DOI: 10.15276/hait.03.2020.5
UDK 004.75

Comparison of authorization protocols for large requests in the operation queue environment

Sergii S. Surkov
Odessa National Polytechnic University, Odessa, Ukraine
ORCID: http://orcid.org/0000-0001-9224-7526

ABSTRACT

Authorization protocols play a foundation role in web security. There is a risk that the data may be changed in places where it is transmitted in the unencrypted form, for example, in a proxy server. The vulnerabilities of payload spoofing aren't thoroughly researched. Existing authorization protocols for large payload implement the "Filling the Buffer" method and its subtype "Buffering to File". Through simulation, it was found that the second subtype "Buffering to Memory" is not ideal for requests with large payloads. In previous papers, a "chunking" method was developed for authorizing HTTP requests with efficient payload verification. However, it was found that in an operation queue environment, the system is prone to falling into the critical mode, which became a subject of further study. This paper aims to reduce the harmful effect of critical modes. Analysis of signing methods for authorization using a parallel architecture based on queuing theory has shown that the "chunking" method is the most promising. The developed methods for ranking authorization protocols for large requests and identifying critical modes made it possible to experimentally determine the conditions for the manifestation of the advantages of the studied methods of signing the payload in different modes and to study the effect of the intensity of the incoming data on the transition of the system to the critical mode. Conducting a computer experiment, the dependencies of the multithreaded write speed on the number of threads for the "chunking" and "buffering to file" methods were obtained depending on the number of threads and the data transfer rate. The parallel processing of the digital signatures of requests has improved the performance of the system, keeping the sequential processing of data. The study of the influence of the intensity of the incoming data on the transition of the system to the critical mode makes it possible to calculate the limitation of the system load. Thus, the goal of reducing the harmful effect of critical modes and ensuring greater reliability and speed of the system is achieved.

Keywords: digital signature; authorization; large payload; operation queues; network requests; verification

For citation: Surkov S. S. Comparison of authorization protocols for large requests in the operation queue environment. ***Herald of Advanced Information Technology*** **. 2020;Vol.3, No.3: 163–173. DOI: 10.15276/hait.03.2020.5**

INTRODUCTION

Authorization protocols play a foundation role in web security. Modern information systems are characterized by large payload sizes. However, not all protocols guarantee data immutability during transmission [1]. Moreover, for authorizing large payload size, there are no feasible protocols that verify the payload, which is critically important for video streaming, document storage, database services with a complex datacenter infrastructure[2], etc...

LITERATURE REVIEW

Payload integrity problems arise during data transfer. The generally used protocols for communication between clients and servers are the HTTP and HTTP/2 [3–4], which are widely used not only in the web browsers because of their simple and obvious structure. The general solution for maintaining data encryption and integrity is the TLS protocol [5–6].

However, there are cases when data is decrypted during its transfer. The most common case is proxy servers of corporations that replace TLS certificates of the web sites for the user [3; 7–8]. The modern version of the proxy implementation in such corporations is a "transparent" proxy [9].

Also, web services often use "reverse proxy" [10–11], after which the data is transmitted in unencrypted form in the data center. Both of these cases may happen together. Even though proxy servers are considered trusted by users in these situations, they are the most vulnerable point in the system. At these points, the payload of the request may be changed, which makes the system vulnerable to MITM (Man in the Middle) attacks [12–15].

Payload spoofing situations at vulnerable points aren't well researched. Most of the protocols [16–22] only authorize the headers, not the payload of the request itself. The HMAC method [23], which is implemented by many authorization protocols, is used to prevent the modification of the payload. Existing HMAC-based authorization protocols such as OAuth 1.0a [24–26] and HAWK [25; 27–28] are great for verification of small payload sizes. According to our classification, they implement the "Filling the Buffer" method and its subtype "Buffering to Memory". Through simulation, it was found that "Buffering to Memory" is not ideal for requests with large payloads.

A second subtype is the “Buffering to File” method, which is a general solution for handling large payloads. The disadvantage of this method is that the payload is read after uploading again. Due to the caching of files by the operating system into RAM, this drawback is only noticeable when the large payloads are transmitted.

In previous papers, a “chunking” method was developed [29–30] for efficient authorization of payload of HTTP requests [3–4], which seems to be the most promising.

All the described processes are implemented in the operation queues [31–32]. At the same time, the possible transition of the system to a critical mode poses a particular threat, and these issues are little studied. It occurs when operations in the queue accumulate and have no time to be processed. This may cause an instability and termination of the server process.

Thus, the study of critical modes for authorization protocols for large payloads is an actual problem.

THE PURPOSE OF THE ARTICLE

The aim is to research authorization protocols for requests with a large payload, as a result of which is a reduction in the harmful effect of critical modes. In this paper, new methods are developed for ranking authorization protocols for large requests and identifying critical modes based on multi-threaded simulation in an operation queue environment to achieve better reliability and performance.

To achieve the goal, the following tasks have been identified.

1) Analyze the scope, advantages, disadvantages of payload signing methods.

2) Develop methods for ranking payload signing implementations and detecting critical modes.

3) Experimentally determine the conditions of manifestation of the advantages of the studied methods of signing the payload.

4) Investigate the impact of the incoming data rate on the transition of the system to the critical mode.

MAIN PART. THE METHODS OF SIGNING THE PAYLOAD OF THE REQUEST

In our previous papers [29; 33], it was pointed out that implementations of the method “filling buffer do buffer the whole payload of the request to RAM and then authorize it. It works well with small payload sizes, the exact content of which will be neither stored nor used in the future.

Most requests imply that the payload size is not large, so the “RAM buffering” method is used in almost all implementations, which gives an advantage in terms of memory size and relatively small requirements for computational resources.

The second advantage is that it does not require the implementation of complex logic and, therefore, testing is also simpler.

The clear disadvantage of “filling buffer” method is memory allocation requirement:

O(b) = n, where n is the size of the payload of the request, which imposes large memory allocations.

The method of filling the buffer is in *Fig. 1*.

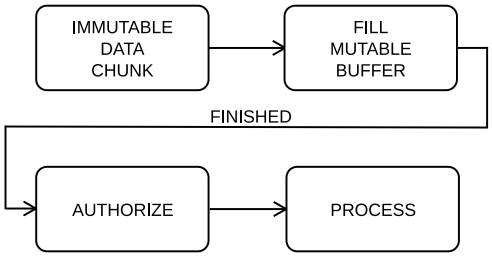

Fig. 1. **The filling buffer method**

However, for IoT communication systems [34] to ensure the integrity of the transmitting data, it's not a wise choice to store an additional copy of the request body in RAM.

The problem with allocations of large chunks of memory is very CPU intensive [2; 35–37] and can add an unnecessary delay in the handling of the request. As a result, for an API which allows large payload size, implementing such an approach would make it an easy target for Distributed Denial of Service (DDoS) attacks.

The “Filling the buffer” method for large and small payloads implies different implementations. The usual way to implement this method for small payloads is to keep the entire buffer in RAM. The buffer size is always equal to the request payload size. The “Buffering to Memory” method is shown in *Fig. 2.*

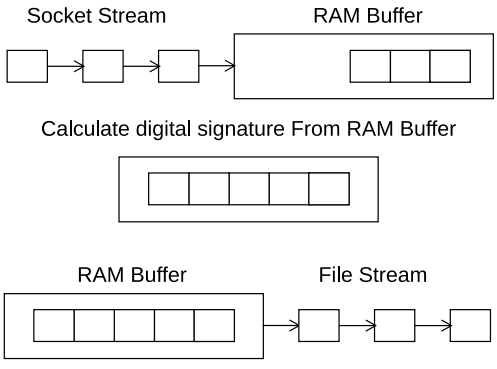

Fig. 2. **The standard technique of filling buffer model**

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

The "Buffering to Memory" method is feasible for the requests with small content size. A general solution of modification of the "filling the buffer" method for HTTP requests with large payload sizes is to store the request body in persistent storage and after the upload is finished and compute the digital signature of the request. The "Buffering to File" method is in *Fig. 3.*

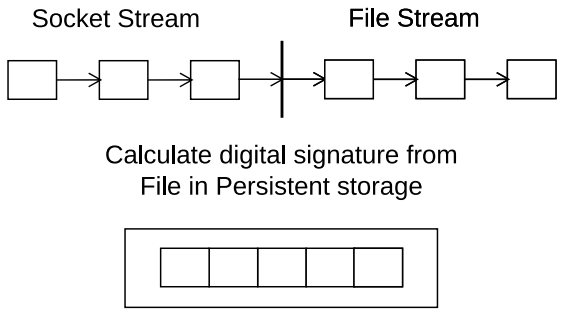

Fig. 3. **"Buffering to file" method for signing request payload**

In our previous works [29; 33], a "chunking" method was developed, which is also called the "method of an efficient authorization for blocking and non-blocking sockets". It has the same purpose as the "buffering to file" method with the difference that it gradually calculates the digital signature of incoming data, computing it from incoming chunks.

The "chunking" method uses a small buffer to store a chunk of data, which must be a multiple of the hashing algorithm block size. Moreover, the chunking method frees the developer from the constraint of keeping the entire request in a buffer. This provides a possibility to implement streaming API with real-time HMAC authentication. The scheme of processing of the request by the "chunking" method is shown in *Fig. 4.*

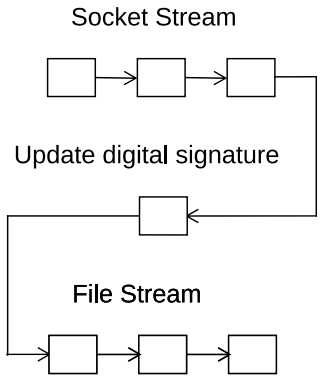

Fig. 4. **"Chunking" method for signing request payload**

It's worth mentioning that the usage of non-blocking sockets provides surpassing performance with the same hardware resources. It is especially significant for transferring large amounts of data over the network, as well as servicing many connections.

Using either "Buffering to file" or "Chunking" methods will release the server from storing the payload of the request in RAM.

ANALYSIS OF THE OPERATION QUEUE ENVIRONMENT FOR AUTHORIZATION METHODS FOR LARGE PAYLOADS

With serving more requests at a time, multithreading gives an enormous advantage if the data of the HTTP request is getting archived or encrypted in the process before used further. Additionally, for streaming, there's no need to buffer an entire payload. Moreover, for modern software, the operation queue implementation won't spawn more threads than available CPU threads.

Operation queues are used to balance the CPU core, which allows the main thread to focus on receiving the data from clients.

After the operation is put to the queue, it's being taken sequentially for each request. The queue is working on the First-In-First-Multi-Out paradigm.

The operation queue is in *Fig. 5.*

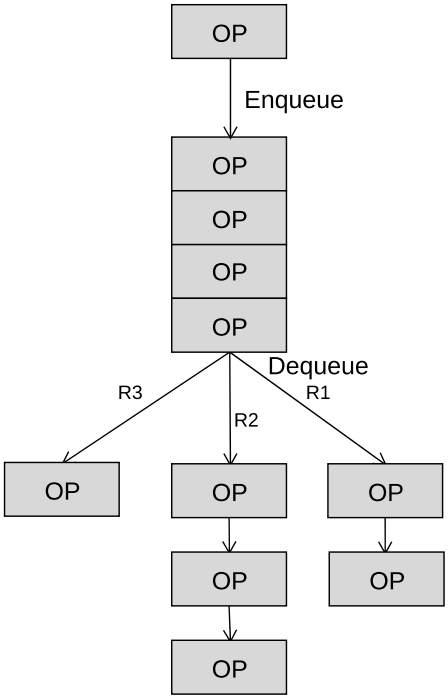

Fig. 5. **Operation Queue**

In the Fig. 5**,** http requests R1, R2, R3 are processed sequentially, which ensures data integrity. This situation is typical as the data is saved to the disk system.

However, queued operations tend to accumulate if they can't be processed fast enough, which can lead to a critical mode. Critical mode increases

memory consumption and leads to system instability.

ANALYTICAL COMPARISON OF THE EXAMINED METHODS

In the multithreading environment, a common way to deal with the chunks of data is using operation queues. The best way to compare the methods analytically is to create a model based on the queuing theory. To distinguish "server" as a process running in the operating system and "queuing theory server", they are called server and QT Server in this paper.

Typically, the weakest point in any system is the disk drive. The drive has a limited writing speed, but in the case of SSDs, it reaches several gigabytes per second.

For the simulation of the behavior of the server, the following formula is assumed to calculate the data arrival rate per one thread:

$$\lambda = \lambda_R + \lambda_A + \lambda_S ,$$

where: λ_R – duration of processing request, seconds;

λ_A – duration of processing temporary information of authorization;

λ_S – duration of saving the data to the server's drive;

λ – the processing duration, seconds.

In the modern server environment, the read/write workload between processor cores usually is shared within the single logical drive. The duration of processing requests (λ_R) is only constrained by the processing power of the server. The duration of saving the data to the server's drive (λ_S) is divided among active requests. It's also designated as (D_{CW}).

Because the calculation of the digital signature is required to read the data from the drive after the upload is complete, the concurrent read duration (D_{CR}) is added into the equation for the "Buffering to drive" method.

$$\lambda_B = \lambda_R + (\lambda_A + D_{CR}) + D_{CW} .$$

Since there is no requirement to access the chunks after they are received, there is no necessity for intermediate storage. That is why the "Chunking" method is constrained only by the processing power of the CPU.

$$\lambda_{C1} = \lambda_R + \lambda_A + D_{CW} .$$

Service rates are subtracted to compare methods

$$\Delta\lambda_1 = \lambda_C - \lambda_B = D_{CR} .$$

Worth noting, there may be the case when storing a request payload to persistent storage is not necessary. Then it is suitable to keep in RAM only the chunks that are in the queue to be authorized.

In this case:

$$\lambda_{C2} = \lambda_R + \lambda_A .$$

The difference increases even more

$$\Delta\lambda_2 = \lambda_C - \lambda_B = D_{CR} + D_{CW} .$$

Despite such a huge difference, because the drive speed is the slowest element in the system, the most common use case for authorizing the payload of the request is uploading a file to a server.

The analysis has clearly shown the advantage of the "chunking" method. Because the methods can't be compared experimentally, it's necessary to compare their implementations. In order to validate the theory, it is necessary to create new methods for ranking payload signing implementations and identifying critical modes.

NEW METHODS FOR RANKING OF PAYLOAD SIGNING IMPLEMENTATIONS AND IDENTIFYING CRITICAL MODES

The processing duration (λ) significantly depends on the number of active clients. Because all the operations are in the queue, the number of threads will depend only on CPU hardware.

The proposed method for ranking payload signing implementations is based on simulation modeling of authorization processes with large payloads. It features the parallelized simulation of payload processing and sampling of comparison based on runtime criteria, compared to existing ones. It allows us to improve the performance of the system by choosing the feasible method of authorization.

In addition, the method provides the following features:

1) to estimate in which regimes the system will go to the critical mode;

2) to compare "Chunking" and "Buffering to file" methods;

3) to determine how the hardware deals with the planned workload.

The subsidiary aim is to reuse the parts of the method for the research of the critical modes in the operation queue environment.

The method assumes the following steps:

1) choose a payload signing implementation to study its characteristics;

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

2) initiate the launch of a predetermined number of connections at the same time;

3) measure the average request processing time for the chosen method of signing the request payload;

4) measure the processing speed per thread and the total data rate;

5) display the results of the measurement.

Because the regimes in which the system goes into critical mode have not yet defined and putting the system into critical mode could cause software instabilities, the method simulates the "thread per connection" pattern. However, the determination of the parameters when the system goes into critical mode is going to provide start conditions for the research. Determination of an average concurrent write speed (D_{CW}) among a given number of threads and service rate (μ) allows us to do this.

The most important parts of the technique are in the paper and presented as C++ code. The BenchmarkChunking and BenchmarkFileBuffering classes have been implemented to study the "Chunking" and "Buffering to file" methods of payload signing (step 1). These classes inherit from the BenchmarkBase class. A virtual function "benchmark" needs to be overridden to create an implementation for a new payload signing method.

For the research of the critical modes in the operation queue environment, the parts of the method can be reused. The SHA-512 algorithm was selected because it consumes the most computing resources among SHA family algorithms. As a means of simulation of the data arrival rate (λ), the next chunk of data is delayed in case if it has arrived too soon.

To control the test conditions, the following constants are defined:

NChunks – number of chunks, should be defined to have the test running enough time to get reliable results;

ChunkSize – the size of the chunk, bytes, should be chosen the low as possible with the best concurrent write speed;

ClientBandwidth – network bandwidth of client, megabits per second.

The following formula is used to find the data arrival rate (microseconds): $\mu = 1/\lambda$.

The result of conversion is in the ChunkTransmissionTimeMs parameter.

In accordance with step 2, a specified number of threads are started simultaneously, where each one processes a defined number of chunks. All the chunks are processed sequentially. After all the chunks are processed, statistics are displayed.

The "benchmark" function for the "chunking" method is shown in *Fig. 6.*

```
virtual void benchmark() { // For Chunking Method
     for (ssize_t i = 0; i < NChunks; i++) {
         uint8_t* data =
                (uint8_t*) malloc(ChunkSize);
         ssize_t chunkWrite = benchmarkChunk(data);
         updateTotalTime<true>(chunkWrite);
     }
     fclose(file);
     printStatsForThread();
     remove(filePath.c_str());
}
```

Fig. 6. **Function "benchmark" for the "chunking" method**

According to step 4 to measure average duration, the test code does the identical calculation of digital signature and writing to the server's drive as in production applications. As a means of measuring the writing service rate for both methods, the standard C++ "std::chrono" package is used.

The function of measuring the writing data arrival rate is in *Fig. 7.*

```
ssize_t benchmarkChunk(const uint8_t* data) {
    auto writeBegin = chr::steady_clock::now();
    CC_SHA512_Update(&ctx, data, ChunkSize);
    fwrite(data, ChunkSize, 1, file);
    auto writeEnd = chr::steady_clock::now();
    return diff_us(writeEnd, writeBegin)
}
```

Fig. 7. **Function of measuring the data arrival rate**

The next important step (3) is to measure the processing time for the studied method of signing the request payload. Despite the relative simplicity of this step, it also adds a delay for the simulation of the data arrival rate (λ). As this step is applied for the research of the critical modes, where the pause takes place in a different position, a conditional compilation is added. The "update total time" function is in *Fig. 8.*

```
template <bool shouldSleep>
void updateTotalTime(ssize_t processTimeMs) {
    ssize_t timeDiff =
        ChunkTransmissionTimeMs - processTimeMs;
    if (timeDiff > 0) {
        if constexpr (shouldSleep) {
            sleep_us(timeDiff);
        }
        totalTimeMs += ChunkTransmissionTimeMs;
    } else {
        totalTimeMs += processTimeMs;
    }
}
```

Fig. 8. **Function for updating total time**

For the "Buffering to file" method, the "benchmark" function is similar to the one for the "chunking" method. The digital signature of the chunk is calculated not in the "benchmarkChunk" function, but in "calculateBufferHmac". It's because

of the calculation of the digital signature is done while reading the file after its uploading. The benchmark function for the implementation of "Buffering to file" method is shown in *Fig. 9.*

```
virtual void benchmark() {
     for (ssize_t i = 0; i < NChunks; i++) {
         uint8_t* data = (uint8_t *) mal-
loc(ChunkSize);
         ssize_t chunkWrite = benchmarkChunk(data);
         free(data);
         updateTotalTime<true>(chunkWrite);
     }
     calculateBufferHmac();
     printStatsForThread();
     fclose(file);
     remove(filePath.c_str());
}
```

Fig. 9. **Function "benchmark" for the "Buffering to file" method**

After uploading, the file is read sequentially and the digital signature is calculated. For "Buffering to File" method, the purpose of this function is to measure the speed of parallel reading, since the test is performed in a multithreaded environment.

The function for measuring the concurrent read speed is in *Fig. 10.*

```
void calculateBufferSignature() {
    fseek(file, 0, SEEK_SET);
    char *readChunk = (char *) malloc(ChunkSize);
    while (!feof(file)) {
        auto readBegin = chr::steady_clock::now();
        ssize_t readBytes =
            fread(readChunk, ChunkSize, 1, file);
        CC_SHA512_Update(&ctx, readChunk, readBytes);
        auto readEnd = chr::steady_clock::now();
        readChunks += 1;
        totalReadTimeMs +=
                diff_us(readEnd, readBegin);
    }
    free(readChunk);
}
```

Fig. 10. **Function for measuring the concurrent read speed**

Since the system uses operation queues, it is prone to falling to critical mode. It happens because operations accumulate in the RAM, and the system does not have time to process them. There is a possibility of system instability and abnormal termination of the server process.

A method was developed based on the payload signing implementations ranking method to identify critical modes. It is distinguished by optimal criteria that determine the transition of the system to the critical mode. It allows for increasing the speed and reliability of the system by choosing real equipment when designing the system.

The new method for identifying critical modes can be used in practice to select the optimal equipment load. The system load can be regulated using our method of migration from a single server to a server cluster [2]. With the new method, the probability of the system going into critical mode is decreased. It affects an increase in the reliability and speed of the system.

To investigate the states in which the system goes into a critical mode, one needs to conduct a series of experiments in which:

1) define the client bandwidth parameter according to the needs of the experiment;

2) in each experiment measure: the total write speed and the average write speed;

3) gradually increase the number of connections while the change of the result write bandwidth is significant.

After the methods were established, it is time to get experimental data in order to fulfill the aims.

EXPERIMENTAL DETERMINATION OF THE CONDITIONS OF MANIFESTATION OF THE ADVANTAGES OF THE STUDIED METHODS OF SIGNING THE PAYLOAD

To compare the methods, a series of experiments is required. They measure the total duration for the investigated implementation of payload signing method, in accordance with the payload signing implementation ranking method described in the previous section. The advantage of the chunking method is that the payload is not read from the buffer the second time.

For all tests conducted in the paper, the chunk size is 1 megabyte. In the production environment, it's a parameter that the user sets up to get the maximum writing performance.

For the test, the following hardware is used:
OS: Ubuntu 18.04 LTS
CPU: Core i7 8700K
RAM: 32G
SSD: Samsung 960 Evo 512G

It is crucial that modern operating systems cache recently used files into RAM. That makes the "Buffering to file" method feasible in the case when payload size is smaller than RAM size, or the server is not loaded heavily. The calculation of the gain of the "chunking" method is carried out according to the following formula:

$$G = \frac{\lambda_C}{\lambda_C + \Delta\lambda},$$

where: G – the percentage gain of the chunking method;

λ_C – processing time for the chunking method;

$\Delta\lambda$ – the difference in processing time between the "chunking" method and the "buffering to file" method.

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

For 100 Mbps, the optimal configuration is 16 threads and payload size per thread from 100MB to 1000 MB.

The measured payload processing times for the two methods are in *Table 1.*

Table 1. **Comparison of the processing time of the payload by different methods for 100 Mbps/client**

Size (Mb)	λ_C	$\Delta\lambda$	G
100	10071226	89685	0.882647
250	23591242	77614	0.327916
350	30965550	45644	0.147186
500	40738723	207820	0.50754
750	62815039	9125822	12.6852
1000	85070208	12898829	11.9905

For small payload sizes, the advantage is not significant and is about 0.5 %. But when the total payload size is more than 12 Gigabytes, then the advantage is 12 %.

For 1Gbps, the optimal configuration is 4 threads and payload size per thread from 250 Megabytes to 4000 Megabytes.

The measured payload processing times for the two methods are in *Table 2.*

Table 2. **Comparison of the processing time of the payload by different methods for 1 Gbps/client**

Size (Mb)	λ_C	$\Delta\lambda$	G
250	2199525	30617	1.37287
500	5595948	57185	1.01156
1000	10565035	122192	1.14335
2000	23621152	242193	1.01492
3000	29601724	8832594	22.981
4000	50147151	10535229	17.3613

The same can be seen for 1Gbps, where the advantage is 20 % in case the total payload size is more than 12 Gigabytes.

It can be concluded that the "Chunking" method shows the advantage in the range of 10-22 % when there's not enough memory for the total payload size.

STUDY OF THE INFLUENCE OF THE INTENSITY OF THE INCOMING DATA ON THE FALLING OF THE SYSTEM INTO THE CRITICAL MODE

According to the critical mode identification method, the total write speed and the average write speed are measured in each experiment. At the same time, in the next experiment, the number of connections is gradually increased, provided that the overall write speed changes significantly.

For the first series of experiments, the usual upload bandwidth for the 2020 year of 100 Mbps is chosen. The results are in *Table 3.*

The following symbols are used in the table:

#t – number of connections / threads;

R_{CW} - average processing rate per thread;

$\sum R_{CW}$ is the total data processing rate of all threads.

Table 3. **Concurrent write speed for 100 Mbps/client**

#t	R_{CW} (MB/s)	$\sum R_{CW}$ (MB/s)
32	10.107	323.446
34	10.327	351.125
36	10.235	368.478
38	9.872	375.172
40	9.780	391.210
42	9.594	402.975
44	9.004	396.178
46	8.850	407.1

The histogram representation of the table is in the *Fig. 11.* Each pair in the chart represents: average concurrent write speed and the sum of concurrent write speeds.

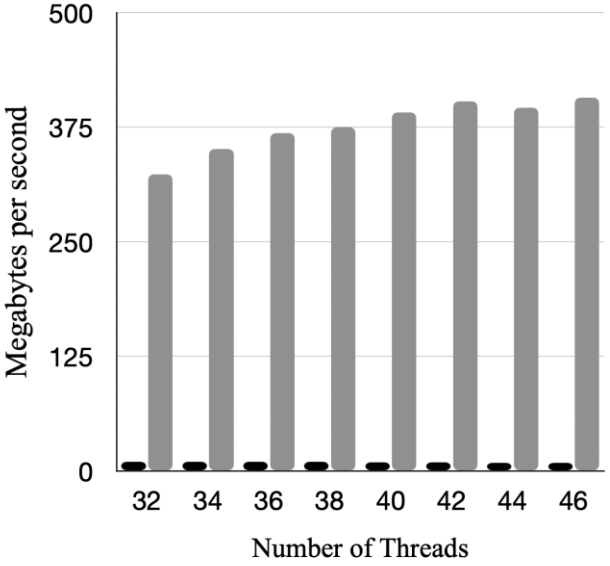

Fig. 11. **Concurrent write speed for 100 Mbps/client**

For 100 Mbps per client, the decrease of average concurrent write speed starts from 38 threads. That is where the critical mode is likely to happen in the operation queue environment.

In the next experiment, it's assumed that client has a premium internet connection of 1 Gbps. The maximum number of threads is limited to 12. The results are in *Table 4.*

And the histogram representation of the table is in *Fig. 12.*

Table 4. **Concurrent write speed for 1Gbps/client**

#	R_{CW} (MB/s)	$\sum R_{CW}$ (MB/s)
1	115.838	115.838
2	119.2	238.817
3	112.012	336.036
4	116.159	464.636
5	83.2644	416.322
6	75.4237	452.542
7	72.5202	507.641
8	68.2824	546.259
9	52.794	475.146
10	49.3414	510.456
11	46.3804	517.243
12	39.6218	486.632

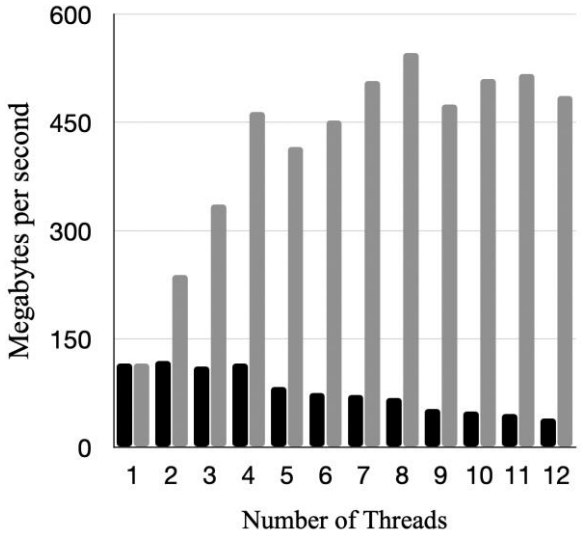

Fig. 12. **Concurrent write speed for 1 Gbps/client**

The total speed doesn't increase after 4 connections, after that the critical mode is going to be. That's because input bandwidth is more than the drive concurrent writing speed. As a note, the maximum concurrent write speed (546MB/s) is not the maximum sequential drive write speed (1500 MB/s) for this drive.

The obtained results can serve as a starting point for studying the critical modes themselves. In practice, this allows us to limit the load on the system using our method of migration from a single server to a server cluster[2]. It can also be used to select the optimal equipment.

CONCLUSION

The modern methods of signing the payload of the request were reviewed and compared. Comparing the “Buffering to file” and the “Chunking” method, it was found that the “Buffering to file” method introduces “Concurrent Read Time” to the processing of the request. Functionally, for the case of uploading files to the server, these methods are equivalent. For streaming audio and video tasks, and so on, where file storage on the server is not required, the “chunking" method has the advantage of saving disk space.

The developed methods for ranking of payload signing implementations and identifying critical modes made it possible to experimentally determine the conditions for the manifestation of the advantages of the studied methods of signing the payload and to study the influence of the incoming data rate on the falling of the system to the critical mode.

The “Chunking” method showed the advantage of 22 % for 100 Mbps and 13 % for 1Gbps client speeds when OS has no available memory to cache the data. However, when OS can cache the incoming data, the advantage drops to the value of about 1 %.

The influence of the incoming data rate on the falling of the system to the critical mode is studied. It makes it possible to limit the load of the system. Thus, the goal of reducing the harmful effect of critical modes and providing better reliability and speed of the system is achieved.

REFERENCES

1. Kizza, J. M. “Computer Network Security and Cyber Ethics Fourth Edition”. *Publ. McFarland.* Jefferson. NC, United States: 2014. 240 p.

2. Surkov, S. & Martynyuk, O. “Method of Migration from Single Server System to Server Cluster”. In *Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS’2015)*. Warsaw, Poland: 2015. DOI: 10.1109/IDAACS.2015.7341415.

3. Fielding, R. & Reschke, J. (2015). “ Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, IETF RFC 7230”. Available from: https://tools.ietf.org/html/rfc7230. [Accessed 12th August 2020].

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

4. Belshe, M. & Peon, R. (2015). “Hypertext Transfer Protocol Version 2 (HTTP/2), IETF RFC 7540”. Available from: https://tools.ietf.org/html/rfc7540. [Accessed 12th August 2020].

5. Sanae, H. “Security Requirements and Model for Mobile Agent Authentication”. *Smart Network Inspired Paradigm and Approaches in IoT Applications. Republic of Singapore*. Singapore: 2019. p. 179–189. DOI: 10.1007/978-981-13-8614-5_11.

6. Liu, Q., Zhang, L. & Fan, A. “Scheme to authenticate requests for online banking based on identity-based mediated RSA”. *Jiefangjun Ligong Daxue Xuebao/Journal of PLA University of Science and Technology (Natural Science Edition)*. Beijing, China: 2015; Vol.16: 29–33. DOI: 10.7666/j.issn.1009-3443.20140929001.

7. Saini, K. “Squid Proxy Server 3.1: Beginner's Guide Paperback”. *Publ. Packt Publishing.* Birmingham, United Kingdom: 2011. 332 p.

8. Wessels, D.“Squid: The Definitive Guide”. Sebastopol, CA, United States: *Publ. O'Reilly Media.* 2010. 472 p.

9. Rash, M. “Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort”. San Francisco, CA, United States: *Publ. No Starch Press.* 2007. 336 p.

10. Fjordvald, M. & Nedelcu, C. “Nginx HTTP Server – Fourth Edition: Harness the power of Nginx to make the most of your infrastructure and serve pages faster than ever before”. Birmingham, United Kingdom: *Publ. Packt Publishing*. 2018. 400 p.

11. Blokdyk, G. “Apache Web Server A Complete Guide – 2020 Edition”. Brisbane, Australia: *Publ. 5STARCooks*. 2020. 238 p.

12. Eugene, F., John, O. R. & Kevin, C. “Security evaluation of the OAuth 2.0 framework”. *Information and Computer Security*. 2015; Vol. 23(1): 73–101. DOI: 10.1108/ICS-12-2013-0089.

13. Cheol-Joo, Chae Ki-Bong & Han-Jin Cho. “A study on secure user authentication and authorization in OAuth protocol”. *Springer Cluster Computing*. 2019; Vol. 22(2). DOI: 10.1007/s10586-017-1119-6.

14. Farooqi, S., Zaffar, F., Leontiadis, N., et al. “Measuring and mitigating OAuth access token abuse by collusion networks”. In *Communications of the ACM*. New York, NY, United States: 2020. p. 103–111. DOI: 10.1145/3387720.

15. Feng, Y. & Sathiamoorthy, M. “A security analysis of the OAuth protocol”. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. Victoria, BC, Canada: 2013. p. 271–276. DOI: 10.1109/PACRIM.2013.6625487.

16. Seung, J. S., J. “Personal OAuth authorization server and push OAuth for Internet of Things”. *International Journal of Distributed Sensor Networks.* Thousand Oaks, CA, United States: 2017; Vol. 13. DOI: 10.1177/1550147717712627.

17. Se-Ra, O. & Young-Gab, K. “AFaaS: Authorization framework as a service for Internet of Things based on interoperable OAuth”. *International Journal of Distributed Sensor Networks*. Thousand Oaks, CA, United States: 2020; Vol. 16(2): 1–15. DOI: 10.1177/1550147720906388.

18. Hossain, N., Hossain, M.A., Hossain, M., et al. “OAuth-SSO: A Framework to Secure the OAuth-based SSO Service for Packaged Web Applications”. In *Proc. of 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. New York, NY, United States: 2018. p. 1575–1578. DOI: 10.1109/TrustCom/BigDataSE.2018.00227.

19. El-hajj, M., Fadlallah, A., Maroun, C., et al. “A Survey of Internet of Things (IoT) Authentication Schemes”. *Sensors – Open Access Journal*. Basel, Switzerland. 2019; Vol. 19: 1–17. DOI: 10.3390/s19051141.

20. Hardt, D. “The OAuth 2.0 Authorization Framework, IETF RFC 6749”. Available from: https://tools.ietf.org/html/rfc6749. [Accessed 26th Jule 2020].

21. Jones, M. & Bradley, J. “JSON Web Token (JWT) IETF RFC 7519”. Available from: https://tools.ietf.org/html/rfc7519. [Accessed 18th Jule 2020].

22. Richer, J. “User Authentication with OAuth 2.0”. Available from: https://oauth.net/articles/authentication/. [Accessed 17th Jule 2020].

23. Krawczyk, H. & Bellare, M. “HMAC: Keyed-Hashing for Message Authentication”. Available from: https://tools.ietf.org/html/rfc2104. [Accessed 02th Jule 2020].

24. Leiba, B.. “OAuth Web Authorization Protocol”. *IEEE Internet Computing*. Nicosia, Cyprus: 2012; Vol. 16: 74–77. DOI: 10.1109/MIC.2012.11.

25. Hammer-Lahav, E. (2010). “The OAuth 1.0 Protocol. IETF RFC 5849”. Available from: http://tools.ietf.org/html/rfc5849. [Accessed 15th August 2020].

26. Frost, M. “Integrating Web Services with OAuth and PHP ”. Alexandria, VA, United States: *Publ. musketeers.me*. 2016. 116 p.

27. Hammer, E. “OAuth 2.0 and the Road to Hell”. Available from: http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/. [Accessed 29th June 2020].

28. Hammer, E. “HAWK / HTTP Holder-Of-Key Authentication Scheme”. Available from: https://github.com/hueniverse/hawk. [Accessed 14th August 2020].

29. Surkov, S. S. “Model and method of chunk processing of payload for HTTP authorization protocols”. *Proceedings of 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*. Slavske, Ukraine: 2020. p. 317–321. DOI: 10.1109/TCSET49122.2020.235447.

30. Surkov, S. S. & Martynyuk, O. M.. “Improvement of security for web services by research and development of OAuth server”. *Electrotechnic and Computer Systems.* Odesa, Ukraine: 2016; Vol. 23(99): 99–105. DOI: 10.15276/eltecs.23.99.2016.16.

31. Apple Inc. “Grand Central Dispatch”. Available from: https://github.com/apple/swift-corelibs-libdispatch. [Accessed 27th June 2020].

32. Grosch, S. “Concurrency by Tutorials (Second Edition): Multithreading in Swift with GCD and Operations”. McGaheysville. VA, United States. *Publ. Razeware LLC.* 2020.100 p.

33. Surkov, S. S., Martynyuk, O. M. & Mileiko, I. G. “Modification of open authorization protocol for verification of request“. *Electrotechnic and Computer systems.* Odesa, Ukraine: 2015; Vol. 19(95): 178–181 (in Russian).

34. Mukherjee, A. “Physical-Layer Security in the Internet of Things: Sensing and Communication Confidentiality Under Resource Constraints”. *Proceedings of the IEEE*. 2015; Vol. 103: 1747–1761. DOI: 10.1109/JPROC.2015.2466548.

35. Drozd, O., Kharchenko, V., Rucinski, A., et al. “Development of Models in Resilient Computing”. In *Proc. of 10th IEEE International Conference on Dependable Systems. Services and Technologies (DESSERT’2019)*. Leeds, UK. DOI: 10.1109/DESSERT.2019.8770035.

36.. Drozd, O., Kuznietsov, M., Martynyuk, O., et al. “A method of the hidden faults elimination in FPGA projects for the critical applications”. In *Proc. of 9th IEEE International Conference on Dependable Systems, Services and Technologies (DESSERT’2018)*. Kyiv, Ukraine: 2018. p. 231–234. DOI: 10.1109/DESSERT.2018.8409131.

37. Drozd, A., Antoshchuk, S., Drozd, J., et al. “Checkable FPGA Design: Energy Consumption, Throughput and Trustworthiness”. In *Green IT Engineering: Social, Business and Industrial Applications, Studies in Systems, Decision and Contro.*, Warsaw, Poland: 2018. p. 73–94. DOI: 10.1007/978-3-030-00253-4_4.

DOI: 10.15276/hait.03.2020.5
UDK 004.75

Порівняння протоколів авторизації для великих запитів у середовищі черг операцій

Сергій С. Сурков
Одеський національний політехнічний університет, Одеса, Україна
ORCID: http://orcid.org/0000-0001-9224-7526

АНОТАЦІЯ

Протоколи авторизації відіграють основну роль в веб-безпеки. Існує небезпека того, що дані будуть змінені в місцях, де вони передаються в незашифрованому вигляді, наприклад проксі сервера. Ситуації підміни корисного навантаження в уразливих точках є малодослідженими. Існуючі протоколи авторизації великих корисних навантажень реалізують метод «Заповнення Буфера» і його підвид «Буферизація в файл». За допомогою імітаційного моделювання було виявлено, що другий підвид "Буферизація в пам'ять" не є ідеальним для запитів з великими корисними навантаженнями. У попередніх роботах було розроблено «порційний» метод, призначений для авторизації HTTP запитів з ефективної верифікації корисного навантаження. Однак було виявлено, що в середовищі черзі операцій система схильна до переходу в критичний режим, що стало предметом подальшого вивчення. Метою даного дослідження є зниження шкідливого ефекту критичних режимів. Аналіз

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

методів підписання для авторизації з використанням паралельної архітектури на основі теорії черг показав, що «порційний» метод є найбільш перспективним. Розроблені методи ранжирування протоколів авторизації для великих запитів і виявлення критичних режимів дозволили експериментально визначити умови прояву переваг досліджуваних методів підписання корисного навантаження в різних режимах і досліджувати вплив інтенсивності вхідного потоку інформації на перехід системи в критичний режим. Шляхом комп'ютерного експерименту були отримані залежності швидкості багатопотокової записи від кількості потоків для «порційного» і «буферизация в файл» методів в залежності від кількості потоків і швидкості передачі даних. Паралельна обробка цифрового підпису запитів підвищила продуктивність системи, забезпечуючи при цьому послідовну обробку даних запиту. Дослідження вплив інтенсивності вхідного потоку інформації на перехід системи в критичний режим дає можливість розрахувати обмеження навантаження системи. Таким чином досягається мета зниження шкідливого ефекту критичних режимів і забезпечення більшої надійності і швидкодії системи.

Ключові слова: цифровий підпис; авторизація; великий розмір запиту; черги операцій; мережеві запити; верифікація

DOI: 10.15276/hait.03.2020.5
UDK 004.75

Сравнение протоколов авторизации для больших запросов в среде очередей операций

Сергей С. Сурков
Одесский национальный политехнический университет, Одесса, Украина
ORCID: http://orcid.org/0000-0001-9224-7526

АННОТАЦИЯ

Протоколы авторизации играют основополагающую роль в веб-безопасности. Существует опасность того, что данные будут изменены в местах, где они передаются в незашифрованном виде, например прокси сервера.
Ситуации подмены полезной нагрузки в уязвимых точках являются малоисследованными. Существующие протоколы авторизации больших полезных нагрузок реализуют метод «Заполнение Буфера» и его подвид «Буферизация в файл». С помощью имитационного моделирования было обнаружено, что второй подвид «Буферизация в память» не является идеальным для запросов с большими полезными нагрузками. В предыдущих работах был разработан «порционный» метод, предназначенный для авторизации HTTP запросов с эффективной верификации полезной нагрузки. Однако было обнаружено, что в среде очереди операций система склонна к переходу в критический режим, что стало предметом дальнейшего изучения. Целью данного исследования является снижение вредного эффекта критических режимов. Анализ методов подписания для авторизации с использованием параллельной архитектуры на основе теории очередей показал, что «порционный» метод представляется наиболее перспективным. Разработанные методы ранжирования протоколов авторизации для больших запросов и выявления критических режимов позволили экспериментально определить условия проявления преимуществ исследуемых методов подписания полезной нагрузки в разных режимах и исследовать влияние интенсивности входящего потока информации на переход системы в критический режим. Путем компьютерного эксперимента были получены зависимости скорости многопоточной записи от количества потоков для «порционного» и «буферизация в файл» методов в зависимости от количества потоков и скорости передачи данных. Параллельная обработка цифровой подписи запросов повысила производительность системы, обеспечивая при этом последовательную обработку данных запроса. Исследование влияние интенсивности входящего потока информации на переход системы в критический режим дает возможность рассчитать ограничение нагрузки системы. Таким образом, достигается цель снижения вредного эффекта критических режимов и обеспечения большей надежности и быстродействия системы.

Ключевые слова: цифровая подпись; авторизация; большой размер запрпоса; очереди операций; сетевые запросы; верификация

ABOUT THE AUTHORS

Sergii S. Surkov – PhD Student of Computer Intellectual
Sytems and Networks Department, Odessa National Polytechnic University, Odessa, Ukraine
k1x0r@ukr.net

Сергій С. Сурков – аспірант каф. комп'ютерних інтелектуальних систем і мереж, Одеський національний політехнічний університет, Одеса, Україна

Сергей С. Сурков – аспирант кафедры компьютерных интеллектуальных систем и сетей, Одесский национальный политехнический университет Одесса, Украина

Received 03.08.2020
Received after revision 15.09.2020
Accepted 20.09.2020