Department of Computer Science and Engineering

COMPUTER NETWORKS

UE16CS301

Session: August-December 2018

PROJECT TITLE : SIMULATING DENIAL OF SERVICE ATTACK USING SLOW LORIS

NAME OF THE STUDENT	**ROHAN MOHAPATRA**	**ROHAN RAJESH TALESARA**	**SANKARSHANA RAO**
SRN	**01FB16ECS307**	**01FB16ECS309**	**01FB16ECS342**
BRANCH	**COMPUTER SCIENCE AND ENGINEERING**	**COMPUTER SCIENCE AND ENGINEERING**	**COMPUTER SCIENCE AND ENGINEERING**
SEMESTER & SECTION	**V - F**	**V - F**	**V - F**

OUR IMPLEMENTATION

Slowloris is basically an HTTP Denial of Service attack that affects threaded servers. It works like this:

1. We start making lots of HTTP requests.
2. We send headers periodically (every ~15 seconds) to keep the connections open.
3. We never close the connection unless the server does so. If the server closes a connection, we create a new one keep doing the same thing.

This exhausts the servers thread pool and the server can't reply to other people.

ABOUT SLOWLORIS

A Slowloris is a type of denial of service attack tool invented by Robert "RSnake" Hansen which allows a single machine to take down another machine's web server with minimal bandwidth and side effects on unrelated services and ports.

Slowloris tries to keep many connections to the target web server open and hold them open as long as possible. It accomplishes this by opening connections to the target web server and sending a partial request. Periodically, it will send subsequent HTTP headers, adding to—but never completing—the request. Affected servers will keep these connections open, filling their maximum concurrent connection pool, eventually denying additional connection attempts from clients.

MITIGATING SLOWLORIS ATTACK

While there are no reliable configurations of the affected web servers that will prevent the Slowloris attack, there are ways to mitigate or reduce the impact of such an attack. In general these involve increasing the maximum number of clients the server will allow, limiting the number of connections a single IP address is allowed to make, imposing restrictions on the minimum transfer speed a connection is allowed to have, and restricting the length of time a client is allowed to stay connected.

In the **Apache web server**, a number of modules can be used to limit the damage caused by the Slowloris attack; the Apache modules **mod_limitipconn**, **mod_qos**, **mod_evasive**, **mod security**, **mod_noloris**, and **mod_antiloris** have all been suggested as means of reducing the likelihood of a successful Slowloris attack. Since Apache 2.2.15, Apache ships the module **mod_reqtimeout** as the official solution supported by the developers.

Other mitigating techniques involve setting up reverse proxies, firewalls, load balancers or content switches. Administrators could also change the affected web server to software that is unaffected by this form of attack. For example, lighttpd and nginx do not succumb to this specific attack.

MODULES

1. Socket: This module is used to initialize sockets, and establish connection with an IP address. socket.socket() functionis used to make a socket, socket.settimeout() is used to set timeout period for the socket, socket.connect() is used to establish connection, and socket.send() is used to send data across the connection.
2. Random: To generate random numbers.
3. Time: To get current time, and measure time between events.
4. argparse: Used to accept command line arguments.

INSTALLATION GUIDE

1. Connect the server and the clients to the same router/switch.
2. Setup an Apache Server on one **host machine** and let the other clients connect and access the web pages on the Server.
3. Let **A** be an attacker trying to force packets to huge sockets on the server.
4. Attacker **A** runs the code as follows:

```
python3 slowloris.py --ip [IP-to-be-attacked]
```

5. The program will start blocking all sockets on the server.
6. Once all 200 sockets are occupied in the server, the server has crashed and no legitimate user can access the web pages on the server.

PTO

TOOLS AND TECHNOLOGIES USED

•**Socket Programming using Python**: We used Python to open sockets on multiple ports to flood the server with unclosed TCP connections to successfully carry out the exploit.

•**Postman**: It's a handy HTTP request creator which we used to test the server's response time.

•**Browser**: We used a browser to simulate legitimate users.

•**XAMPP Server**: We used an Apache server to act like a simple HTTP server serving a simple webpage.

•**HTML, CSS, JavaScript**: We used basic web tech stack to design the webpage.