Code Review of an HTTP GET Attack

Hauge, Ole André
Department of Information Security and Communication Technology
Norwegian University of Science and Technology (NTNU)
Gjøvik, Norway
oleahau@stud.ntnu.no

***Abstract*—The HTTP protocol is used to connect devices like computers and servers to improve processes, both at work and in their personal lives. Sadly, these connections are vulnerable to attacks and exploitation. A popular method of attacks used to disrupt this application layer communication is HTTP GET attacks. In this study, we try to improve future research regarding detection and prevention by understanding how an adversary develops HTTP GET attacks. We used static code analysis on the "saphyra.py" python script and tested the script on a LAN, and towards an online webserver. We found that an adversary can use Python modules to create a simple, yet effective HTTP GET attack. The analysis yielded information about how an attacker obfuscates the traffic, and what such scripts and network traffic might look like. This knowledge can be used to improve detection methods. Further research is needed to discover new methods for detection or prevention.**

***Index Terms*—Denial of Service, Distributed Denial of Service, HTTP, GET, slowloris, mitigation, prevention, detection**

I. Introduction

This paper is inspired by the famous quote of Sun Tzu: *"[...] If you know the enemy and know yourself [...]"* [1], in that it aims to further improve research regarding detection and prevention by understanding how an adversary would develop an HTTP GET attack.

In today's society people use networks like the Internet to connect devices like computers and servers to improve processes, both at work and in their personal lives. These connections are enabled due to protocols like HTTP(S), which are used to exchange information. Sadly, these connections are vulnerable to attacks and exploitation. A popular method of attacks used to disrupt this communication is Distributed Denial of Service (DDoS) attacks. A Denial of Service attack (DoS) is an attack where an attacker makes a service or network component unavailable for legitimate users by flooding the target with requests from a host [2]. An evolved form of this attack is known as a Distributed Denial of Service (DDoS), which utilizes several host machines, often huge botnets, to send massive amounts of requests at the target [2]. These types of attacks target different vulnerabilities in the OSI model. Some, like SYN flood attacks, target the network layer while others like reflection attacks are aimed at the transport layer [3]. This paper focuses on the application layer.

The rest of this paper is structured as follows: Section two describes the different HTTP methods used in DoS attacks and what mitigation and prevention techniques are available. Section three explains the methodology for the study. Section four presents the findings of the study. Section five discusses the findings further with a focus on relevance, capability, and possibilities. Section six concludes the paper followed by suggestions for future work in section seven.

II. Background

In this section we present the necessary information about the methods used in DoS attacks and what mitigation and prevention techniques are available. We further mention the Saphyra script since it is the main focus of this papers analysis.

A. HTTP Flood Attack

The HTTP flood attack aims to overload the target server with HTTP requests by monopolizing the connections to the server. This method is typically used as a DDoS tool. There are two variants of this attack: HTTP POST attack and HTTP GET attack [2]. In this paper, we focus on the HTTP GET attack.

B. HTTP GET Attack

The HTTP GET attack aims to deny the service to legitimate users by saturating the connections repository of the target server by sending HTTP GET requests.

A special form of this attack is known as *Slowloris*. Its goal is to consume as many connections as possible at the server. It exploits how the target server use threads to keep concurrent connections open. By sending HTTP GET requests that never close the connections will stay open longer and consume more resources. This will eventually consume all connections at the server while using minimal of the attackers bandwidth, and a denial of service will occur [2].

C. Saphyra

Saphyra is a DDoS tool targeting the application layer by sending genuine HTTP GET requests with various obfuscation methods applied. Internet chatter indicates that it was responsible for taking down government websites like NASA in 2016 [4]. It has been sold as a tool on the Dark web and articles about the tool seem to have become more common during 2020 as the tool became easily accessible on GitHub [5].

D. Mitigation and Prevention

Research has been and is being, conducted on the topic of DoS and DDoS mitigation. Classic mitigation and prevention techniques are based on analyzing traffic patterns to detect the attack [2]. After detection, different filtering techniques are applied to mitigate the attack. Modern techniques include the use of cloud networking [6], [7] and hardware equipment developed to handle DoS and DDoS attacks [8], [9].

III. Methods

We aimed to understand how an adversary creates an HTTP GET attack, to better understand how to detect and prevent it. We decided to focus on the "saphyra.py" script since it was easily available. It can be downloaded [5] or found in appendix A. Static analysis of the script was conducted to gain an understanding of its workings. This entails a line-by-line readthrough. Afterward, a lab test was conducted to test the script by monitoring the produced network traffic between the attacker and the server with Wireshark.

A. Experimental Setup

The lab test consisted of two computers, "A" and "B", connected in a LAN. "A" ran a simple web server, while "B" ran the "saphyra.py" script targeting the IP address of the webserver. Further, a test was conducted towards an online web server, with the owner's consent [10].

Wireshark ran on "B" during the attack to capture the packet traffic. These packets were later analyzed to verify the creation and number of packets sent from *"saphyra.py"*. To monitor the connectivity and baud rate towards the server we ran *"ping*[1]*"* with the *"t"* option to continuously check the connection to the webserver running on client "A".

B. Prerequisites

The scope of the study was meant to facilitate discovery of new mitigation methods, not discover them.

To grasp the underlying aspects of this paper it is viewed as a prerequisite to have some prior knowledge of the HTTP protocol and networking.

IV. Results

After conducting the code analysis we were able to map out the structure of the script and the functionality. With the dynamic testing of the code, we managed to confirm that the packets were modified as intended, but we did not manage to take down the server. Below we present the relevant findings.

A. Structure

The script is written in python and is 3,532 lines long. The required lines to run the attack is only 156. It contains 7 global variables, 8 functions and 2 classes, in addition to the `execute` part of the program. Some of the functions is used to modify and randomize the HTTP header before transmitting the packets to the target. The `useragent_list()` function contains 3,177 different user agent entries, and the `referer_list()` function contains 181 different entries. This gives the possibility of 574,856 unique combinations, not considering the *"URL"*-, *"buildblock"*-, *"keep-alive"*-, and *"host"*-values.

At the beginning global variables are declared. Following is the declaration of the 8 functions and two classes which are listed in table I.

At last there is the `execute` part of the script that checks the users inputs, and starts the thread if they meet the conditions. This is done with a nested if-statement and the sys.argv(), which stores the command line arguments in an array.

```
#execute
if len(sys.argv) < 2:
    usage()
    sys.exit()
else:
    if sys.argv[1]=="help":
        usage()
        sys.exit()
    else:
        print "Starting the Attack"
        print "ANONYMOUS"
        if len(sys.argv)== 3:
            if sys.argv[2]=="safe":
                set_safe()
        url = sys.argv[1]
        if url.count("/")==2:
            url = url + "/"
            m = re.search('http\://([^/]*)
            /?.*', url)
        host = m.group(1)
        for i in range(700):
            t = HTTPThread()
            t.start()
        t = MonitorThread()
        t.start()
```

We see that if less than two arguments are supplied by the user then the script will print the usage information of the script and exit, which means that the user started the program without supplying any arguments as the first argument (sys.argv[0]) is the program itself. The same will happen if the first user argument (sys.argv[1]) is set to "help". From the second nested else-statement we see that we need to supply at least one argument for the script to run, which should be an URL address. This URL will be cleaned and used as the host variable. The URL and host variables are then used in the `HTTPThread()` and `MonitorThread` functions.

Furthermore, the script will check and see if the second user argument (sys.argv[3]) equals "safe" and set the set_safe flag. Other arguments supplied after the URL will simply be ignored by the script.

In this version of the attack, the user ends up with 700 HTTP GET threads, which we see of the for-loop. Each thread will run until the target URL replies with an HTTP 500 error message. This means that there is an internal server failure at

[1]A common tool used to measure the round-trip-time of transmitted data between devices. It gives a latency metric in milliseconds.

Nr.	Type	Name	Description
1	Function	`inc_counter()`	An incremental counter for the number of requests.
2	Function	`set_flag()`	Used to trigger different states in the code.
3	Function	`set_safe()`	Can be added by the user after the URL when running the script to make it auto-shut after the DoS attack.
4	Function	`useragent_list()`	Creates an user agent array that is used by the `httpcall()` function.
5	Function	`referer_list`	Used to generate a referrer array that is used by the `httpcall()` function.
6	Function	`buildblock()`	Used to build a random ASCII string that is used in the `httpcall()` function.
7	Function	`httpcall()`	Uses the three earlier mentioned methods and creates an unique, authentic HTTP GET request. It returns the "code" value used to stop the `HTTPThread()`.
8	Function	`usage()`	Shows how to use the attack.
9	Class	`HTTPThread()`	Used to start threads that will run the `httpcall()` function until the *"flag"* value "2" is triggered. This happens when the returned *"code"* value in `httpcall()` equals 500 and the *"safe"* value is present.
10	Class	`MonitorThread`	monitors the HTTP threads and counts the requests as long as the *"flag"* value is zero. Otherwise it prints a message if the *"flag"* value is two.

TABLE I: Functions and Classes in Saphyra

the target. The attacker could potentially stop the attack by canceling the script. This can be seen in the following class:

```
class HTTPThread(threading.Thread):
    def run(self):
        try:
            while flag<2:
                code=httpcall(url)
                if (code==500) & (safe==1):
                set_flag(2)
        except Exception, ex:
            pass
```

B. Lab Testing

Running the script shows that it does not increase the time latency or baud rate of the connection as expected. Analyzing the network traffic using Wireshark shows that the script is capable of producing about 7,000 HTTP GET requests per second. We did find that it managed to create random GET requests, user agent, and referrer fields.
Others have had similar results when testing the script [4], [10].

V. Discussion

The fact that the script is written in python means that it can be deployed on any device. The botnet this tool is intended for could therefore be cross-platform based and make detection harder as the traffic cannot be filtered based on a specific type of originating platform.

A. Techniques

The attack utilizes four techniques to be as effective as possible:

1) **Obfuscation:** It hides the source identity by creating random user-agent values for each package. In doing so the attacker hides information about what platform is used [11]. This could be a giveaway in the detection of

the attack if Python is used to send the traffic. The same is done to hide the referrer information in the header, thus making it harder to trace [12]. However, the attack does not change or hide its actual IP address and is therefore reliant on a larger botnet to reduce the risk of being traced.

2) **Disguising:** It utilizes legitimate HTTP GET traffic to monopolize the server connections by never completing the request, and exploits the fact that enterprise-grade intrusion detection and prevention systems are relying on signatures to fend of attacks. Therefore is even harder to detect and prevent.
3) **Persistence:** The script keeps the connections open longer by modifying the "keep-alive" option in the header field, defining a time window for each connection.
4) **Resource consumption:** As McMillen points out in his article [4], to put a further load on the target server, the script uses the "cache" header field to define the use of no-cache. Resulting in the server loading a unique web page for each connection.

B. Attack Development

The creation of such an attack is evidently easy. One needs basic knowledge about the HTTP protocol and python programming language. The script is simple and poorly written, which begs the question: *"How devastating will a similar attack written by a government actor be?"*. The script can easily be improved. For example, adding bigger lists of user agents and referer's would make for even more unique combinations. The number of threads created per instance of the script can be optimized to suit the machine it is running on, thus giving the attack optimal effect. However, this might make it easier to discover, since the sudden spike in CPU usage on the computer could lead to the discovery of botnet malware.

C. Testing

Using Wireshark we saw that approximately 7,000 HTTP GET requests were created per second. The content of the header was modified as the static code analysis suggested. The nature of the code seems to indicate that this would be a possible DDoS tool for an attacker. With more time the next step would be to scale up the attack with more hosts.

D. Short Comings of the Study

We were not able to verify the validity of the attack in our lab environment. The ICMP test showed no noticeable change in usage of bandwidth. The reason for this could be that:

1) The tool needs a botnet to be effective.
2) The code is a "slowloris" attack.
3) The code does not work well enough.

If we are to believe the author of the script, it is affiliated with the "Liphyra Botnet". It did, allegedly at the time the script was written, consisting of 1,798,445,657 bots. This could indicate that the tool needs to be used with a botnet to be effective. Further bragging from one of the authors indicates a capability of taking down government websites. The claim was investigated in a YouTube video [13], made by the author, showing how this was done. However, the websites targeted in the video do not seem to be "official". Thus, the claim remains weak at best.

If this is a "slowloris" attack it would explain the low bandwidth, and the reason why the target server did not crash could be because the webserver was designed to withstand such attacks by default [14]. A characteristic of this attack is to maintain the connections to the server to appear as "slow" traffic. The script could try to accomplish this by modifying the *"keep-alive"* HTTP header field to keep the connection open longer.

E. Mitigation and Prevention

As seen in the script, it is fairly simple to create a multiplatform tool. The countermeasures towards such attacks are to limit the rate of incoming connections and delay binding which implements load balancing software [2]. Other researchers have found a way to detect HTTP GET attacks by analyzing page access behavior [15]. The attack is hard to distinguish since it looks like legitimate traffic. As earlier mentioned modern uses of cloud services could help in mitigating the risk of the service going down. However, if the company is not careful the use of an elastic cloud scaling service, such as Amazon AWS/ECC, might imply that resources are added to the service as the traffic increases. This means that the company might suffer from a huge monetary loss due to overspending on cloud resources. Therefore the company has to be careful when signing the agreement with the cloud service provider.

F. Future Research

This study was time-limited, given more time it would be preferable to test the attack on multiple different web servers, and with a larger number of hosts. Further research should be conducted on page access behavior, network behavior [16], pattern recognition, cloud solutions [6], [7], and AI techniques [17], [18] to be able to detect and prevent such attacks.

VI. Conclusion

An adversary can use basic Python modules to create a simple, yet effective HTTP GET attack. Using Python has the bonus of being cross-platform capable. As discussed, the attack is obfuscated and made more effective by modifying the HTTP header fields. We were not able to discover new methods for the detection or prevention of HTTP GET attacks. However, we were able to confirm how an attacker obfuscates the traffic, and what such scripts and traffic might look like. This knowledge can be used to improve detection methods. Promising work from other researches is looking at network behavior detection using AI and ML, which further confirms the importance of this research area.

References

[1] Sun-tzu and S. B. Griffith, *The art of war*. Oxford: Clarendon Press, 1964.

[2] W. Stallings and L. Brown, *Computer Security: Principles and Practice*. 330 Hudson Street, New York, NY 10013: Pearson, 2018.

[3] H. Obaid and E. Abeed, "Dos and ddos attacks at osi layers," pp. 1–9, 01 2020.

[4] D. McMillen, "Dissecting a hacktivist's ddos tool: Saphyra revealed." https://securityintelligence.com/dissecting-hacktivists-ddos-tool-saphyra-revealed/.

[5] H1R0GH057, "Anonymous/saphyra.py." https://github.com/H1R0GH057/Anonymous/blob/master/saphyra.py. [Online; accessed: 21.10.2020].

[6] O. Osanaiye, K.-K. R. Choo, and M. E. Dlodlo, "Distributed denial of service (ddos) resilience in cloud: Review and conceptual cloud ddos mitigation framework," *Journal of Network and Computer Applications*, vol. 67, 01 2016.

[7] T. Alharbi, A. Aljuhani, and Hang Liu, "Holistic ddos mitigation using nfv," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 1–4, 2017.

[8] S. Agarwal, T. Dawson, and C. Tryfonas, "Mitigation via regional cleaning centers,"

[9] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, q. li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," 01 2020.

[10] NetworkChuck, "i bought a ddos attack on the dark web (don't do this)." https://www.youtube.com/watch?v=eZYtnzODpW4&list=PLs83vifxOnNzaDa-y1hFrk0CHRaKMmX8a&index=11&t=40s&ab_channel=NetworkChuck.

[11] M. contributors, "User-agent." https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent.

[12] M. contributors, "Referer." https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer.

[13] H. StroKE, "Taking down nasa subdomains + saphyra ddos download (lulzsecglobal + gsh)." https://www.youtube.com/watch?v=Bk-utzAlYFI&ab_channel=HaXStroKE.

[14] Wikipedia contributors, "Slowloris (computer security) — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Slowloris_(computer_security)&oldid=983975024, 2020. [Online; accessed 24-October-2020].

[15] T. Yatagai, T. Isohara, and I. Sasase, "Detection of http-get flood attack based on analysis of page access behavior," pp. 232 – 235, 09 2007.

[16] Yi Zhang, Qiang Liu, and Guofeng Zhao, "A real-time ddos attack detection and prevention system based on per-ip traffic behavioral analysis," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 2, pp. 163–167, 2010.

[17] B. Zhang, T. Zhang, and Z. Yu, "Ddos detection and prevention based on artificial intelligence techniques," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1276–1280, 2017.

[18] Y. Wu, H. Tseng, W. Yang, and R. Jan, "Ddos detection and traceback with decision tree and grey relational analysis," in *2009 Third International Conference on Multimedia and Ubiquitous Engineering*, pp. 306–314, 2009.