The Implement of Voting Device in Mimicry Defense Model

Ming Jiang  
School of Computing, HangZhou Dianzi University, ZheJiang, China

Huibing Liang  
School of Computing, HangZhou Dianzi University, ZheJiang, China

Jingfan Tang*  
School of Computing, HangZhou Dianzi University, Hangzhou 310018, China  
Corresponding author (Email: tangjf@hdu.edu.cn)

Min Zhang  
School of Computing, HangZhou Dianzi University, ZheJiang, China

Enhao Hu  
School of Computing, HangZhou Dianzi University, ZheJiang, China

Abstract  
“Mimicry Defense Model” is a model based on the structure of “Dynamic Heterogeneous Redundancy”, that copy the same input through the distributor as N copies and distribute it to the N heterogeneous assemblies in the executor collection for processing. The processing results are collected to the voting device to vote, and the only relative correct output is obtained, so that the security of the Web server can be greatly improved. Among them, distributor and voting device are both the most important devices, and their performance also directly determines the performance of the whole server system. This paper uses buffer to temporarily store data packages sent by file server, create a queue for each server, and hash the packages to the Hash table. The Hash table uses a red-black-tree to reduce operational complexity, implements a voting device with low delay and high efficiency, which provides a more efficient solution for the whole system.  
Key words: Mimicry Defense Model, Dynamic Heterogeneous Redundancy, Voting Device

1. INTRODUCTION

The security problems faced by Web server system are becoming more and more serious, which poses a serious threat to the information security of enterprises and individuals(Chunming Wu, 2016; Minjie Ma, Fenggang Sun, and Lidong Zhai, 2014). Traditional firewalls and intrusion detection technologies are always in a passive defense position, and cannot cope with the threat of unknown attacks well. The idea of moving target defense (MTD)(Jajodia, S., Ghosh, A. K., Swarup, V., Wang, C., Wang, X. S., 2011; Xiaoyu Zhang, Zhengbang Li, 2013) was proposed in the United States, which was a "change the rules of the game" technology to against weakness of the current defensive position. It was expected to confuse an attacker by implementing sustained and dynamic changes to increase the attack cost and complexity, and reduce the attack success rate(Guilin, Cai, Baosheng Wang, and Tianzuo Wang, 2016).

“Mimicry Defense Technology” (Qing Tong, Zheng Zhang, Weihua Zhang, Jiangxing Wu, 2017; Jangxing Wu, 2014) realizes active defense based on “dynamic heterogeneous redundancy” structure, which is handled in the processing section using a heterogeneous set of executable. It copies the same input through the distributor as N copies and distributes it to the N heterogeneous assemblies in the executor collection for processing. The processing results are collected to the voting device to vote, and the only relative correct output is obtained. For an attack executed, the system replaces it by dynamically generating a new executable entity. Among them, the voting device is one of the most important devices, and their performance also directly determines the performance of the whole server system (Jianmin Pang, Yujia Zhang, Zheng Zhang, 2016).

This paper discusses the dynamic heterogeneous redundancy technology, and proposes a voting comparison algorithm, which implements the voting function, analyzes the factors that influence the performance of the voting device, and provides a more efficient solution for the whole set of systems. Compared with the original solution that compares all the files after downloading, the voting device presented in this paper has better time performance and space performance. After the attack or malfunction occurs, the alarm can be sent to the monitoring system to remind the monitoring system to reconfigure and realize the discovery. Accordingly, the voting device has better augmentability, which can shield and recovery of the attack or malfunction.
2. VOTING DEVICE

The voting structure is as follows. We suppose the number of servers is $N$, the server $S_i$ transfers the corresponding data stream $T_i$ to the voting device, and then returns the data to the user after the processor's processing (Jun Xu, Pinyao Guo, Bo Chen, Robert F., Ping Chen, Peng Liu, 2016).

As shown in Figure 3, the upstream server group has $N$ stations, and the voting device needs to analyze the results from them, which has the following situations:

1. In case of no attack and server error, all returned results are consistent, so one of the selected results is
returned to the downstream client.

(2) When some of the results are inconsistent, but there are still N/2+1 of the same results, we should choose to return the results with the big probability, which not only affect the normal business by the attack, but also confuse the attacker.

(3) When no same result exceeds the number of N/2+1, it is not sure which result is the most correct, and the voting device returns the specified error result. It will create log files locally when the upstream server returns inconsistent results. The voting device will deposit it in the suspicious log, and send the corresponding information to the administrator to realize the alarm of the attack. Work to create a log file should be handled by a dynamic heterogeneous redundancy system (Cox B., Evans D., Filipi A., Rowanhill J., Hu W., Davidson J., 2006).

Voting algorithm is the basic principle of voting device, which needs to solve the following problems:

1) How to store the data packages and compare them sequentially after they arrive at the voting machine? The packages are stored in the buffer and are sent sequentially to the blocking queue by buffers to transmit data sequentially.

2) How to make rapid comparative test? The voting device will hash data packages to Hash table, which uses red-black-trees to reduce operational complexity and speed up the comparison.

3. SYSTEM ARCHITECTURE

The overall architecture of the voting device is shown in Figure. The file stores multiple copies in each server, and the voting device receives the data from the server into the temporary buffer. The buffer is divided into several fixed size regions, each storing data from the corresponding server and mapping the corresponding blocking queue. Once data is passed into the queue, it will be mapped to the Hash table in turn. The same packages will be mapped to the same location in the Hash table. When the number of packages exceeds N/2, the package is declared done. At this time, the Sender can return the secure and reliable packages back to the client (Chengchen Hu, Yi Tang, Xuefei Chen, Bin Liu, 2007).

![Figure 4. System Architecture](image)

3.1 Buffer Structure
The data packages from each server will be stored into the corresponding region in the buffer, and the N server requires N buffer regions, and N threads.

3.2 Buffer Queue Structure
Data stored in the buffer needs a blocking queue for sequential comparison. The elements in the queue correspond to the buffer region, in addition to the increasing number of a queue, packages from the region the queue, you need to add a number in the end of queue, each package over comparison when you need to delete a number from the head of queue. So the N buffer regions requires N+1 queues and N threads. The data packages in the queue can start comparing without waiting for all packages in the queue to arrive by using multi-threading. Each queue's package is compared to the queue header number of the number queue before comparing. If the number in the number queue is greater than this queue, the package representation of the package has been completed, and the package needs to be discarded. This ensures a sequential comparison of data.
3.3 Structure of Hash Table

The Hash table is the area where data packages compare. In the process of comparison, a lot of search operations are needed, and the search speed of the Hash table is fast, so the comparison in the Hash table will have a better time performance.

There is the mapping relationship between Hash table and the buffer, which will be denoted by $P_i \rightarrow (B_k,n)$. Each entry contains two data fields: the identity of the package $P_i$, the number of times mapped to it, and the location of the corresponding package region. When the data packages compare finished, its mapping entries will be removed from the Hash table, and the corresponding buffer locations will be released.

The Hash table uses a single list to resolve the conflict. If two packages in queue P and Q get the same result through the hash function, it will connect them into the list of the hash table. When the chain length exceeds a certain threshold, the list will be converted to red-black-tree to improve search efficiency.

When the package $P_i$ arrives, the voting device will find out whether it is in the Hash table through the identification field. The number of packages will be added by one if it is in the Hash table. Otherwise it will add an item $P_i \rightarrow (B_k,1)$. Once the number of item is more than $N/2$, it means more than $N/2$ of the packages in the queue are exactly the same as the package $P_i$. So this package voting has been completed, it will immediately returns the mapping entries and removed $P_i \rightarrow (B_k,n)$ from the table, in order to release the space. Because the data using the input stream, each region does not need to store all the data of a stream. It only needs to store the current data package for comparison with a number of data packages as a buffer. When a vote is completed, the voting device will release the occupied space by current voting. As a result, the space occupancy in the voting device can be reduced from $N$ full large files to several small packages.

With the continuous transmission of packages, the voting device can return the package and release the space if the number of packages is more than $N/2$. However, there are more and more legacy packages in Hash table since it cannot locate and eliminate the exception packages that are no more than $N/2$. The voting device needs to clean the entire Hash table in a period of time. In the cleaning process, the voting device will set a synchronization point in each queue, and the packages in queue will be blocked when they arrive at the synchronization point. In other words, each thread pauses reading the data in the queue, and wait until all the queues arrive at the synchronization point. At that time, the cleaning program will be started. The voting device will clear all data from the Hash table, restore the initial state, wake up all the queues, and proceed with the following operation.

![Figure 5. Hash Table](image)

Packages from different servers are stored in different queues. The packages are mapped to the Hash table through the hash function. Because each thread maintains a queue, the operation of inserting into the Hash table is parallel. If the synchronization is not executed on the Hash table, it will cause serious problems.

We solve the problem of Hash table synchronization by locking the chain header or the red-black tree root node. In the process of thread insertion, we lock the chain header or the red-black tree root node, and then release the lock after the insertion completes. In this way, the Hash table is thread-safe. There is no need to lock during reading because multiple threads access the Hash table at the same time without causing thread synchronization problems.
4. PERFORMANCE ANALYSIS

The performance of the voting device is divided into two aspects: time performance and space performance.

The time performance of the voting device is directly affected by the time performance of the Hash table which depends entirely on the performance of the lookup operation since it will be executed in the Hash table before executing insert, delete, and modify operations. The time for comparing each element in hash table is fixed, so it will significantly improve the lookup performance of Hash by reducing the average number of lookups.

The space occupancy of the voting device depends on the length of the buffer and queue. If the processing in Hash table is slow, the data will pile up in queues and buffers, which will require larger buffers and queues to store data. The space occupancy will be increased, which will also affect the performance of the entire system. However, the speed of data processing locally is much faster than the speed of data transfer in the network, the buffer size and queue length are not consumed very much.

The Hash table contains a count flag with an initial value of 0. If a package is hashed to the element and performs an insert operation, its value is added by 1. For the delete operation, its value is subtracted by 1, while for the other case, it remains the same. Suppose the length of the Hash table is \( L \), and there are \( M_1, M_2, \ldots, M_L \) packages stored in each table entry, the number of packages stored in the table is \( M = M_1 + M_2 + \ldots + M_L \).

In the voting process, the data package will be inserted and deleted only once, while be queried many times. The number of queries is proportional to the number of \( M \) packages in the Hash table. Therefore, when each Hash table item is organized by a linked list, the time complexity of the whole system is:

\[
T = M \times T(\text{query}) + T(\text{insert}) + T(\text{delete})
\]  \( \text{(1)} \)

Query: when looking for data, it will first calculate the Hash value of the package to find the corresponding location in the Hash table, and execute the corresponding operation. When each Hash table entry is organized by a linked list, the average length of successful search is \( 1 + \frac{M}{2L} \) under simple uniform hashing.

Theorem: the time complexity of the voting device is \( O(N^*M/L) \).

Proof: For each package, the average length of search in the Hash table is \( L(\text{query}) \), the time complexity for the process of searching each package is \( O(1) \). In the voting process among the \( N \) server, the voting time \( T = N^*M/L \).

From the above formula, the time complexity of inserting, deleting, and query operations is \( O(M/L) \). Therefore, when the Hash table entry is organized by a linked list, the time complexity of the whole system is \( O(N^*M/L) \). So, if we want to improve the time performance of the voting device, it is a feasible way to reduce the average search length of the Hash table.

When the list length is too long, the search efficiency will be decreased. We can replace the list by using the red-black-tree. When the Hash table uses the red-black-tree structure, the average search length will become \( \log(M_k) \) if the number of entries is \( M_k \). Each table corresponds to the number of nodes is \( M/L \) in a simple uniform hashing. The average success of Hash table search is \( 1 + \log (M/L) \).

5. EXPERIMENTAL VERIFICATION

5.1 Experimental Setup

Experimental platform is implemented in the local area network, the concrete structure is as follows:

![Figure 6. Voting structure](image)

We used one client and two servers in the experiment. One server realized the Nginx module of distributor and voting device. The other server used port mapping to simulate \( N \) server: \( S_1, S_2, \ldots, S_n \). We
performed two groups of the experiments. One group used the algorithm proposed in this paper, the other group used the way to transfer all files to the voting units and then start the comparison. The experiment was carried out separately, and the experimental data were obtained successfully.

5.2 Results Analysis

1) The relationship between the size of files and time

We suppose \( N \) as 3, and the server's corresponding address configuration and URL information are as follows:

<table>
<thead>
<tr>
<th>Server</th>
<th>IP Address:Port</th>
<th>Hostname</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>10.1.17.26:8088</td>
<td><a href="http://10.1.17.26:8088/mtd">http://10.1.17.26:8088/mtd</a></td>
</tr>
<tr>
<td>( S_2 )</td>
<td>10.1.17.26:8089</td>
<td><a href="http://10.1.17.26:8089/mtd">http://10.1.17.26:8089/mtd</a></td>
</tr>
<tr>
<td>( S_3 )</td>
<td>10.1.17.26:8090</td>
<td><a href="http://10.1.17.26:8090/mtd">http://10.1.17.26:8090/mtd</a></td>
</tr>
<tr>
<td>Nginx</td>
<td>10.1.17.10:80</td>
<td><a href="http://10.1.17.10/multipost">http://10.1.17.10/multipost</a></td>
</tr>
</tbody>
</table>

There are seven files (file1.iso, file2.iso, file3.iso, file4.iso, file5.iso, file6.iso, file7.iso) in all the three servers, with the size of 50M, 100M, 200M, 300M, 400M, 500M, 600M. The experiments run many times until the results are stable. The line chart of the file size and the transmission time is shown as follows:

![Figure 7](image-url)

Figure 7. The relationship between the size of files and time

The “download” is the time for simple file download. As shown in the figure, it doesn’t need to wait for all files to cache the CMP server in the comparison process through the way of stream processing. When the file is small, there is little difference on the spent time between these two ways because the transmission is very fast. When the documents gradually increase, the old method of calculating sha256sum time will be longer and it will consume more time. While for the new method, the data comparison will be carried out at the same time in the transmission process, so the time occupied has no obvious increase.

2) The relationship between the number of servers and time

It uses port mapping in the server to simulate \( N \) servers: \( S_1, S_2, ..., S_n \). The appropriate address configuration and Web site information is shown as follows:

<table>
<thead>
<tr>
<th>Server</th>
<th>IP Address:Port</th>
<th>Hostname</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>10.1.17.26:8088</td>
<td><a href="http://10.1.17.26:8088/mtd">http://10.1.17.26:8088/mtd</a></td>
</tr>
<tr>
<td>( S_2 )</td>
<td>10.1.17.26:8089</td>
<td><a href="http://10.1.17.26:8089/mtd">http://10.1.17.26:8089/mtd</a></td>
</tr>
<tr>
<td>( S_3 )</td>
<td>10.1.17.26(n-8087)</td>
<td><a href="http://10.1.17.26(n-8087)/mtd">http://10.1.17.26(n-8087)/mtd</a></td>
</tr>
<tr>
<td>Nginx</td>
<td>10.1.17.10:80</td>
<td><a href="http://10.1.17.10/multipost">http://10.1.17.10/multipost</a></td>
</tr>
</tbody>
</table>

There is one file (file2. ISO) in all the three servers, and the size is 100M. It will be run several times until the result is stable.

The “download” is the time for simple file download. As shown in the Figure, in the case of only one server, it will not perform comparison for the old method after the transfer is completed. But for the new method, it has adopted a multi-thread approach, so there is little difference on the spent time between these two ways. When the service is gradually increased in number, it requires all the files to be downloaded for comparison in the old method. So, the performance cannot be increased much for the more files. While for the new method, it
can perform comparison immediately when the related files complete the download, which shows the better time complexity.

![Figure 8. The relationship between the number of servers and time](image)

6. CONCLUSIONS

In the network attack, the attack is dependent on the platform. “Mimicry Defense Model” based on the dynamic redundancy structure uses this characteristic. Through the heterogeneous and dynamic multi-level, it can reduce the probability of success by single-step attack, disturb the feedback information of the attack, and increase the uncertainty of the system, so as to increase the difficulty of the attack and enhance the security of the system. This paper analyzed many difficulties in the implementation of voting device, and put forward a model of voting device realizing “Mimicry Defense Model”. It uses buffer, buffer queue and Hash table to implement a voting device with low delay and high efficiency, which provides a more efficient solution for the whole system.

ACKNOWLEDGMENTS

This work is supported by national key research and development program (No. 2016YFB0800201) and Zhejiang Provincial Technical Plan Project (No. 2017C01064, 2017C01055, 2018C01088).

REFERENCES