Issues in Information Systems

Volume 13, Issue 1, pp. 190-198, 2012

LIVING IN DENIAL - A COMPARISON OF DISTRIBUTED DENIAL OF SERVICE MITIGATION METHODS

Elizabeth L. Unrein, Washburn University, elizabeth.unrein@washburn.edu
Delaney L.S. Fish, Washburn University, delaney.fish@washburn.edu
Joshua Boeker, Washburn University, joshua.boeker@washburn.edu
Wenying Sun, Washburn University, nan.sun@washburn.edu

ABSTRACT

Denial of service attacks are becoming increasingly common. While good network security policies can help prevent a DoS attack, there is little that can be done to eliminate the chance of one happening. Therefore, mitigation of the effects of these attacks is a problem. Several packet filtering programs and mitigation techniques exist, but no one strategy has been tested and proven effective. In this experiment, we set up a network and server and simulate an HTTP request attack where the effectiveness of two Apache modules and a script called (D)DoS Deflate is tested. We collect data on server uptime and packets filtered by each program separately and in combination. Finally, we compare and contrast the methods and identify the one that provides the best mitigation against a DDoS attack.

Keywords: Distributed Denial of Service (DDoS), Information Technology (IT), Apache, Information Security

INTRODUCTION

In 2011, the Hong Kong Stock Exchange was attacked [8]. The attacks affected their website, which "is used to disseminate price-sensitive information." The Hong Kong incident serves as a reminder that DDoS attacks have the potential to damage financial markets. They can also be used as a distraction while attackers obtain information from elsewhere in the system. Last year's PlayStation Network attacks only took down Sony's website for 20 minutes, but the data of 77 million users was compromised [8]. The data included names, addresses, phone numbers, and encrypted credit card information.

A denial of service attack is any action meant to prevent normal (authorized) traffic from accessing a server. A distributed denial of service attack uses multiple hosts to carry out the attack. Denial of Service (DoS) attacks are always in the news, and this year has already seen several high-profile attacks on organizations [10]. The effects of DoS attacks are widespread. Even small attacks which take up less than 1Gbps of bandwidth can cause significant traffic disruption to a network [8]. Unfortunately, with the introduction of automated tools and commercial services, launching a DoS attack has never been easier. Anyone with time and an Internet connection can pull off a DoS attack. As more services move into "the cloud", protecting networks is becoming more important than ever.

The majority of DoS attacks are flood attacks, where the attacker overwhelms the target network with packets that appear legitimate in order to take resources and prevent legitimate traffic from accessing the network. UDP and ICMP flooding can also occur. Ping flooding is a type of ICMP flooding which can be avoided by disabling ping traffic. Application-layer attacks exploit vulnerabilities such as buffer overflow in victim equipment; thus, it may be difficult to defend against these without knowing the specific vulnerability being targeted.

Multiple utilities and services exist to combat the effects of these attacks. However, our literature review suggests there is little research comparing the effectiveness of the various techniques. In this study, we developed an infrastructure to test three methods of DDoS mitigation (two Apache modules and a script) and attempt to answer the following research questions:

> *1. Are the available Apache modules more or less effective at mitigating DDoS attacks than the script-based method?*

2. Of our selected methods, is there any significant difference in effectiveness at mitigation against DDoS attacks?
3. Would any of these methods be useful for organizations to implement in order to mitigate DDoS attacks?

We believe our study makes several contributions. First, this adds to the body of information about DDoS mitigation. Second, it has practical applications because these methods can be used in a real world setting. Third, it provides information to the developers of these utilities about the efficacy of their tools. The rest of the paper is organized as follows. In our literature review, we examine current methods of DDoS mitigation. In methodology, we explain how our study was conducted and how we gathered data. In the results section, we talk about the data we collected. In the discussion section, we find significance in the data and discuss what that might mean. Finally, in limitations, we explain what limiting factors our research had.

LITERATURE REVIEW

Denial of service attack defense can be separated into two stages: preventative mechanisms (prevention) and reactive mechanisms (mitigation). Most prevention techniques are simply good network security practice: computers should frequently be updated and scanned for viruses, policies should be in place on users' permissions, etc. If attacks have happened in the past, an administrator can look for patterns in protocols used and filter traffic at the gateways accordingly. Another prevention method is to use an application or appliance that monitors for anomalous traffic patterns. Finally, one can increase resources to the point DoS attacks pose little to no threat. In most cases, this method is too expensive to use practically. However, there are cloud services available that will increase bandwidth during the extreme traffic spikes consistent with a DDoS attack, rendering it mostly harmless [2].

Mitigation techniques involve detecting attacks after they have started and lessening the impact. Mitigation can begin with just firewalls and routers. If the origin of the harmful traffic can be identified, firewalls can block traffic from suspected source IPs and ports while allowing legitimate traffic. However, firewalls cannot determine legitimate traffic from harmful traffic on their own. Additionally, Firewalls may be located too far down in the network architecture to be effective at preventing DoS attacks. Routers have the advantage of blocking traffic before it enters the network and can filter traffic with ACLs. Routing filter techniques, such as blackhole routing and sinkhole routing, can be implemented. However, like firewalls routers cannot distinguish between harmful and legitimate traffic on their own. Because neither firewalls nor routers can make these distinctions, they are best when used by a skillful administrator who can identify potential sources of harmful traffic and react accordingly [8].

Traditional router and firewall security implementations are ineffective against more sophisticated DoS attacks. To rectify this problem, many vendors offer appliances made specifically for DDoS mitigation. Devices such as Cisco's Traffic Anomaly Detector monitor traffic, look for signs of deviation from normal traffic patterns, and alert administrators when anomalous traffic is detected. In the case of an attack, specialized appliances such as Cisco Guard exist to analyze and filter traffic; allowing legitimate traffic through [1]. Additionally, many companies that provide other security services, such as Verisign, provide DDOS monitoring and/or mitigation services that monitor and/or filter traffic off-site [8]. In our literature review, we found some information outlining methods to mitigate DDoS attacks [3, 2, 7] , but very little on comparing effectiveness of these methods. In particular, we found no studies comparing utilities similar to ours. Much of the current literature focuses on mitigating attacks from the network level as opposed to an application level [3, 7, 9]. These methods are not accessible to many individuals. Also, software mitigation methods, which are popular with smaller businesses, were rarely mentioned outside of advice on message boards. For these reasons, we decided to focus our research on comparing software mitigation methods.

RESEARCH METHODOLOGY

The first step was to narrow down our experiment to the mitigation of one kind of DDoS attack. We chose to test mitigation methods against application layer HTTP DDoS attacks. We chose these types of attacks for a few

reasons. HTTP DDoS attacks have been popular because they are so easy to execute. Because of the prevalence of this kind of attack, there are also many free programs available to aid in mitigation of these attacks. It would be useful to test the effectiveness of some of these methods.

The test was carried out on a network consisting of a web server running Apache 2.2, two Cisco 1841 routers to simulate a WAN connection, each connected to a Cisco Catalyst 2900 series switch. Though most organizations would have their web server behind a firewall, it was decided unnecessary for the test, as the attack was only carried out on port 80. While a firewall could be used to limit the number of connections on port 80 to help mitigate an attack, this would block legitimate web traffic as well. Blocking specific IPs with a firewall would require human intervention and was deemed unmeasurable for the test.

For the baseline test, Apache was set up with default settings, with the exception of enabling mod_status, for tracking purposes, on a computer running OpenSUSE. The hardware configuration was identical to that of the attacking hosts. We monitored the server using the mod_status module, which collects statistics about uptime and number of worker threads.

To execute the attack, four similarly-built hosts, connected to the opposite switch, ran a custom-made Java program (see Appendix A for source code) with 80 threads sending HTTP GET requests in an infinite loop. Under our settings, Apache was able to serve 150 concurrent connections. Our DDoS attack worked by occupying all of these connections, thus denying access to the website for legitimate users. To test whether legitimate traffic could get through, we simply tried to access the website from a fifth computer on the attacking network. The website was deemed inaccessible if the browser's request timed out while trying to access the page. To measure how quickly the server was taken down, we compared the timestamps of the first connection made and the first connection refused in the Apache access log. After collecting a baseline, measuring how long it took our web server to start refusing connections, we installed the (D)DoS Deflate.

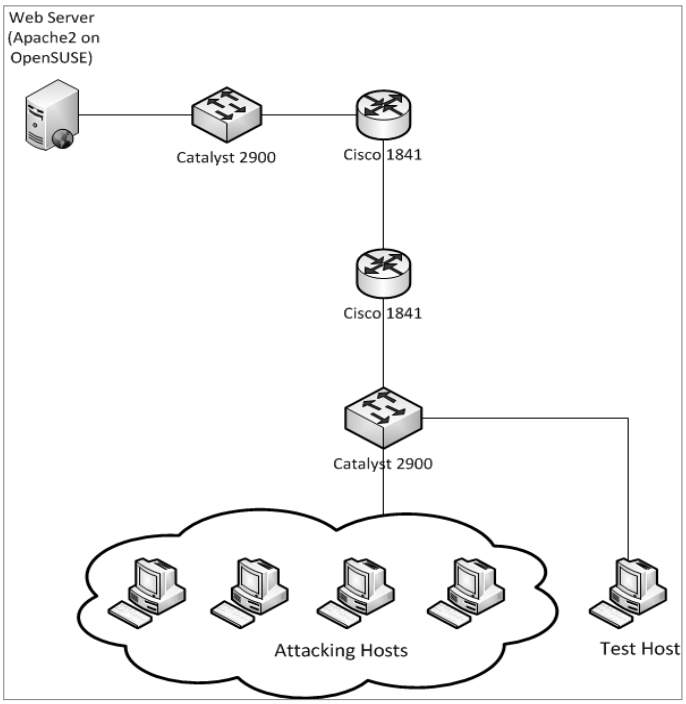

Figure 1. Network Diagram

(D)DoS deflate is a simple script that works by blocking IPs after they make a certain number of simultaneous connections to the Apache server. After being blacklisted, connections are allowed from the blocked IPs again after a pre-configured amount of time. Originally, we had planned to run the script with the default settings. However, by default the script blocked IPs after 150 connections. The attacking hosts were only making 80 connections each, so this high of a limit would be useless. Instead, we tested the script blocking IPs after 70 connections and 50 connections. In each case, the script was not intended to prevent DDoS attacks, but rather mitigate them, so it did not prevent the server from refusing connections initially.

Instead, we chose to measure how long it took for the server to begin accepting legitimate connections again. We did so by having our fifth, legitimate host attempt to connect to the server in 5 second intervals. This time range was chosen to give proper time for the web page to load, and because sending requests any more frequently could be contributing to the attack by making more unused connections. We measured the time it took for the web server to begin accepting connections again, using the script to block IPs after 70 simultaneous connections, and then again after 50 connections.

We then uninstalled (D)DoS Deflate and tried mod_evasive. mod_evasive is an Apache module created for DoS and DDoS attack mitigation. It will block access from any IP that attempts to make more than 50 (or any other configured amount of) simultaneous requests or requests the same page multiple times in less than a second. As with (D)DoS Deflate, we had mod_evasive set to allow 70, and then 50 concurrent connections. All other settings we left as the default.

Mod_security is a web application firewall meant for blocking application layer attacks, including HTTP DDoS attacks. It is not a tool made specifically for preventing or mitigating DoS attacks, but rather to act as a firewall and to prevent various kinds of attacks. Much like the others, mod_security's DoS mitigation functionality works by blacklisting IPs that make too many requests in a defined period of time. For our test of Mod_security, we used the default settings.

RESULTS

During our baseline test, it took an average of 10 seconds for the server to stop responding. Without utilities, the server was completely inaccessible for the duration of the attack.

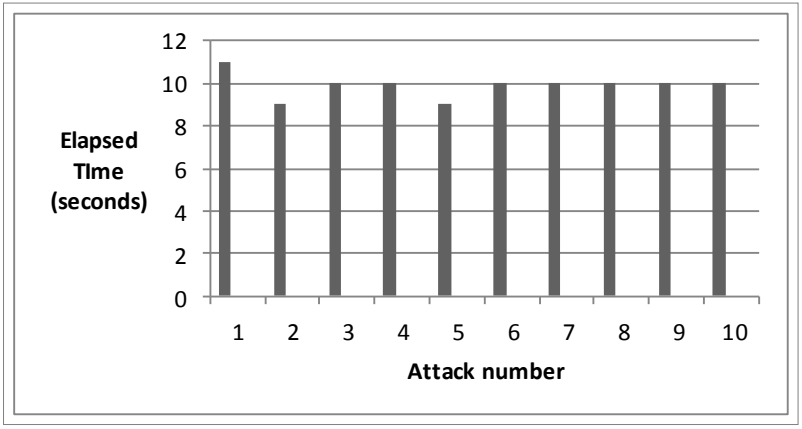

Figure 2. Baseline Test Results

After our baseline test, we installed the (D)DoS Deflate script. Because our utilities are meant to mitigate attacks, and not prevent them, the time it took for the server to go down never changed. What did change was how much time passed before a web page on the server was accessible again.

Under (D)DoS Deflate, the server did eventually come back up. We tested it with two different configurations. In one, the setting for maximum requests per IP was set at 70, and in the second, it was set at 50. This made a small, but noticeable difference: the average time for the server to come back up went down from 40 seconds to 33.5 seconds.

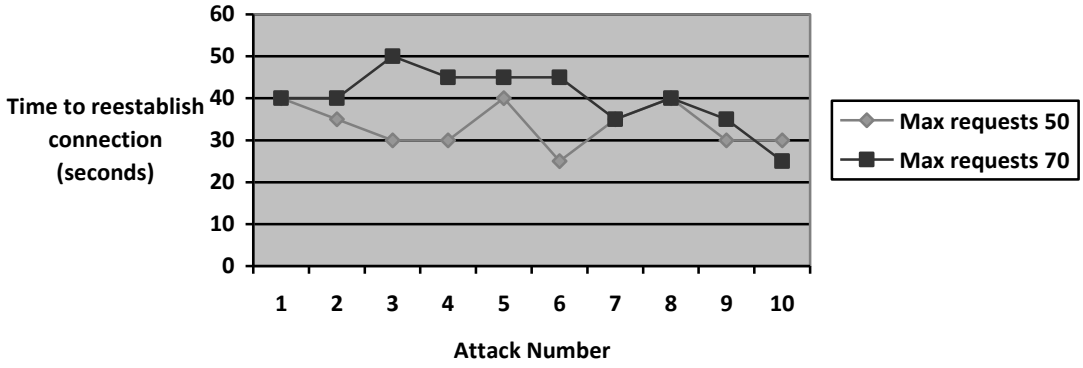

Figure 3. (D)DoS Deflate Results

Next, we tested mod_evasive. This, too, had a setting for maximum connections per IP, which we configured at 70 and 50. With mod_evasive installed and set to 70 maximum requests, the server came back up in an average of 59.5 seconds. When we changed the setting to 50, it came up in 56.5 seconds on average.

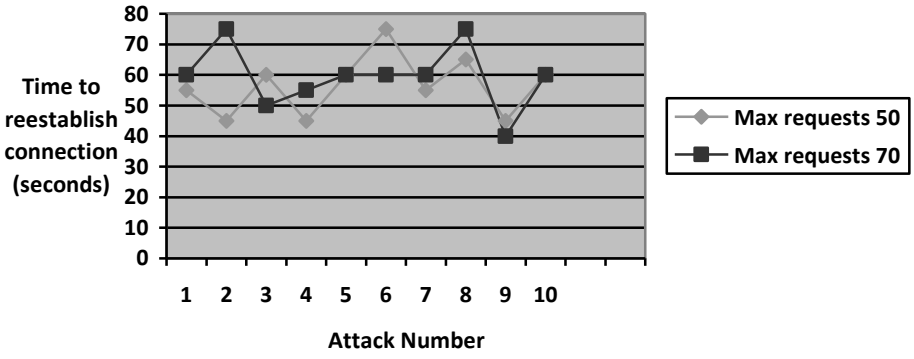

Figure 4. Mod_evasive Results with Maximum Connections Option at 50 and 70.

For our final mitigation method, we tested mod_security. Mod_security had very inconsistent results, ranging from succeeding in 30 seconds to 1 minute and 25 seconds. The average time for the server to come back up under mod_security was 60.5 seconds.

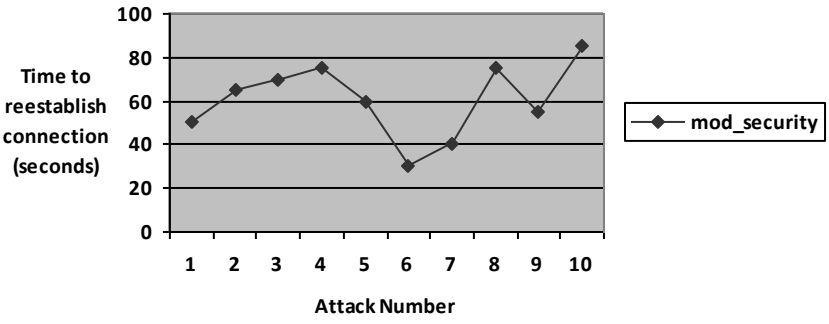

Figure 5. Mod_security Results

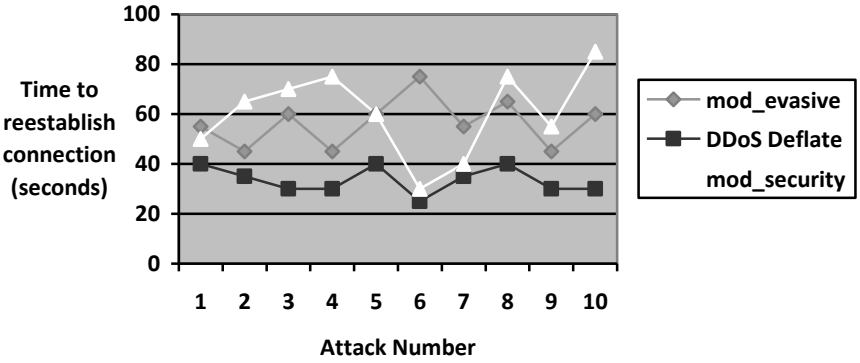

Figure 6. A Comparison of Mod_evasive, mod_security, and (D)DoS Deflate.

DISCUSSION

These utilities are best suited to small servers running Apache. (D)DoS Deflate is the only method we tested that can be used on servers not running Apache, but because it is a shell script, it can only be run in a Linux environment. These methods are also only suitable for protection against HTTP DDoS attacks and would not be effective against other kinds of DDoS attacks.

(D)DoS Deflate succeeded in its purpose. Although it could not stop the server from going down, it blacklisted the IPs of our attack computers during each test and allowed legitimate traffic to access the server. It is also scalable. If the administrator knows about the type of traffic the server receives, the configuration file can be tweaked to suit the needs of the server. The max requests per IP can be scaled up or down. It is therefore a viable solution for mitigating DDoS attacks.

Mod_evasive fared worse. Though it was effective, it was less so than (D)DoS Deflate. However, because Apache is cross-platform, it would be possible to run in a Windows environment, unlike (D)DoS Deflate. Mod_evasive can also be configured to suit a specific host, but it lacks in good error logs and documentation. (D)DoS Deflate is easier to troubleshoot and validate.

Mod_security, though it worked, was highly inconsistent, lacking in logs, and not easily configured for a specific server's needs. It performed the worst out of all three. Therefore, mod_evasive is the better alternative for an Apache environment in terms of mitigating DDoS attacks. Mod_security does have the advantage, however, of providing protection against other kinds of attacks and against malware. These aspects were not tested in our experiment, but mod_security may be a valid solution for those looking for protection against a variety of attacks. Additionally, mod_security may be more effective with custom rules rather than the defaults.

While all three utilities tested were able to mitigate our DDoS attack, (D)DoS deflate consistently mitigated the attack more quickly than either Apache mod. It also had options for whitelisting IP addresses that may legitimately provide high amounts of web traffic and to raise or lower the number of allowed connections to be scalable to many different environments. Thus for a Linux environment, (D)DoS Deflate was the most effective utility for mitigating DDoS attacks.

LIMITATIONS AND FURTHER RESEARCH

Limitations

We decided to limit the scope of our experiment to mitigation of HTTP DDoS attacks, but as illustrated in our introduction, there are many different types of DDoS attacks that would not be mitigated by the methods we tested. Budget was our biggest constraint; we were only able to test mitigation methods that were free. Even after narrowing down to HTTP DDoS attacks, some methods we found for mitigation were meant for use on the ISP side, and we did not have the proper equipment to emulate a connection to an ISP. Our architecture was also a limiting factor. Our Apache2 configuration only allowed for 150 simultaneous connections. This limit would be much too small for a large organization. Because of the limitations of Apache and of our hardware resources, we were only able to test our methods on a small scale. Apache is used often for small businesses, however, and our research should prove useful on the small scale. Our network architecture was not necessarily typical, either. The only traffic on the web server's network was traffic to and from the server itself. Simply put, different organizations are going to have different network traffic patterns that we could not expect to emulate. Another limitation of our experiment is the lack of a human element. Common mitigation techniques, such as with a firewall or ACLs on a router, usually involve a network administrator observing traffic spikes and reacting by changing router or firewall configuration accordingly. The effectiveness of these methods relies somewhat on the skill of the administrator and therefore could not be measured for our tests.

Further Research

For further research, it may prove useful to test these methods in combination with each other - perhaps (D)DoS Deflate would be effective combined with an Apache module - or with other methods such as using dedicated appliances or hardware firewalls. It would also be interesting to see how well these mitigation methods would work under high network traffic as well, i.e. a large scale DDoS attack where packets may get queued at a router or switch. Testing these same methods on a server that could allow hundreds of simultaneous connections and see if the results are scalable for a larger organization would be a very valuable addition to our research.

REFERENCES

1. Cisco. (2012). cisco guard ddos mitigation appliances. Retrieved from http://www.cisco.com/en/US/products/ps5888/index.html
2. Cross, K. (2011). Steps to defeat a ddos attack on your organisation. Database and Network Journal, 41(5), 16.
3. Garg, A. (n.d.) Mitigation of DoS attacks through QoS regulation. Retrieved from http://www.itsec.gov.cn/docs/20090507160949125372.pdf

4. Patrikakis, C., Masikos, M., & Zouraraki, O. (2004). Distributed denial of service attacks. Internet Protocol Journal, 7(4), Retrieved from http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-4/dos_attacks.html
5. Rashid, F. (2011). Ddos attack knocks out hong kong stock exchange news website. eWeek, Retrieved from http://www.eweek.com/c/a/Security/DDoS-Attack-Knocks-Out-Hong-Kong-Stock-Exchange-News-Web-Site-389466/
6. Rashid, F. (2011). Sony data breach was camouflaged by anonymous ddos attack. eWeek, Retrieved from http://www.eweek.com/c/a/Security/Sony-Data-Breach-Was-Camouflaged-by-Anonymous-DDoS-Attack-807651/
7. Subramani, R. (2011). Denial of service attacks and mitigation techniques: real time implementation with detailed analysis. Retrieved from http://www.sans.org/reading_room/whitepapers/detection/denial-service-attacks-mitigation-techniques-real-time-implementation-detailed-analysi_33764
8. Verisign. (2012). Products and services - network intelligence and availability. Retrieved from http://www.verisigninc.com/en_US/products-and-services/network-intelligence-availability/index.xhtml
9. Walfish, M. (2006). DdoS defense by offense. Retrieved from http://nms.lcs.mit.edu/papers/ddos-offense-sigcomm06.pdf
10. Zheng, Y. (2011). Distributed denial of service attack principles and defense mechanisms. *Advances in Natural Science*, 4(2).

APPENDIX A

```
/**
* This program launches an http DoS/DDoS attack on the owner of "TARGET_IP". It works by opening many
* simultaneous connections.
**/
import java.io.*;
import java.net.*;

public class ThatProgram
{
   private static final String TARGET_IP = "http://0.0.0.0";    //Change to target URL or IP address.
   private HttpReqThread hrt;
   private URL url;

   public ThatProgram()
   {
     for(int i=0; i<80; i++)  // creates 80 threads and thus up to 80 simultaneous connections with the web server.
      {
           hrt = new HttpReqThread();
           hrt.start();
      }
   }
   private class HttpReqThread extends Thread
   {
      public void run()
      {
         while(true)
         {
            try
            {
               url = new URL(TARGET_IP); //the URL class works fine with an IP address.
               HttpURLConnection connection = (HttpURLConnection) url.openConnection();
               connection.setRequestMethod("GET");
               BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
               String line;
               while ((line=reader.readLine())!=null) //Prints the html from the target web page onto the console.
                  System.out.println(line);                 //This serves to keep the threads busy for a little while and the
            }                                                //connection to the server open longer.
            catch (Exception e)
            {
               System.out.println("Whoops!" + e);
            }
         }
      }
   }
   public static void main(String[] args)
   {
      new ThatProgram();
   }
}
```