International Journal of Research in Advent Technology, Vol.7, No.4, April 2019
E-ISSN: 2321-9637
Available online at www.ijrat.org

Security Against Fork Bomb Attack in Linux Based Systems

Krunalkumar D. Shah[1], Krunal V. Patel[2]
Information Technology Department[1,2], SSEC Bhavnagar[1,2]
Email: einstein1410@gmail.com[1], krunal220@gmail.com[2]

Abstract-Linux is one of the most popular and widely used operating system in devices ranging from servers to tiny embedded gadgets. However, linux has greatly enhanced the security in many ways, but still it suffers from many attacks. A major process security issue called Fork Bomb is one of them, which is denial of service attack in which process continually creates itself to make system down or crash due to resource starvation. Most of the solutions found in the literature has their own limitations like false positive detection and resource unavailability. To preserve one goal that is availability among the CIA (Confidentiality, Integrity and Availability) of information security, we proposed to develop efficient solution which handles the fork bomb attack in such a way that system remains available for use by end user.

Index Terms-Linux, Process, Overload, Fork, Bomb, Availability

1. INTRODUCTION

Operating system is one kind of system software which deals with process management, memory management, providing security and all. In short, operating system manages the resources of computing system. When a process or user requests for the resources in order to accomplish certain task, requested resources will be allocated by the operating system and more specifically by the core part called kernel. Once the task has been finished, all the resources are made free and marked as available for use by other process and users. Thus, OS works as a resource manager as it does the management of resources.

When a process creates a sub process, then the process which has created the sub process becomes the parent of that sub process and sub process is called child process. A single process can creates more than one child processes. The same thing is applicable to the child processes. For creating a child process, especially in the Linux based system a special system call is used named as fork(). Once the fork system call is executed, it requests the operating system kernel to create sub process and allocates the resources. As mentioned here, there is no limitation on how many sub processes can be created by a single process.

Fork bomb is much similar to one kind of denial of service attack. Fork bomb is nothing but a simple program which replicates itself. It exploits resource allocation mechanism of operating system. In case of the fork bomb, a single process creates as many processes as the operating system can handle. Once it has exhausted all the available resources, operating system will not be able to handle further processes which leads to system hang or sometimes system crash.

Once the process is created, one entry is made in a special structure maintained by the operating system itself called the process table. Since the process table has finite amount of memory, it can only handle that amount of processes. Once it gets filled, system will start lagging. And after sometime system will become completely hanged. You need to restart the system by powering it off. Thus, fork bomb is a kind of process overload attack whose only aim is to affect the performance of the system and to make it unavailable for further use.

As far as the information security says that, in any situation the system and the data stored inside it must remain available as and when needed, among it's three pillars called CIA (Confidentiality, Integrity, and Availability). Fork bomb affects the availability by exploiting resource allocation mechanism.

Linux is considered as one of the most secure operating system. But not even Linux, in the case of fork bomb can survive. The only reason is that process creation is legitimate operation and doesn't require special powers. If you can run any arbitrary code with out special privileges, then you can create fork bomb.

Thus in this paper, we are going to propose a new approach which handles the fork bomb in such a way that system will remain available. This paper is organized into four subsequent sections which contains the main aim of the whole work, existing solutions with it's limitations, proposed solution along with some experimental results respectively. At the end, conclusion is presented along with the future work.

2. EXISTING SOLUTIONS

In this section all the existing solutions to deal with fork bomb is given along with their own limitations.

2.1. Limitation on the Number of Process Creation

In the Linux or Unix based systems, you can set the limit especially the upper limit of how many total number of processes can be created by particular user. You can do this by setting "nproc" in "/etc/security/limits.conf".

This solution is not adaptive as once the upper bound is reached, system will deny all the subsequent requests even though it might be important or legitimate.

2.2. Limitation of Memory Usage per Process

As the name says itself, it puts the memory limitations on the processes. If a process requires more memory than the predefined memory, it will be denied from further execution. The only limitation is that, we can not predetermined the memory requirements, and hence in future, if a process with more memory requirement comes then it will be denied from execution.

2.3. Limitation of Process Creation Rate of Each User

This approach says that process creation rate of each user is counted. And once the creation rate goes beyond the threshold, it will be denied from further execution. If any user creates a single process which in turns create more processes like as happened in more complex computer aided manufacturing and design software, then user can not be able to run this kind of things. Our system will detect it as a fork bomb and execution is terminated.

2.4. Fork Bomb Attack Mitigation by Process Resource Quarantine

This method does not prevent the fork bomb attack, but it tries to mitigate it’s effect. Here, Operating system don’t terminate the bomb processes, Instead of that, operating system will make resource limitations for the bomb processes and it will be checked periodically. Once, it starts behaving like a normal process, resource limitation will be removed and will be allowed to execute as a normal process.

This is a good approach to remove the false positiveness of the previous solution. And according to information security laws, system should remain available for use, either it is a bomb process or a legitimate one. Thus, this solution tried to mitigate the fork bomb and obeys the information security laws.

2.5. Accurate Fork Bomb Detection by Process Name

In this approach, authors said that, instead of using process identifiers or putting resource limitations, just use the name of the processes. Here, two lists is used called detection list and exception list. User can add the name of the process into the exception list, if he or shes wishes that this particular process not to be checked by the system and allowed execution. The name of the processes which are identified as a bomb process will be added to the detection list. Once the name is added in that list, and if in future the process having the same name comes, then without any further check it will be killed.

The only limitation of this method is that once the name of any process is added to the detection list then it will stay there forever. In future, it might be possible that process having the same name as one of the process in the detection list comes, but not a fork bomb process, it will be killed directly. Thus, even though the process is legitimate, system will not allow it’s execution.

3. PROPOSED SOLUTION

The proposed method for efficiently dealing with fork bomb attack is as follows:

Our approach of dealing with fork bomb is combination of the [1] and [2]. As suggested in [2], accurate detection is made. Once done, we took it as base and applied the solution presented in [1]. Reason behind doing this is, once the process name is added in to the detection list, it stays there forever. When in future, process arrives and have the same name but not the fork bomb then solution defined in [2] will directly deny the execution.

Also, if a malicious user create one process having some well known name or name of that process derived from some popular software, then it will be added in the detection list and after that when the legitimate process arrives from which the attacker derived the name, it can not execute itself.

Following figure 1 is the flow chart of the proposed solution. It also shows how the various steps are executed in order to efficiently deal with fork bomb.

As shown in the above figure, there are two flows from which system can pass.

In first step, when process requests fork, then it will be checked in the exception list. If it found in the exception list then without performing any check, it will be permitted for execution. Otherwise, it will be checked in the detection list.

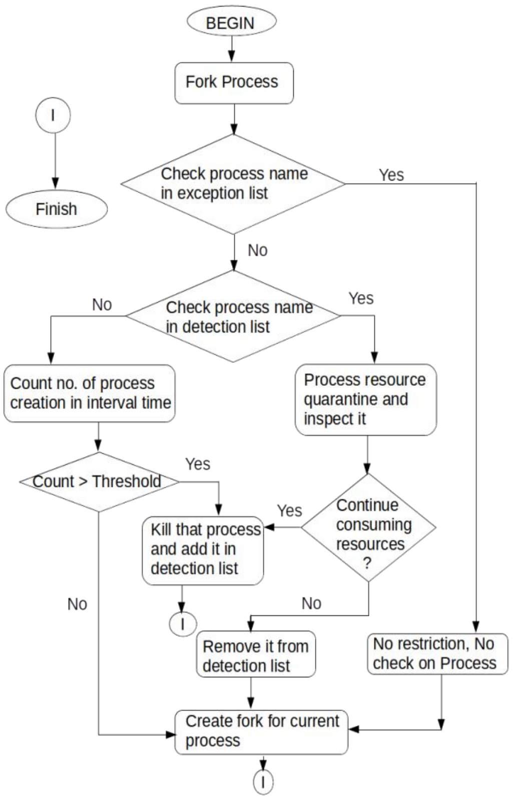
BEGIN
Fork Process
I
Finish
Check process name in exception list
Yes
No
Check process name in detection list
No
Yes
Count no. of process creation in interval time
Process resource quarantine and inspect it
Count > Threshold
Yes
Kill that process and add it in detection list
Yes
Continue consuming resources ?
No
No
I
Remove it from detection list
No restriction, No check on Process
Create fork for current process
I

Fig. 1. Flow of the proposed solution

In second step, let's say the process is not found in the detection list, then the process growth rate is measured in particular interval time and compared with the predefined threshold value. If it exceeds then it's name is added to the detection list and will be killed.

In last step, if match found in the detection list, then instead of directly denying execution, it will be permitted to execute in the resource quarantine manner, where the resource limitation is imposed on the process. Here, it will be checked periodically and if it has normal behaviour then it will be released from the resource limitation and it's name will be removed from the detection list, otherwise, it will be killed.

4. EXPERIMENTAL RESULTS

When you apply fork bomb in the system whether intentional or unintentional, it is clearly visible that before the fork bomb system behaved normally. Once, the fork bomb occurs, the system started lagging. For experimental purpose, we have used bash fork bomb, which nothing but the simple bash shell script. In this shell script, bomb process replicates itself twice. Thus, the process growth can be seen like 1,2,4,8,16,32,64... etc. And all are the do nothing processes. The only task they are doing is the forking of themselves twice. Here, parent will never die. And hence it makes system overloaded.

In the previous solutions, as mentioned above, the process growth rate is measured and if it goes beyond the some predefined threshold, it will be killed. But in some cases, if a user wants to make this happen, then in this case, our proposed solution provides the facility to add that process into the exception list.

System configuration details and parameters that has been applied are as below:

Parameter	Value
FORK_RECORD_DEPTH	500 record
FORK_SPEED_LIMIT	1000 fork/sec
MEMORY_CAPTURE	750 MiB
INSPECTION_INTERVAL	10 msec

Table 1. Parameters and Values

CPU	Intel Celeron
Frequency	2.20 GHz
RAM	1GB
Linux Kernel Version	4.15.0-47-generic
Xubuntu Version	18.04 LTS

Table 2. System Configuration

In order to find fork bomb, our system will check for the fork system call. And will count the number of times the fork system call executed. If it goes beyond the threshold, then it will be added into the detection list. So, for new processes, the time for checking if consumed. Else, it will not check for the same. Hence the time is reduced drastically.

In contrast, to achieve the system availability, it also checks the processes which are already in the detection list for it's validity. Thus, this solution may generate slide system load especially the load on the system memory.

Following table 3 shows the time required to identify the fork bomb with respective to the threshold as shown in the table 1.

Threshold	Time (msec)
500	44
700	124
1000	668
1500	2468

Table 3. Fork Bomb identification time

For finding the load on the system, well known web server cockpit is used. Task manager can also be used which is provided as part of almost all operating systems. Here, in our case, xfce4-taskmanager provides good diagnostic information like memory utilized, load on CPU and utilization of swap space.

Following figure 2 shows the load on the system via CPU Load, Memory Load at the time of attack.

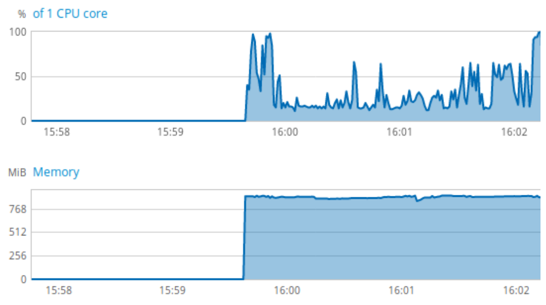

Fig. 2. Load Monitor

As we can see, once the bomb is exploited on the system, our proposed method will check according the flow shown in figure 1. In the above load monitoring case, the process name is already added into the detection list. Hence, without checking the system will kill that processes. Thus, it makes the system available.

5. CONCLUSION

In this paper, we have used the hybrid mix approach as of storing the name of the process for future reference and putting resource limitations. There are several solutions for handling the fork bomb. They have their own limitations. However, proposed method will remove the limitations of the existing solution, it will slightly affects the memory. But that memory requirement is small. Thus, by combining the existing two solutions in order to remove the limitations of each other, we have provided the efficient solution in this paper.

The result of the evaluation experiment shows that, once the process name is added into the detection list, it won't be killed, rather than that, it will be examined under memory limitations. Hence, according the information security laws, the availability is maintained and efficient dealing with fork bomb is done.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their valuable suggestions and the proposed references.

REFERENCES

[1] Gaku Nakagawa, Shuichi Oikawa, “Fork Bomb Attack Mitigation by Process Resource Quarantine”, Fourth International Symposium on Computing and Networking, IEEE 2016.
[2] M. Hareesh, K. Yaswanth, M. Sreeja, Saidalavi Kalady, “Accurate Fork Bomb detection by Process Name”, International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), IEEE 2017.
[3] Mohiuddin Ali Khan, Sateesh Kumar Pradhan, Huda Fatima, “Applying Data Mining Techniques in Cyber Crimes”, IEEE 2017.
[4] Michele Berlot, Janche Sang, “Dealing with Process Overload Attacks in UNIX”, Information Security Journal: A Global Perspective, 2008.
[5] Ashvini T. Dheshmukh, Dr. Parikshit. N. Mahalle, “Survey on Linux Security and Vulnerabilities”, International Journal of Engineering And Computer Science (IJECS), 2014.
[6] Abraham Silberschatz, Peter Bear Galvin, Gerg Gagne. Operating System Principles. 7th Edition. ISBN: 9812531769.
[7] “Why did the command “:(){ :|: & };:” make my system lag so badly I had to reboot?”, accessed on 05 November 2018, https://askubuntu.com/questions/159491/why-did-the-command-make-my-system-lag-so-badly-i-had-to-reboot/.
[8] R. singh, “fork bomb defuser (rexfbd).”, accessed on 01 December 2018, http://rexgrep.tripod.com/rexfbd.htm.
[9] “understanding bash fork bomb,”, accessed on 08 November 2018, https://www.cyberciti.biz/faq/understanding-bash-fork-bomb/.
[10] “limiting user process in kernel.”, accessed on 15 October 2018, http://www.cyberciti.biz/tips/linux-limiting-user-process.html.
[11] Paul Cobbaut. “System Administration”, accessed on 05 October 2018, http://linux-training.be/linuxsys.pdf.