***Abstract – When initially implemented, Hypertext Transfer Protocol (HTTP) was designed to mostly pass text-based information. Due to the increase of demand for more robust transfers of data, the second generation of HTTP was implemented in 2015. HTTP/2 solved many of the issues that HTTP 1.1 suffered from and was adopted by most major browsers by the end of the year. By implementing items such as binary framing and multiplexing, the overall connection speed and latency were far improved. With this increase in speed however, came an increase in the capabilities of Distributed Denial of Service (DDoS) attack speeds as well. Extensive research has been completed on HTTP/2, helping to mitigate many of the attack vectors threat actors take. Yet despite those advances and different mitigation factors implemented, beginning in August of 2023, AWS, Google, and Cloudflare all reported DDoS attacks in excess of 201 million requests per second (RPS): Nearly triple the previous record of 71 million RPS. This was achieved through a zero-day vulnerability now titled a rapid-reset attack and leverages the way HTTP/2 functions. This study begins by discussing the different features of HTTP/2 and how it differs from its’ processor, then tests the baseline binary frames to establish the increase in traffic each causes. It ends by examining potential other vulnerabilities within the protocol to build upon in the future.***

***Index Terms – HTTP/2, Distributed Denial of Service, DDoS, Rapid Reset***

1. INTRODUCTION

Originally released in 1997 [1], the Hypertext Transfer Protocol (HTTP) 1.1 was released and put into practice. This protocol was text-based and required multiple Transmission Control Protocol (TCP) connections to load a single website [2]. As the internet continued to expand, with more information being passed back and forth between an end user and the web server, it HTTP/1.1 began to slow the connectivity between the two. HTTP/1.1 had added request pipelining, but that did not fully address the issues of request concurrency and had an issue with head-of-line blocking [3]. Due to this, in May of 2015 HTTP/2 was proposed.

HTTP/2 addressed the issues by introducing binary framing and multiplexing, allowing a single TCP connection to a web server to yield everything the browser needed, including any Cascading Style Sheets (CSS) and JavaScript (.js) files in addition to the HTML base. Section three of this study goes into the differences in more detail.

After the initial release and adoption of HTTP/2, extensive research was conducted on its’ ability to defend against DDoS attacks. Then, in the eight years following the research began to taper as HTTP/3 was released. Yet despite all the advancements in protections, a new zero-day vulnerability within the HTTP/2 protocol itself was discovered and leveraged beginning in August of 2023.

Between the dates of August and October of 2023, Amazon Web Services (AWS) [4], Google [5], and Cloudflare [6], all reported a new type of Distributed Denial of Service (DDoS) attack that far exceeded previous Request per Second (RPS) records. To that point, the previous record of RPS against a service was 71 million. However, leveraging this new vulnerability, AWS reported RPS of 155 million, Cloudflare reported RPS that exceeded 201 million, and the RPS topped out with Google reporting RPS exceeding 398 million.

The attackers were able to do this by leveraging a feature in HTTP/2 known as stream cancelation. By leveraging a simple, “request, cancel, request, cancel” cycle at scale, the threat actors were able to create massive requests per second with minimal number of endpoints [7]. Cloudflare noted that the new record-breaking attack took only approximately 20,000 machines to be successful. To put that in context, the Federal Bureau of Investigation (FBI), took down a botnet in August of 2023, known as Qakbot, that had over 700,000 botnets worldwide, with more than 200,000 of them on United States soil. [8][9].

While HTTP/3, the next generation of protocol, began development in 2018 [10], HTTP/2 is still the most widely implemented in the world. Given this new vulnerability being identified, this study revisits HTTP/2’s protocols to test other potential vulnerabilities to DDoS attacks. It begins by exploring related works, then details the differences between HTTP/1.1 and HTTP/2, taking a deep dive into potential attack vectors within the protocol. It then tests against publicly available HTTP/2 test servers, draws conclusions and builds future works.

1. RELATED WORKS

After implementation of the protocol, several research projects tested the viability and defense of it against DDoS attacks. The first study tested four major parameters: CPU usage, memory consumption, network throughput, and packet loss [11]. The purpose of this study was to examine concerns with what is known as flash traffic. This is an unintentional attack on a web servers resources when an unexpected amount of traffic is naturally generated by a large amount of individuals accessing the resource simultaneously. Most notably, this happened in October of 2013 with the launch of HealthCare.gov, the US federal website that was to enroll individuals in a special healthcare plan for those in need [12]. The research found that by leveraging the window\_size\_increment and window\_update binary frames, they were able to cause an unnecessary use of resources. It also found that leveraging a value of 2*14,* 16,384 as the window size, they were able to successfully DDoS a server with approximately 100,000 and 1,000,000 frame packets. Ultimately, the intention behind the research was to avoid detection, and found that the protocol itself does not restrict intensity of traffic generated.

Another study also leveraged the stealthy attack model [13]. This study attempted to evade detection by Intrusion Detection System (IDS), by generating traffic that was indistinguishable from normal traffic. Leveraging the false negative, the percentage of traffic that a machine learning algorithm incorrectly identified as legitimate traffic, they were able to create traffic that would not be blocked. The research built 2 attack models, one that mimicked the size\_rstAck feature and one that mimicked the count\_syn feature values. While utilizing those two models, four different ML algorithms were trained: Naïve Bayes, Decision Tree, JRip, and Support Vector Machine (SVM). Between the two techniques sued, the stealthy model built to take advantage of the count\_syn feature values was able to have false negative percentages as high as 26.51%, while the test using the size\_rstAck were able to evade detection more than 50% of the time in a Naïve Bayes model.

Other studies were conducted using event sequence analysis in attempt to leverage “slow” HTTP/2 DoS attacks [14] and other threat vectors [15]. These studies both leveraged specially crafted frames to once again evade detection. The target servers were: Apache 2.4.23, Nginx 1.10.1, H20 2.0.4 and the Nghttp2 server. Using varying frames, the studies tested varying rates of waiting time and different frames. It was found that whether the traffic was encrypted or decrypted, each of the servers tested were vulnerable to the attacks.

Each of the studies listed attempted to evade detection through varying methods. However, with the recently found Rapid Reset attack, stealth was not the intention. Rather it was to generate the most traffic possible with minimal effort. Because of the simplicity of the attack, this study will test and set a baseline of degradation from each of the binary frames introduced in section three to understand the load each carries against a web server. Then it will test the rapid reset vulnerability and finally try to pair other binary frames together to find if the pairings create a similar amount of congestion.

1. HTTP

The HTTP protocol builds the basis for most of the internet connectivity from an endpoint to a webserver worldwide. As the amount of devices accessing the webservers grew, paired with the newer rich-formatting of videos, images, and streaming, the protocol needed to be updated.

The original protocol, HTTP/1.0, only allowed a single request to be outstanding on a TCP connection at one time [16]. A year later, HTTP/1.1 was released, providing several enhancements. Namely, it made four major changes [17]:

* Header: HTTP/1.1 requires a host header, helping to route messages through proxy servers and assisting in identifying different domains that point to the same internet protocol (IP) address.
* Persistent connections: With HTTP/1.0, each request/response pairing required opening a new connection. With HTTP/1.1, several requests can use a single one.
* A white background with black text

  Description automatically generated with medium confidenceContinue status: Clients can send a request header and only connect if given a continue status.
* Additional Request Methods: PUT, PATCH, DELETE, CONNECT, TRACE, and OPTIONS

The issues were then remedied slightly with the implementation of HTTP/1.1, however despite the pipelining implementation, end points were still having to make multiple TCP connections to a web server to load a single page. The process can be seen in figure 1.

A screenshot of a computer program

Description automatically generated

Fig. 1. Established TCP connections in HTTP/1.1 [2]

HTTP/2.0 remedied this by allowing end points to download the varying resources asynchronously from the web server in question. This parallelization decreased overall turnaround time when accessing a server and can be seen in figure 2.

Fig. 2. Established TCP connections in HTTP/2.0 [2]

A screenshot of a computer

Description automatically generated

HTTP/2 also implemented header compression, to shrink a large number of redundant header frames. Both the end point and the server maintain a list of headers used in previous connection requests. All frames passed via HTTP/2 begin with a fixed 9-octet header followed by a variable length payload [16]. The layout can be seen in figure 3.

Fig. 3. HTTP/2 frame breakdown [16]

HTTP/2 uses these frames and then converts them to binary to avoid any confusion. The list of frames available are:

* Connection Preface: Connection Preface (contains a magic string PRI \* HTTP/2.0\r\n\r\nSM\r\n\r\n) is used to establish the initial settings for a HTTP/2 connection and final confirmation of the protocol in use.
* HEADERS and CONTINUATION Frames: HEADERS frame is used to carry a header block. In case a header block is large enough not to fit in a single HEADERS frame, CONTINUATION frames are used to transmit remaining parts of header block.
* DATA Frame: This frame is used to carry message body sent by the endpoints. For example, client uses DATA frame to carry message body of a POST request while server uses DATA frame to carry response to a client's GET request.
* SETTINGS Frame: This frame is used by endpoints for the purpose of negotiating connection parameters like initial window size, maximum concurrent streams, etc.
* WINDOW\_UPDATE Frame: WINDOW\_UPDATE frame is used to indicate the number of bytes that the sender is willing to accept by its peer in addition to the existing HTTP flow-control window.
* GOAWAY Frame: This frame is used either to tear off an established connection between endpoints or to indicate some serious error condition. RFC 7540 defines various error codes that GOAWAY frame carries to convey the reason behind connection or stream error. Some of the error codes are:
  + NO\_ERROR: This code indicates graceful shutdown of a connection.
  + PROTOCOL\_ERROR: This code indicates an unspecific protocol error. This code is usually transmitted when there is no more specific code available.
  + INTERNAL\_ERROR: This code is transmitted in case an endpoint encounters an unexpected break in the normal working of protocol.
  + FLOW\_CONTROL\_ERROR: An endpoint transmits this code if peer violates the flow control negotiations made.
  + SETTINGS\_TIMEOUT: In case an endpoint does not receive acknowledgement of a SETTINGS frame sent by it, it sends a GOAWAY frame with SETTINGS\_TIMEOUT code.
  + FRAME\_SIZE\_ERROR: An endpoint transmits this code if it receives a frame with an invalid size.

Finally, the major feature that was implemented with HTTP/2 was the concept of streams and multiplexing. A stream is considered an independent, bidirectional sequence of frames that pass between client and server. Streams have the following characteristics:

* A single HTTP/2 connection can contain multiple concurrently open streams.
* Streams can be used unilaterally or shared between end point and server
* Streams can be closed by either party
* The order in which frames are sent on the stream is significant as the recipients process the frames in the order received.
* Streams are identified by an integer and assigned by the endpoint initiating the stream.

These streams also have a specific lifecycle. Once a PUSH\_PROMISE frame is sent, with implied continuation, or if a HEADERS frame is sent with implied continuation, the connection between is opened and the stream is reserved. Once an END\_STREAM flag is sent or received, the connection is then half-closed or open. Finally, the END\_STREAM flag has to be sent, if received, or received, if sent, or a RST\_STREAM frame is sent, to close the connection. The lifecycle can be seen below in figure 4.

A diagram of a block diagram

Description automatically generated

Fig. 4. HTTP/2 stream lifecycle [16]

The concept of streams becomes significant, as that is was caused the sizable increase in traffic with the recently observed rapid reset attack. HTTP/2 has implemented a maximum number of concurrent streams that a single endpoint can open [7]. This is built as a parameter and is listed by the web server. If an endpoint attempts to open another stream above the limit set by the server, it is rejected by the server sending a RST\_STREAM frame.

The weakness observed with the rapid reset attack is that only connections with a state of open or half-closed, as seen in figure 4, count towards the maximum number of streams. Recognizing this, the rapid reset attack is successful by constantly opening streams, then forcefully closing them. While this is a useful feature in HTTP/2, for instance if a web page only needs to load images that are being viewed at the time, it also forces the stream through the entire lifecycle prior to being closed.

In the current landscape, it is common to see a deployment architecture that involves load-balancers in front of other items within the deployment. Requests are quickly handled, but the actual work is handled elsewhere. But when such a deployment is implemented, it can become difficult for the load-balancer to clean up the requests [7]. This disconnect is then leveraged in the rapid reset attack to quickly fill the buffer and take down the servers upstream.

1. ENVIRONMENT

The environment for the experimental analysis of HTTP/2, focusing on performance and vulnerability to DDoS attacks, was meticulously set up to ensure precise and accurate results. The expanded details of this environment are as follows:

* Development Platform:
  + Jupyter Notebook: Used for its versatility in combining live code, visualizations, and narrative text. It allowed for an interactive and iterative approach to running experiments and analyzing results on the fly.
  + Version Control and Documentation: Jupyter Notebook facilitated version control and extensive documentation, ensuring that each experiment step was recorded and reproducible.
* Programming Language and Libraries:
  + Python: Chosen for its widespread use in data science and networking, Python's robust ecosystem and readability made it an ideal choice for conducting complex network experiments
  + httpx: A powerful HTTP client for Python, httpx was instrumental in handling HTTP/2 requests, enabling the simulation of various HTTP/2 interactions with the test server.
  + SSL Module: Used to establish secure connections over the internet, ensuring that all communications during the experiments were encrypted and secure.
  + asyncio: This library allowed for efficient asynchronous programming, crucial for simulating high-traffic network conditions and handling multiple simultaneous HTTP/2 connections.
  + pandas and matplotlib: Essential for data analysis and visualization, these libraries were used to process the experimental data and present it in a comprehensible format.
  + h2 Library: A pure-Python HTTP/2 protocol stack, critical for in-depth manipulation and analysis of HTTP/2 protocol features.
  + Test Server nghttp2.org: A well-known server with comprehensive HTTP/2 support, used for its reliability and to ensure that the results were reflective of real-world HTTP/2 performance.

1. EXPERIMENT

The experimental setup, designed to comprehensively test the HTTP/2 protocol, encompassed several distinct but interconnected components to evaluate the protocol's performance, resilience, and security. Each aspect of the experiment is elaborated below:

* Feature Testing:
  + Stream Multiplexing: This test involved initiating multiple HTTP/2 streams concurrently to assess the protocol's capability to handle numerous requests over a single TCP connection, a fundamental feature distinguishing HTTP/2 from its predecessor.
  + Header Compression: HTTP/2's header compression mechanism, HPACK, was tested for its efficiency in reducing overhead. This involved comparing the size of headers before and after compression.
  + Data Streaming: The protocol's ability to stream large amounts of data efficiently was evaluated. This included testing server push capabilities and monitoring how well the server and client managed varying flow-control windows.
* DDoS Attack Simulation:
  + High Traffic Simulation: Simulated a high-traffic environment by sending hundreds of requests simultaneously to the test server, thereby assessing its capacity to handle heavy loads, which is crucial in understanding the protocol's vulnerability to DDoS attacks.
  + Rapid Reset Attack: Specific focus was given to the rapid reset attack, where numerous HTTP/2 streams were rapidly created and then reset, testing the server's ability to handle such malicious patterns without significant degradation in performance or stability.
* Frame-Level Testing:
  + Individual Frame Testing: Each type of HTTP/2 frame was tested individually, focusing on their specific roles and impacts. This included sending custom headers, data, settings, window updates, and GOAWAY frames.
  + Combined Frame Testing: Various combinations of frames were sent in sequences to observe the interplay and potential conflicts or efficiencies that arise when different frame types are used in conjunction.
  + Performance Metrics: Key performance metrics such as response time, data throughput, and server load were measured and analyzed for each type of frame and their combinations.
* Security and Vulnerability Assessment:
  + Common Vulnerabilities: The server was scanned for typical web vulnerabilities like exposed .git directories, configuration files, or unprotected admin interfaces, which are often targeted in web-based attacks.
  + SSL/TLS Configurations: The experiment included validation of SSL/TLS configurations, ensuring that the communications were encrypted and secure, and checked for vulnerabilities like weak ciphers or outdated protocols.
  + Headers Security: The security of HTTP/2 headers was also scrutinized, checking for headers that might reveal sensitive information or could be exploited in attacks.
* Methodology and Execution:
  + Asynchronous Programming: A key aspect of the methodology was the use of asynchronous programming to mimic real-world web traffic and server-client interactions more accurately.
  + Error Handling and Logging: Robust error handling and logging mechanisms were implemented to capture any anomalies or unexpected behaviors during the tests.
  + Data Analysis: Post-experiment, the collected data was systematically analyzed using statistical tools and visualizations to derive meaningful insights and conclusions.

This experimental setup provided a detailed examination of HTTP/2's capabilities and vulnerabilities, offering an understanding of how the protocol performs under various scenarios, ranging from regular operation to high-stress conditions like DDoS attacks.

1. CONCLUSION

Performance under Normal and Stress Conditions:

The HTTP/2 protocol demonstrated robust performance in terms of handling multiple requests, efficient data streaming, and header compression under normal operating conditions.

Under high traffic scenarios, simulating DDoS attack conditions, the protocol showed resilience, but also revealed areas where performance could degrade, particularly in the rapid reset attack simulations.

Rapid Reset Attack Vulnerability:

The experiments highlighted a critical vulnerability in the HTTP/2 protocol - the rapid reset attack. This vulnerability, leveraged by attackers in real-world scenarios as noted in the paper, poses a significant risk to HTTP/2's security framework.

The ease with which attackers could generate high Request Per Second (RPS) rates using minimal resources underscores the need for enhanced security measures and protocol adjustments.

Frame-Level Analysis:

Individual and combined frame tests revealed that while the server could efficiently handle various frame types under normal conditions, specific scenarios (like Window Update handling) resulted in longer response times.

This suggests that while HTTP/2 offers improvements over HTTP/1.1 in terms of efficiency and speed, certain aspects of frame handling could be optimized further.

Security and Vulnerability Assessment:

The security checks for common vulnerabilities and SSL/TLS configurations reinforced the importance of robust security practices in HTTP/2 implementations.

These assessments serve as a reminder that while HTTP/2 offers advanced features, it is not immune to traditional web vulnerabilities and requires diligent security measures.

Alignment with Research Objectives:

The study successfully revisited HTTP/2’s protocols, as proposed in the introduction, highlighting both its strengths and potential vulnerabilities to DDoS attacks.

The exploration of different aspects of the protocol, from basic functionalities to specific attack vectors, aligns with the initial goal of understanding and improving HTTP/2's resilience against emerging threats.

In summary, while HTTP/2 marks a significant improvement over its predecessor in terms of efficiency and performance, this study brings to light critical areas where the protocol can be vulnerable to sophisticated DDoS attacks. The rapid reset attack, in particular, represents a novel challenge that needs to be addressed to bolster HTTP/2's defense mechanisms against such high-scale attacks. This research contributes valuable insights to the field, laying a foundation for future enhancements in HTTP/2 security and performance.

1. DRAWBACKS AND FUTURE WORK

For the purposes of this research, only one web server was utilized. To truly understand how the rapid reset and other HTTP/2 specific attacks work at scale, an architecture akin to a content delivery network (CDN) should be utilized, potentially with a load balancer distributing the traffic. It was due to the CDN and load balancing that most of Google, AWS, and Cloudflare’s clientele noticed minimal disruption, despite the scale of the attack.

For further testing, additional combinations of the binary frames should be leveraged. While the rapid reset attack was immediately creating and tearing down connections to bypass the maximum limit of open streams, another couplet of frames or potentially triplets could cause additional harm. The server utilized for this test was also publicly available without an IDS implemented. Adding security measures and testing their viability against HTTP/2 attacks such as rapid reset should also be done.

Finally, an examination of the scalability of the rapid reset attack could also be conducted. As previously mentioned, the attacks recorded only involved an estimated botnet of approximately 20,000 endpoints. Recognizing that botnets can grow significantly larger than that, testing with a growing number of endpoints would be needed to understand the magnitude.

**References**

1. Nielsen, H.F. (Jan 1999) HTTP Timeline <https://www.w3.org/Protocols/Timeline.html>
2. Factory.hr. (July 2018) HTTP/2: The Difference Between HTTP/1.1, benefits and how to use it. <https://factoryhr.medium.com/http-2-the-difference-between-http-1-1-benefits-and-how-to-use-it-38094fa0e95b>
3. Belshe, M. (May 2015). Hypertext Transfer Protocol Version 2(HTTP/2). <https://httpwg.org/specs/rfc7540.html>
4. Haranas, M. (Oct 2023). AWS CEO Confirms Thwarting ‘New Type’ of DDoS Security Attack. <https://www.crn.com/news/cloud/aws-ceo-confirms-thwarting-new-type-of-ddos-security-attack>
5. Snellman, J. & Iamartino, D. (Oct 2023). How it Works: The Novel HTTP/2 ‘Rapid Reset’ DDoS Attack. <https://cloud.google.com/blog/products/identity-security/how-it-works-the-novel-http2-rapid-reset-ddos-attack>
6. Bourzikas, G. (Oct 2023). HTTP/2 Zero-Day Vulnerability Results in Record-Breaking DDoS attacks. <https://blog.cloudflare.com/zero-day-rapid-reset-http2-record-breaking-ddos-attack/>
7. Pardue, L. & Desgats, J. (Oct 2023). HTTP/2 Rapid Reset: Deconstructing the Record-Breaking Attack. https://blog.cloudflare.com/technical-breakdown-http2-rapid-reset-ddos-attack/
8. Fadilpasic, S. (Aug 2023). The FBI has taken down one of the biggest botnets in the world. <https://www.techradar.com/pro/security/the-fbi-has-taken-down-one-of-the-biggest-botnets-in-the-world>
9. FBI (Aug 2023). FBI, Partners Dismantle Qakbot Infrastructure in Multinational Cyber Takedown. <https://www.fbi.gov/news/stories/fbi-partners-dismantle-qakbot-infrastructure-in-multinational-cyber-takedown>
10. Sabron, S. (May, 2023). HTTP/2 vs HTTP/3: Which Protocol Reigns Supreme? <https://medium.com/@siddharth.sabron/http-2-vs-http-3-which-protocol-reigns-supreme-e3db0271d094>
11. Adi, E., Baig, Z.A., Hingston, P. et al. Distributed denial-of-service attacks against HTTP/2 services. Cluster Comput 19, 79–86 (2016). <https://doi-org.aurarialibrary.idm.oclc.org/10.1007/s10586-015-0528-7>
12. Cold Call Podcast (Oct 2016). The Spectacular Fall and Fix of HealthCare.gov. <https://hbswk.hbs.edu/item/the-spectacular-fall-and-fix-of-healthcare-gov>
13. Erwin Adi, Zubair Baig, Philip Hingston, Stealthy Denial of Service (DoS) attack modelling and detection for HTTP/2 services, Journal of Network and Computer Applications, Volume 91, 2017, Pages 1-13, ISSN 1084-8045, <https://doi.org/10.1016/j.jnca.2017.04.015>.
14. Tripathi, Nikhil. “Delays have Dangerous Ends : Slow HTTP/2 DoS attacks into the Wild and their Real-Time Detection using Event Sequence Analysis.” IEEE Transactions on Dependable and Secure Computing, 2023, pp. 1–13, <https://doi.org/10.1109/TDSC.2023.3276062>.
15. Nikhil Tripathi, Neminath Hubballi, Slow rate denial of service attacks against HTTP/2 and detection, Computers & Security, Volume 72, 2018, Pages 255-272, ISSN 0167-4048, <https://doi.org/10.1016/j.cose.2017.09.009>.
16. Belshe, M. (May 2015). Hypertext Transfer Protocol Version 2 (HTTP/2). <https://www.rfc-editor.org/rfc/rfc7540.html>
17. Fulber-Garcia, V. (Nov 2022). HTTP: 1.0 vs 1.1 vs 2.0 vs 3.0. <https://www.baeldung.com/cs/http-versions>