HTTP/2 Cannon: Experimental analysis on HTTP/1 and HTTP/2 Request Flood DDoS Attacks

HTTP/2 Cannon: Experimental analysis on HTTP/1 and HTTP/2 Request Flood DDoS Attacks

David Beckett, Sakir Sezer
CSIT, Queens University Belfast, Northern Ireland

***Abstract*—Distributed Denial of Service (DDoS) attacks are a frequent cyber attack vector which cause significant damage to computer systems. Hypertext Transfer Protocol (HTTP), which is the core communication protocol of the internet, has had a major upgrade and is released as RFC 7540. This latest version, HTTP/2, has begun to be deployed in live systems before comprehensive security studies have been carried out on its risk from DDoS. In this piece of research we explore using experimental methodology, the DDoS risk posed by the upgraded functionality of the HTTP/2 protocol, in particular its risk from a flood attack. Our results show that a website implementing HTTP/2, scales up the flood attack magnitude, increasing the risk from DDoS.**

***Index Terms*—DDoS, HTTP2, Flood, Attack, Apache, nghttp2, Nginx, Vulnerabilities**

I. Introduction

Cyber-attacks have become a common sight in the news. These attacks range from simple attacks able to take websites offline to sophisticated malware featuring zero day exploits.

When a distributed attack aims to take a network or service offline, it is classified as a Distributed Denial of Service Attack (DDoS). As more key services are migrated to the internet, DDoS is becoming an increasing worry for corporations and governments due to the risk of critical services being taken offline.

DDoS attacks are growing in popularity[1] and their attack magnitudes are rapidly increasing. The most common attack target for DDoS is the network infrastructure due to its limited bandwidth and connection capacities [2]. Using relatively simple attack tools, these network links can be flooded with junk data to take the connection offline. These attack tools are available for download online for self-deployment or alternatively a large botnet can be rented using a DDoS booter to carry out the attack, accessible from a simple web store interface. DDoS attacks can easily be started, requiring not even the technical abilities of a script kiddie, all that is currently required is a payment method and an ability to use an online shop.

The Mirai botnet has taken over thousands of Internet of Things devices such as webcam, and has taken several key internet websites offline. Cloudflare, a DDoS CDN provider, has recorded attacks reaching 1.75M HTTP requests per second[3].

Due to the simplicity of many of these attack tools, it would be easy to dismiss the research potential of this area, however if a sophisticated threat attacker were to become motivated to launch these attacks, the attacks will cause create significantly more damage and be harder to mitigate without significantly scaling up network resources, which will eventually become economically unsustainable.

Clients commonly communicate with web servers using the Hypertext Transfer Protocol (HTTP) protocol. In 2015 a new version of HTTP was released, HTTP/2[4]. This latest version of the HTTP protocol expanded the protocols capabilities with the aim of decreasing website load times. HTTP/2 implements multiplexed streams allowing multiple requests and responses to be sent in a single packet, as well as introducing higher levels of header compression with the aim of reducing packet sizes. These new capabilities are aimed at improving the system for benign users however there is currently a lack of research regarding its resilience against DDoS.

The benefits of HTTP/2 has been widely covered by existing works, however the question that has not been answered is its potential risks to DDoS. Unfortunately, designing increased functionality for benign users, may also provide a larger attack surface to the threat actors.

This paper explores the vulnerability of the HTTP/2 protocol against DDoS flood attacks. First we provide an overview of the protocol features as many in the research community are not aware of these major changes. After this, we investigate the popular well known request flood attack vector, to understand how an attacker may carry out this attack using HTTP/2.

We contrast on the potency of a request flood using HTTP/2 compared to the previous HTTP/1.1 standard.

II. HTTP/2 Protocol Overview

Internet connection speeds have significantly increased over the years to meet the demand of the internet, however latency has only marginally improved due to the constraints posed by transferring a signal over long distances. In practice, the previous versions of HTTP only allowed a single active HTTP request to be active, with each request requiring its own individual packet. These factors resulted in a slow connection due to round trip times. Google initially tried to address this issue by deploying their own version of the HTTP protocol called SPDY onto their web servers and Chrome browsers. After several years of fine tuning, SPDY was formally adopted by the HTTP working group and was published as an open standard in 2015[4]. HTTP/2 pushes out new features to reduce page load times by reducing the number of packets required by allowing multiplexed streams and by allowing the server to push content to the client before it is requested. Header compression has also been introduced to significantly reduce the levels of duplicate header data being sent in a single connection. The updates implemented by HTTP/2 will now be discussed in-depth so that the risks to DDoS can be exposed.

A. HTTP/2 Headers

Since the introduction of HTTP, header sizes have significantly increased. Relatively large cookie and user agent values are now sent with each request. These values are typically the same for subsequent request resulting in large amounts of data being duplicated throughout the connection. HTTP/2 aims to solve this by introducing its own bespoke header compression format, HPACK[5]. HPACK introduces memory tables to store frequent data so that requests can reference the header values from these memory tables. When header values are sent, they are also Huffman Encoded to reduce their transmission size.

HPACK introduces two memory tables, the static and dynamic table. The static table is defined by the RFC, and is predefined with common values such as header names and response codes. The dynamic table however is volatile and is initialised and filled during each connection. A dynamic table is created independently by each party. When a header field value is first sent, the client can notify the server to store it in its dynamic table. Subsequent requests by the client can reference the value in the table instead of repeatedly transmitting the value in full. HTTP header fields can be stored in this table, such as frequently recurring values such as the domain name, user agent and cookie fields.

B. HTTP/2 Server Push

The protocol also enables new functionality to allow the server to initiate flows to the client. It has a Server Push feature allowing for the server to send a file to the client before it is requested. It is common that a certain file is always required on a website for example a style sheet. As the server knows the client will eventually request this, it can reduce the load time by sending it to the client in advance to provide the user with a quicker load time.

C. HTTP/2 Stream Control

Historically HTTP requests were sent sequentially in a queue and each request requiring it's own individual packet. HTTP/2 implements flow control in the application layer allowing multiplexing within the HTTP frame. This flow control has similar functionality to TCP, resulting in the possibility of TCP becoming redundant in the future [6]. Both HTTP requests and responses are multiplexed in HTTP/2 as can be seen in Figure 1. The multiplexing of outgoing requests is not a completely new concept as pipelining was introduced in HTTP/1.1 however it was never enabled on modern browsers due to performance issues.

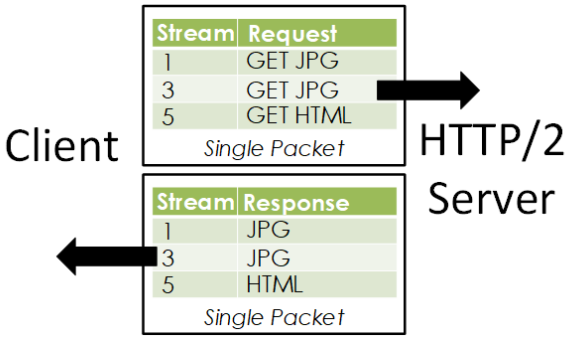

Fig. 1. HTTP/2 Streams

By default, each HTTP/2 connection has a maximum concurrent stream value, this puts a limit on the number of streams which can be opened at a point in time. The RFC recommends a minimum of 100 however each HTTP/2 implementation can decide the exact value. In current implementations, the current chosen values are 100 for Apache and nghttp2, whilst Nginx uses 128. If a value is not defined by the clients, the initial value is assumed to be infinite. This allows the maximum number

of concurrent requests to be sent from a client simultaneously, on a single TCP connection and potentially all within a single packet. This gives benefits to the user due to less packets being required, therefore less packet overheads and an overall reduced round trip time.

If a client has more requests to send than the maximum concurrent thread limit, they initially will only be able to send the amount specified by the limit, however they can send the remaining requests once the initial streams are closed.

Each request is also given a priority weighting which is implemented using dependency trees, allowing for the client to control the scheduling of responses. Flow control is also enabled for the management of the bandwidth rates. Window sizes can be set either for the entire connection or per stream. The window sizes are controlled using window size updates which are set by a HTTP/2 setting frame. Each Stream is given a unique ID, starting at Stream 0. Setting Frames are always transmitted using stream 0, with the incremented streams being divided between the client and server. The client has control of the odd numbered streams, whereas the server uses the even numbered streams.

D. HTTP/2 Potential Vulnerabilities

Historically, multiple DDoS attack methods have been discovered for targeting a HTTP/1.1 web server. Many of these existing attacks are still applicable with the new protocol, however a sophisticated attacker carrying out these attacks may be able to obtain a larger attack power using this new protocol. Due to this risk, the RFC contains a brief section dedicated to some of the issues the protocol introduces in relation to DDoS [4].

New attacks have also been enabled by this new protocol version. HTTP/2 manages flows using priorities and window updates. An attacker can exploit these newly introduced flow mechanisms to cause DDoS. If an attacker floods the web server with flow window updates, the web server will consume processor power trying to manage the flow scheduler, which can cause the server to be unable to respond to genuine requests

HTTP Flood has been a major DDoS attack vector. This attack aims to flood the server with application requests, causing the server to be unable to respond to genuine requests due to being overloaded. With HTTP/2 multiplexing, an attacker will be able to send simultaneous requests on the same connection, hence giving the attacker a greater attack power, as with a single packet they can now issue magnitudes more requests instead of just a single request.

III. Related Work

E Damon et al [7] analysed slow attacks against HTTP/1.1 servers by consuming all of the available server resources. These attacks are sophisticated requiring knowledge by the attacker on the inner workings of the server thread controller.

S Ranjan et al [8] investigated the HTTP Flood attack were attackers flooded the server with requests to overwhelm the processing power of the server. They investigated HTTP/1.1 attacks including request flooding, asymmetric workload and repeat one shot attacks.

E Adi et al[9] [10] were the first to analyse the implications of DDoS on HTTP/2. They looked at the flow mechanisms introduced to the application layer and demonstrated attacks relating to the abuse of window sizes to overload the web server due to the management of flows.

Researchers at the cyber security vendor Imperva, released a white paper in August 2016 analysing four flaws with HTTP/2[11] which were a mix between bug flaws in web server software and flaws with the protocol itself. Their first flaw was with Microsoft IIS's implementation of stream multiplexing, their second flaw was implementation issue on all major web servers when an attacker implemented slow read, this attack was applicable to HTTP/1.1 however using HTTP/2 they required less TCP connections, their third flaw was against the HTTP/2 protocol's priority and dependency controls by creating a dependency cycle loop and their last flaw was a implementation issue of HPACK, which they exploited by creating a HPACK header compression bomb which when decompressed consumed all of the web servers memory.

IV. Rationale for Current Research

Extensive research has been carried out on previous HTTP versions however currently, the only published work directly relating to HTTP/2 DDoS has been performed by E Adi et al[9][10] and the security vendor Imperva [11] and do not provide a complete coverage of potential attack vectors.

The changes in the new protocol dramatically change the overall operation of the protocol and has altered the threat posed by DDoS. There is currently no research into the risks posed by existing flood attacks when performed using HTTP/2.

Request flood attacks are a major threat, and are a frequent attack vector, however due to the multiplexing of requests, this allows more requests to be sent simultaneously to a victim.

From obtaining knowledge of this attack, before it is seen by industry, this paper aims to provide the field with a threat model so that the security risk can be mitigated in future defence proposals. This will allow for defences to keep ahead of the attack tools currently used by attackers and also allow for companies to decide if the benefits of HTTP/2 outweigh the potential DDoS risk.

V. Request Flood Attack

A website can be taken offline if the processing power of the web server can be consumed by malicious requests. One method of doing this is using a HTTP Flood attack, it does this by issuing thousands of bogus requests. As the requests are received at the web server, this consumes the applications resources by requiring large CPU power to generate responses. There are also large overheads required for handling the incoming connection states which can consume up the server's available threads.

HTTP Flood attacks have existed with the previous version of HTTP, however we question if the latest version, HTTP/2 will increase the potency of this attack due to the introduction of concurrent streams as can be seen in Figure 2. This risk has not been looked at by existing research.

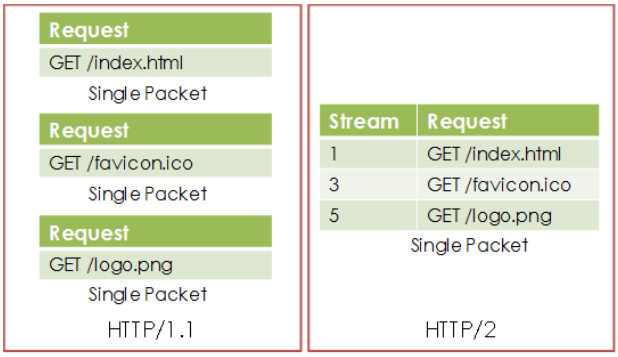

Fig. 2. HTTP/1 and HTTP/2 Flood Attacks

To understand this risk, we initially performed the historical HTTP/1.1 attacks to understand where the attack strength bottleneck was located. This provided us with increased understanding on the limitations of the current DDoS attacks. We subsequently preformed the same attack using HTTP/2 to demonstrate how the attack would perform and if the new functionality of the protocol would provide an attacker with a larger attack surface whilst using the same attack resources.

To perform the experiments without causing harm to live networks, a testbed was instead deployed comprising of web servers running HTTP/1.1 and HTTP/2 in their default configurations with no proxy or load balancing. Extrinsic throughput measurements were taken and analysed to provide the necessary metrics to show the damage caused by each protocol version.

A. Testbed

The test bed composed of two Ubuntu Linux hosts. The attack node composed of a physical computer comprised of a Intel i7-4770 CPU, 3.40GH, 4 cores, 8 threads with 12GB Ram whilst the virtual server was configured on a Xen Server, Intel Xeon E5-2637 3.5Ghz, with 8 threads and 4GB of main memory allocated. The monitored traffic generation was the egress traffic from the physical desktop.

The web server used for all flood simulations was nghttp2 v1.10.1. The hosts are linked together using a Cisco c3750x with a 1Gbit Ethernet link. The hosts links are speed tested to prove they can transmit 1GBit/sec between them.

B. Flood Attack Methodology

An attack tool was created to generate the attack. Firstly, the script sent HTTP/1.1 requests repeatedly to the web server. The attack software was run using the physical desktop utilising 800 simultaneous connections each running on an independent thread. Using multiple connections allows for the link to be utilised to it's maximum. Varying numbers of connections were attempted to determine the optimum quantity of 800.

The aim of the attack was to generate the maximum number of requests to the web server, therefore the attack tool did not parse the web server responses.

The attack was repeated, but modified to utilise the HTTP/2 protocol. Due to HTTP/2 Multiplexing, each TCP packet now comprised of many simultaneous streams dependent on the max concurrent value set by the server. All subsequent requests are also able to reference the first requests header values reducing the bandwidth per request. An initial Settings Frame and window update frame were sent to initialise the channel before the subsequent streams were transmitted as can be seen in Figure 3.

The RFC recommends a minimum max concurrent value of 100 however can be altered by the server. To give a large range of results, this value was set to 100, 128, 256 and 512 for the attack simulations in order to provide full visibility on how the max concurrent values affect the risk from DDoS.

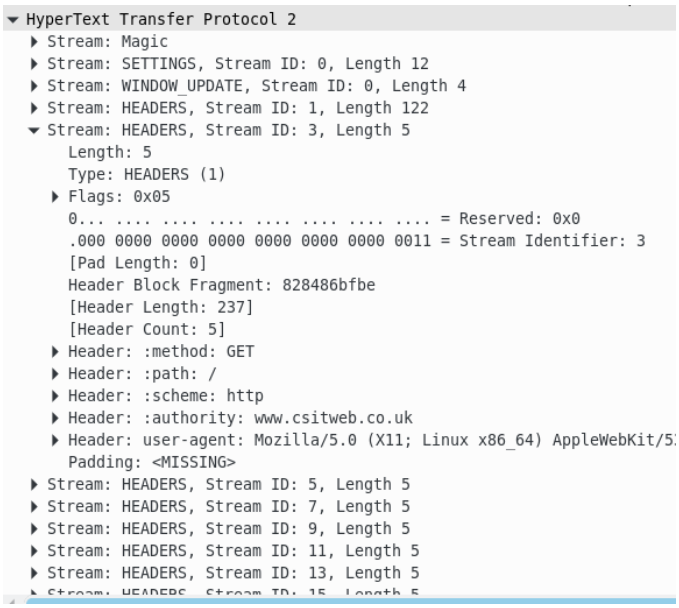

Fig. 3. HTTP/2 Packet Format captured by wireshark

C. Flood Attack Results

During the HTTP/1 attack, an average of 15,975 Requests/Second were sent to the server. When the traffic was analysed to obtain the location of the bottleneck, it was clear that it was not the network link capacity as it was not saturated. The outgoing attack traffic only consumed 46.8Mb/s of throughput of the available 1Gbit/s link, the bottleneck however was found to be the packet generation rate of the device. It was only able to generate 0.016M outgoing Packets/Sec, due to the kernel overheads required to generate each data packet. This can be seen in Figure 4.

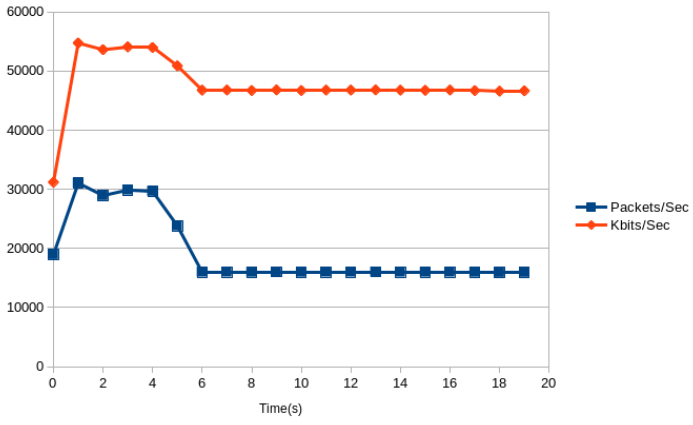

Fig. 4. Packets and KBits per second during HTTP/1 Attack

TABLE I
POSSIBLE REQUEST RATES

Protocol	Max Concurrent	Req/Sec	Relative to HTTP/1
HTTP1	N/A	15,975	N/A
HTTP2	100	914,706	57.26x
HTTP2	128	995,670	62.33x
HTTP2	256	1,414,859	88.57x
HTTP2	512	1,525,272	95.48x

Each HTTP/1 packet generated was small resulting in the small utilisation of the bandwidth however it was the generation of the unique TCP header and kernel utilisation which limited the system during traffic generation. Modern network card have inbuilt offload capability to offload part of this requirement, however this was enabled for this experiment to minimise the bottleneck.

This results demonstrates that even if an attacker can get an attacker host with large bandwidth, there will be little advantage for a HTTP/1 Flood attack. The limitation is the processing power and network card offload capability to generate large quantities of packets.

When the flood attack was repeated using HTTP/2, the attack computer was able to send more requests. The results in Table I show the larger attack surface for HTTP/2. The HTTP/2 attack bottleneck was again packet generation, however due to multiplexing, the attack had increased request generation as each packet could hold multiple streams. This allowed for an attacker on the same hardware, to carry out a larger attack with the same packet generation limitation.

For the base max concurrent user value of 100, the attacker had a 57 times larger attack power, this effectively gave the attack computer running HTTP/2 the attack power of a 57 computer botnet running HTTP/1.1. This attack power increased as the server's max concurrent stream limit was increased as can be seen in the results table.

VI. DISCUSSIONS AND CONCLUSION

The objective of the research is to understand, in a HTTP/2 deployment, the risk of a DDoS attack relative to the threat posed by the previous version. For HTTP flood attacks, it has been demonstrated that the bottleneck for an attacker, is packet generation. Due to HTTP/2 allowing multiple requests per packet, this gives the attacker a major advantage and results in an attack with increased potency.

Larger max concurrent values do give better QoS to benign users as it allows more streams to be used simultaneously, however it can clearly be seen in the

results that it also gives an attacker a larger attack surface for flood attacks. The RFC specifies a minimum limit of 100 with current implementations use 100 or 128 limits. With the recommended minimum max concurrent stream value of 100, an attacker will leverage 57.3x more flood requests compared to the previous protocol version with our attack script. Presently Nginx uses a higher value of 128, allowing for 62.3x more requests than HTTP/1.1. It is therefore recommended for this value to be kept at a lower level to minimise the risk posed by DDoS Flood attacks.

We believe we have benefited the research community by providing this information so that suitable defence mechanisms can be proposed. We hope to explore suitable detection strategies in our future work and analyse further attack methods relevant to HTTP/2. HTTP/2 deployment is on the rise, however security research on this new protocol is inadequate, we have provided much needed research on this protocol so that future deployments can be implemented with security from DDoS. We end with a question to the technical community of whether the speed improvements of HTTP/2 are worth exposing the web server to a larger attack surface, before suitable detection methods are created and deployed.

REFERENCES

[1] A. Networks, "WORLDWIDE INFRASTRUCTURE SECURITY REPORT XII 2017," 2017. [Online]. Available: https://www.arbornetworks.com/insight-into-the-global-threat-landscape

[2] Akamai, "State of the Internet, Q2 2015," no. 1, 2015.

[3] Cloudflare, "Say Cheese: a snapshot of the massive DDoS attacks coming from IoT cameras," https://blog.cloudflare.com/say-cheese-a-snapshot-of-the-massive-ddos-attacks-coming-from-iot-cameras, 2016, online; accessed 10-Mar-2017.

[4] IETF, "RFC:7540, Hypertext Transfer Protocol Version 2 (HTTP/2)," 2015. [Online]. Available: https://tools.ietf.org/html/rfc7540

[5] IETF., "RFC:7541, HPACK: Header Compression for HTTP/2)," 2015. [Online]. Available: https://tools.ietf.org/html/rfc7541

[6] IETF, "RFC Draft, QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2)," 2015. [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00

[7] E. Damon, J. Dale, E. Laron, J. Mache, N. Land, and R. Weiss, "Hands-on denial of service lab exercises using slowloris and rudy," in *Proceedings of the 2012 Information Security Curriculum Development Conference*, ser. InfoSecCD '12, 2012, pp. 21–29.

[8] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly, "Ddos-resilient scheduling to counter application layer attacks under imperfect detection," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, April 2006, pp. 1–13.

[9] E. Adi, Z. A. Baig, P. Hingston, and C.-P. Lam, "Distributed denial-of-service attacks against http/2 services," *Cluster Computing*, vol. 19, no. 1, pp. 79–86, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10586-015-0528-7

[10] E. Adi, Z. Baig, C. P. Lam, and P. Hingston, "Low-rate denial-of-service attacks against http/2 services," in *2015 5th International Conference on IT Convergence and Security (ICITCS)*, Aug 2015, pp. 1–5.

[11] Imperva, "HTTP/2: In-depth analysis of the top four flaws of the next generation protocol," Aug, 2016.