code:
This project consists of two pieces of code: a simple connection flooder, and a shield node implementation. Both are written in C.

The connection flooder simply opens many connections to the targeted server; it does not do anything with those connections besides wait for them to close or reset. It is implemented using connect() calls on non-blocking sockets; it uses poll() to receive notifications when a connection is acknowledge, closed, or reset by the remote host.

The shield node is a defense against attackers trying to use up connections. Fundamentally, the shield node is implemented using single-threaded multitasking, with non-blocking sockets and poll() to determine readiness. The shield node implements the following in an attempt to maintain availability:

1. culling of the connection with longest inactivity period when it runs out of available task slots
2. rejecting new connections from IP addresses that have reached their connection limit
3. buffering requests, and opening a server connection only when a complete request is received
4. caching responses

Points (1) and (2) are intended primarily to maintain availability of the shield node itself. Points (3) and (4) are intended to reduce load on the main server, thus maintaining its availability.

Flow chart:
A flow chart of the shield node’s task logic is shown below; some logic is omitted for simplicity.

Data analysis and discussion

Output generation:
First, we flood the destination server with connections from the attacker node until the server is unable to handle any other incoming traffic. Next, we try to access the destination server web page from the client server to verify that the destination server is unable to respond to any incoming requests. Then, we activate our shield nodes that we have placed in front of our origin server. We redirect the attack to the shield nodes, since the server is to be accessed only via shield nodes. Then, we try to access a page on the server again, but through a shield node this time.

Output analysis:
When the server is not being attacked, the client is able to access the server normally. Without the shield nodes, when the attacker directly attacks the origin server, the client is no longer able to receive a response from the server. When the shield nodes are active, attacks on the shield nodes do not impact the client’s ability to access the shield node.

**Compare output against hypothesis:**

The webpage was available during the shielded attack, but not available during the unshielded attack. This indicates that our shield node was successfully mitigating the attack. For the attack scenario presented, this data is in agreement with our hypothesis.

**Abnormal case explanation:**

Even though we could successfully defend DOS attacks from a particular attack source our defense does not work as well against botnets which come from multiple IP addresses. For example, one hypothetically possible attack vector when the attacker has a large number of IP addresses is to send connections at an overall rate high enough to cause legitimate connections to be culled before they are able to receive data. Our next goal is to implement defense against distributed denial-of-service attacks using our shield nodes in a more scalable form.

In addition, we do not defend against bandwidth attacks. Thus, one attacker can try to bring down all the shield nodes by flooding them with bandwidth. However, we find this case unlikely if there is only a single attacker, as the attacker would need a huge amount of upload bandwidth to saturate the bandwidth of all the shield nodes all at once.

**Discussion:**

This project was limited in scope. The shield nodes are intended to be discovered through DNS, preferably with a DNS load balancer. This was not implemented, as it was outside our scope of investigation.

The shield node as implemented is intended mainly for use with static HTTP webpages; with dynamic web pages, the caching behavior will cause wrong or outdated pages to be returned. However, this was more intended to be a proof-of-concept of DoS mitigation, than a perfect HTTP reverse proxy.

The DoS attack implemented is a simple connection flood. This is a fairly primitive attack; we did not implement more sophisticated attacks, like slowloris or slow read. However, we expect that the shield node implementation should also provide protection against these attacks, although of course we were not able to actually test this.
Conclusions and Recommendations

Summary and conclusions:

The goal of this project was to attempt to:

- Enhance site security and mitigate risk by cloaking the application origin
- Ensure that web traffic passes through the shield nodes, where attacks can be detected and mitigated.

We attempted to do this by simulating the network topology using mininet. Generated huge amount traffic towards the destination server and successfully detected and mitigated the attack using our shield nodes.

Our simulation results indicate that with the help of the shield nodes the attack could be mitigated one layer before the destination server. We concluded this by trying to access the server with legitimate requests even when the attack was in place. The server responded without any issues.

Recommendations for future studies:

We would like to explore more on our existing system to successfully mitigate botnets distributed denial-of-service attack as well. By using an arbitrary number of shield nodes and using DNS load balancers we believe that we can achieve this.

Bibliography

Appendix
Submitted source code list:
dos-topo.py: the mininet topology used
connflood.c: a simple connection flooder
shield/: directory containing the shield node implementation in C
README: compilation and usage information