ResMon: Securing Resource Consumption of Critical Infrastructure from Wanton Applications

Emmanuel C. Ogu*
Department of Computer Science, School of Computing and Engineering Sciences, Babcock University, Ilishan-Remo, Ogun State. Nigeria.

Sunday A. Idowu
Department of Computer Science, School of Computing and Engineering Sciences, Babcock University, Ilishan-Remo, Ogun State. Nigeria.

Jean-Paul Ainam
Department of Computer Science, Adventist University, Cosendai, Cameroon.

Ogu Chiemela
Department of Computer Science, School of Computing and Engineering Sciences, Babcock University, Ilishan-Remo, Ogun State. Nigeria.

ABSTRACT
Hackers have devised a recent technique of infiltrating critical infrastructure with wanton applications that gulp at the limited resources possessed by these infrastructure for meeting critical needs and deadlines. Also a reality is the fact that hackers could breach already existing and trusted applications or software on these critical infrastructure and bug them with malicious codes that plunge them into a state of wantonness; consuming limited, critical resources and making none (or insufficient) available for other, equally critical applications that depend on a fair portion of the same resources to meet their deadlines and critical requirements. This development portends the next generation of denial of service (DoS) and distributed denial of service (DDoS) attacks to critical infrastructure, where all that is required is to discover vulnerabilities in already trusted and running applications on critical infrastructure or deliver and escalate new applications on these critical infrastructure and plunge them into wantonness, consuming limited resources and resulting in a denial of service. Proposals already exist in literature that could forestall an occurrence of such attacks, but some of these have not previously been tested; one of such being that documented by [1]. This research is an experimental implementation of the theoretical model proposed in the cited article, in order to test and validate its workability and results. An experimental prototype – codenamed "ResMon" – of the model proposed is built and validated within the Ubuntu Linux operating system environment.

General Terms
Network Security, Denial of Service, Critical Infrastructure.

Keywords
Critical Infrastructure, Computing Resources, DoS, DDoS.

1. INTRODUCTION
[1], proposed a theoretical model for monitoring and securing resource of critical infrastructure in the real-time. The advantage which is argued that this model boasts over such other similar models for achieving the same or similar results – such as those by [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], and [13], to mention but a few – is the fact that it is more resource-centric.

In other words, as opposed to trying to analyse and classify traffic, which is the primary feature of most other models and is argued to have its own implication on the limited, critical resources, this novel model proposed by [1] directly monitors for wantonness and shuts down guilty applications that try to usurp these limited resources.

2. PROTOTYPE DESIGN AND IMPLEMENTATION
An algorithm is given in the proposal for the model, and this is represented further in a functional flow block diagram (FFBD) as shown in figure 1. This FFBD would form the basis and guidelines for building the source codes for this prototype experiment. See Appendix I for guides on how to retrieve the compressed archive (*.zip*)[1] of the experimental codes online.

[1] In order to access the source code archive file, a compatible archive (compression) application software that can open and decompress zip archive files (with the extension: ***.zip***) must be installed on the machine.

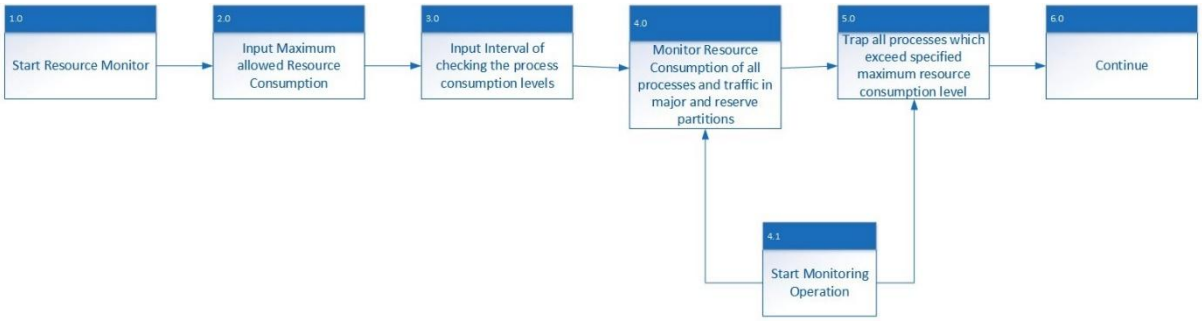

Figure 1: Functional Flow Block Diagram for "ResMon" (Ogu, Idowu, & Adesegun, 2015)

The prototype designed is a proof of principle / concept prototype simply for the purpose of testing the fundamental logic and workability of the theoretical model proposed by [1]. The source code of the prototype is in the C-high level programming language in order that the additional computing overhead imposed by running / executing the prototype program is largely minimal.

Critical computing resources that were monitored for from within the code base of "*ResMon*" were the CPU usage and the Memory usage. Statistics relating to the consumption of these resources were retrieved from the Linux operating environment using the inbuilt Linux "*Top*" command. Basically, in Ubuntu Linux, the "*Top*" command is a built-in command that provides a current overview of processor activity in real time. It displays a list of the tasks, processes and applications that are currently running on the system, and provides an interactive interface (through various commands for specifying different features and parameters for the overview) for manipulating these applications and processes. It sorts the overview of tasks, processes and applications by CPU usage, memory usage and runtime.

The "*Top*" command was called in this case for these needed data so as to reduce the lines of codes that would need to be compiled and computed before "*ResMon*" goes live and real-time.

The prototype was built to function thus:

If application "A" alone, running on the computing infrastructure, begins to consume say up to 60% and above of any the resource provisions in that system; "*ResMon*" traps application A (suspecting that application A may have become wanton) and shuts it down immediately by calling the built-in Linux "*kill*" command which has been programmed into the source code.

The proposed algorithm for the monitoring system given in [1] is enhanced as given in Appendix II for building "*ResMon*"

3. PROTOTYPE TESTING AND RESULTS

The prototype was designed for and experimented within the Ubuntu Linux Desktop operating system (v13) environment. The full specifications for the testing system used for testing the prototype is given in Table 1:

Table 1: Specifications of Testing / Experimental Computer

SPECIFICATIONS	**PC**
Manufacturer	Toshiba
Make	Satellite
Model	P300-20H
HDD Capacity	250GB
Memory	2GB
CPU	Intel® Pentium® Dual CPU T3400 (2CPUs) (~2.16GHz each)
Page File	Unavailable
Bios Version	V3.40
Operating System	Ubuntu Linux Desktop Operating System 32-bit (version 13.0, released 2013)

The prototype system was tested under two different experimental scenarios (no-attack and attack scenarios). This was to ensure test case completeness and effectiveness, as well as dynamism of the testing process. In experiment 1, the prototype was tested for functionality using the PC (see table for specifications); the goal of this experiment was to ensure that the prototype system is able to run in real time and produce results by terminating defaulting applications that exceed the specified threshold as long as the prototype is running. This was essentially a "no-attack experiment". In experiment 2, an Application Layer DoS attack (a *slowloris* attack) was simulated using the *slowhttptest* Application layer DoS attack tool; the goal of this experiment was to observe the behaviour and effects of the system prototype on defaulting applications in an attack scenario.

1.1 Slowhttptest Tool (v1.6)[2]

The SlowHTTPTest tool is a highly configurable tool that simulates some forms of Application Layer Denial of Service attacks, essentially for experimental purposes. This tool works on majority of Linux platforms (Ubuntu inclusive), OSX and Cygwin - a UNIX-like environment and command-line interface for Microsoft Windows.

It can be used to implement most common low-bandwidth Application Layer DoS attacks, such as *slowloris Attack, Slow HTTP POST, Slow Read attack* (based on TCP persist timer exploit) by draining concurrent connections pool, as well as *Apache Range Header attack* by causing very significant memory and/or CPU usage on the server.

Slowloris DoS attacks rely on the fact that the HTTP protocol, by design, requires requests to be completely received by the server before they are processed. If an HTTP request is not complete, or if the transfer rate is very low, the server keeps its resources busy waiting for the rest of the data. If the server keeps sufficient quantity of the resources busy, this can create a denial of service. This tool thus sends partial HTTP requests, aiming to getting a denial of service response from the target HTTP server [14].

After a successful attack scenario and a proper exiting of the slowhttptest tool, the results can be written to a file specified in the input parameters, to contain a detailed report with the parameters and statistics used to orchestrate the attack, as well as some useful information that could be useful to the attacker in orchestrating a more coordinated attack in the future.

1.2 Initialization

"*ResMon*" is designed to display information as shown in the figure above. Information displayed by "*ResMon*" is shown in four (4) columns. In the first column, the process ID (PID) of each running process is displayed. In the second column, the name of the application currently utilising each of the processes is displayed. In the third column, the total memory usage of each running process is displayed with the percentage memory usage (shown in bracket). In the fourth column, the total CPU usage of each running process is displayed with the percentage CPU usage (shown in bracket). See figure 2 in appendix III.

Before launching each time, "*ResMon*" demands for an input each time, percentage upper consumption limits for the Memory and CPU consumption respectively for any wanton application / process.

1.3 Experiment One

- *Step 1*: A random application was opened (Mozilla Firefox web browser in this case, and the resource consumption levels of the application was noted.
- *Step 2*: For experimental purposes, application – Mozilla Firefox – was made a culprit (a defaulting application). The threshold limits of *resmon* was lowered to 19% (for memory consumption) and 25% (for CPU consumption). This made Mozilla Firefox a culprit application by virtue of its memory consumption, as it was trying to download a large file from the Internet while streaming an online video in High Definition, which eventually exceeded the specified threshold limit along the line. See figure 3 in appendix III.
- *Step 3*: This made the application – Mozilla Firefox – to be shut down in less than one second (exactly 0.27s).

1.4 Experiment Two

- *Step 1*: Successfully compile and install the attack tool (see http://code.google.com/p/slowhttptest/wiki/InstallationAndUsage for installation and usage instructions and guidelines). At the time of conducting this research, this tool is a freeware and is free to use for experimental, non-profit purposes. Also ensure you have a webserver (Apache in this case) installed and running.

  Start the Apache service using any of the following commands

```
$ sudo apache2ctl start
      or
$ sudo start apache2
      or
$  sudo  service  apache2  start
      or
#    /etc/init.d/apache2    start
      or
$ sudo /etc/init.d/apache2 start
```

- *Step 2: Configure the attack tool using choice of* parameters as shown in the parameter table above. During this experiment, the following parameter was fed into the attack tool:

```
slowhttptest -c 100 -t ATTACK -x 204 -i
   10 -l 5000 -p 3 -g -o attackinfo
```

See figure 4 in appendix III.

This attack was simulated against the resident apache server on the machine via the localhost address.

- *Step 3*: The resource consumption rate of the attack application was observed by viewing the "*Top*" command. The percentage CPU consumption was 96.3%, and the percentage Memory consumption was 0.1%. The CPU consumption rate was relatively high.

It may be noted and argued that the parameters with which the slowhttptest tool was bombarding the apache server may, in comparison to real-life scenarios, be too insufficient to cause any DoS on a real-life machine. This is a fact of truth. However, the CPU consumption of the attack tool was observed to be increasing at a rather steady rate, rising to 99.6% within 5 seconds apart.

- *Step 4*: The resource monitoring system prototype – "*ResMon"* was started up with threshold inputs of 50% for Memory usage and 115% for CPU usage, in order to forestall the imminent DoS caused by the wantonness of the attack tool.
- *Step 5*: Observation continued until the slowhttptest tool process was killed by "*ResMon"* as soon as its CPU resource usage exceeded the set threshold for CPU usage. See figure 5 in appendix III.

[2] Could be retrieved online from the programmers' website.

Experiment Two lasted for a cumulative time of ~805 seconds. Because the attack tool was terminated abruptly by *"ResMon"*, it was unable to write the statistics / report to file as is expected. Hence, the file – *attackinfo*, which is supposed to carry the statistics / detailed report of the attack experiment – was created, but remained empty at the end of the experiment.

4. EXPERIMENTAL SUMMARY, FINDINGS AND AREAS OF IMPROVEMENT

"ResMon" has been shown to perform well in both a non-attack scenario as well as in an attack scenario. In the non-attack scenario, *"ResMon"* was able to detect and shut down an application (Mozilla Firefox) that was shown to be consuming resources above the preset threshold. This suggests that *"ResMon"* is able to detect and prevent a DoS that may result from Flash Crowd[3] activities when the resource consumption of certain applications that are being accessed by the flash crowds begin to increase dramatically.

In the attack scenario, *"ResMon"* was able to shut down, in real-time, an attacking application that was going to result in a DoS in the process of trying to attack the resident web server with slow-connection-packets.

Throughout the entire experimental process, the resource usage of the system prototype never exceeded 1% for both CPU and Memory. This could be a plus for the proposed model and/or the prototype system.

"ResMon" is configured to run in real time (< ~1s) (*see the source code in Appendix I*). All observations and actions documented in this research occurred within the specified real-time value of (<=1s). This real-time value can be altered from the source code by changing the parameter for the "sleep" command (measured in seconds).

In line with this, it confirmed from the two test cases of attack and no-attack that *"ResMon"* is always able to *effectively* shut down defaulting applications, processes and tasks that exceed the preset threshold limit by as little as 0.01%. It is also proven that *"ResMon"* is able to *efficiently* execute and deliver results based on its requirements within stable time frame (<1s), possibly owing in part to its compact (~600 LOC) and code-optimized nature.

The prototype application could also be experimented with in future researches using a low level programming language such as the Assembly Language, in order to also investigate and compare its computational behaviour under such an instance.

A runtime malfunction was observed during the course of experimenting on *"ResMon"*. It was noticed that after being left to run for a period of time (~7secs), *"ResMon"* shut itself down automatically and dumped to disk. This malfunction was traced to a memory allocation function within the source code that could not be resolved as at the time of documenting this experiment. This memory allocation malfunction could arguably be as a result of runtime memory allocation and re-allocation privilege insufficiencies which may have arisen because *"ResMon"* didn't have enough privileges needed for handling memory allocation and re-allocation above, beyond or about a possible restriction which may have been imposed by the Operating System that could not be ascertained. A workaround for this was found in breaking down the attack experiment into phases / sections (*initialization, pre-attack, attack, monitoring and reaction*). Future research would seek to look towards improvement, enhancement and optimization of *"ResMon"* in a bid to overcome this malfunction.

The resource monitoring prototype system was system was only tested against the Application-layer DoS attack, without any form of distribution in the attack coordination. Future researches would consider experimenting with the prototype in simulated scenarios for other forms of DoS attacks with some level of distribution in the attack coordination.

5. CONCLUSION

The theoretical model proposed by [1] proved laudable within the limits and constraints of this experiment. However, some improvements remain necessary for this proposed model to be able to deliver commendable results with real-world critical infrastructures. Basically, the prototype of the model was able to shut down wanton applications in the attack and no-attack experiments, which were headed to cause a DoS on the computing system. In addition, it is recommended that the resource pool partitioning suggested by [16] may go to a large extent in enhancing the operations of the model, while a way of escalating the permissions level of the model and its prototype may also further boost its effectiveness.

This research has been able to shed light, however, on the imminence of a next generation form of DoS attacks that may not necessarily require the taking over of other machines in order to increase the strength and impact of the attack, as is common with traditional forms of these attacks, but these can actually be orchestrated from within the critical infrastructure itself by plunging already trusted, highly privileged resident applications into a state of resource wantonness.

6. REFERENCES

[1] Ogu, E. C., Idowu, S. A., & Adesegun, O. A. (2015). A Theoretical Model for Real-Time Resource Monitoring for Securing Computing Infrastructure against DoS and DDoS Attacks. International Journal of Advanced Research in Computer Science, 132-136.

[2] Paxson, V. (1998). Bro: a system for detecting network intruders in real-time. Proceedings of the 7th USENIX Security Symposium. 7, pp. 1-22. San Antonio, Texas, USA.: USENIX Association Berkeley, CA, USA.

[3] Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., & Shenker, S. (2001, February). Controlling High Bandwidth Aggregates in the Network. ACM SIGCOMM Computer Communications Review, 32(3), 62-72.

[4] Gil, T. M., & Poletto, M. (August, 2001). MULTOPS: a data-structure for bandwidth attack detection. Proceedings of the 10th USENIX Security Symposium, (pp. 23-38). Washington, D.C., USA.

[5] Yau, D. K., Lui, J. C., & Liang, F. (2002, May). Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles.

[3] **Flash Crowds** are a phenomenon that occurs when a large crowd of legitimate users try to gain access to a server resource, application or service at the same time [15]. Flash Crowds can also cause DoS to occur, and in fact go a long way to further complicate the task of detecting and controlling DoS attacks.

Proceedings of IEEE International Workshop on Quality of Service (IWQoS), 29-41.

[6] Peng, T., Leckie, C., & Ramamohanarao, K. (2003a). Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring. The University of Melbourne, Australia, Department of Electrical and Electronic Engineering. Victoria 3010, Australia: ARC Special Research Center for Ultra-Broadband Information Networks. Retrieved January 30, 2014, from http://www.cs.mu.oz.au/~tpeng/mudguard/research/detection.pdf

[7] Peng, T., Leckie, C., & Ramamohanarao, K. (August 2003b). Protection from Distributed Denial of Service Attack Using History-based IP Filtering. The University of Melbourne, Australia, Department of Electrical and Electronic Engineering. Victoria 3010, Australia: ARC Special Research Center for Ultra-Broadband Information Networks. Retrieved January 31, 2014, from http://ww2.cs.mu.oz.au/~tpeng/mudguard/research/icc2003.pdf

[8] Verkaik, P., Spatscheck, O., Van der Merwe, J., & Snoeren, A. C. (September 2006). PRIMED: Community-of-Interest-Based DDoS Mitigation. Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense (pp. 147-154). New York, NY, USA: Association for Computing Machinery.

[9] Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., & Knightly, E. (2009, February). DDoS-shield: DDoS-resilient scheduling to counter application layer attacks. IEEE/ACM Transactions on Networking (TON), 17(1), 26-39.

[10] Xie, Y., & Yu, S.-Z. (2009, February). Monitoring the application-layer DDoS attacks for popular websites. IEEE/ACM Transactions on Networking (TON), 17(1), 15-25.

[11] Walfish, M., Vutukuru, M., Balakrishnan, H., Karger, D., & Shenker, S. (2010, March). DDoS defense by offense. ACM Transactions on Computer Systems (TOCS), 28(1), Article No. 3 (54 pages). doi:10.1145/1731060.1731063

[12] Das, D., Sharma, U., & Bhattacharyya, D. K. (2011). Detection of HTTP flooding attacks in multiple scenarios. Proceedings of the 2011 International Conference on Communication, Computing & Security (pp. 517-522). Rourkela, Odisha, India: Association for Computing Machinery New York, NY, USA. doi:10.1145/1947940.1948047

[13] François, J., Aib, I., & Boutaba, R. (2012, December). FireCol: A Collaborative Protection Network for the Detection of Flooding DDoS Attacks. IEEE/ACM TRANSACTIONS ON NETWORKING, Volume 20(Issue 6), 1828-1841.

[14] Sergey, S. (2013, November). slowhttptest - Application Layer DoS Attack Simulator - Google Project Hosting. Retrieved from Google Projects: http://code.google.com/p/slowhttptest/

[15] Peng, T., Leckie, C., & Ramamohanarao, K. (2007). Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Computing Surveys (CSUR): Article No. 3, 39(1). doi:10.1145/1216370.1216373

[16] Ogu, E. C., Alao, O. D., Omotunde, A. A., Ogbonna, A. C., & Izang, A. A. (2014, October). Partitioning of Resource Provisions for Cloud Computing Infrastructure against DoS and DDoS Attacks. International Journal of Advanced Research in Computer Science, 5(7).

7. APPENDIX I

Online download links of the source code files for the experimental prototype:

Download the files at:

https://drive.google.com/file/d/0B7sfblJHNzz-TTBXeVFJMXpobEk/view?usp=sharing

For issues or enquiries, contact the corresponding author of this research at: ecoxd1@yahoo.com

8. APPENDIX II

The algorithm for this prototype system is given below:

```
Program:

Step 1:      Start

             While (true)

Step 2:      Get list of all running processes (application). This

             includes:

                    - The PID (Process ID) of each running process

                    - The name of the program executed by the process

                    - The percentage memory usage of the process

                    - The percentage CPU usage of the process.

Step 3:      Store this list in a dynamic array list called

             process_array
```

```
Step 4:     Update process_array after #var interval (specified
            above)
Step 5:     While (true)
            /* Implement a wait until the #var interval time has
            passed */
            - Set variable waiting_time = current time (Get the current system time)
            - Set variable i = #var interval + current time;
                  If (#var waitingtime < #var i)
                        #var waitingtime++;
                        Continue
                  Else  // When #var interval time has passed
                        For each element in the process_array do
                              4.1. Get the process id: pid
                              4.2. Get percentage the CPU
                              consumption: cpuusage
                              4.3. Get percentage memory usage:
                              memoryusage
                              4.4. Compute the overall resource consumption of each
                              process: resourceconsumed
                  If (resourceconsumed > #var consumption)
                  - End the process using the system call: System
                  (kill (pid));
                        End if
                        End for
                        - reset #var waitingtime to current time;
                        - reset #var i to #var interval + current
                        time;
                        - Go back to step 5
                        Break;
                  End if
            End while
      End while
Stop
```

9. APPENDIX III

Experimental images of the prototype

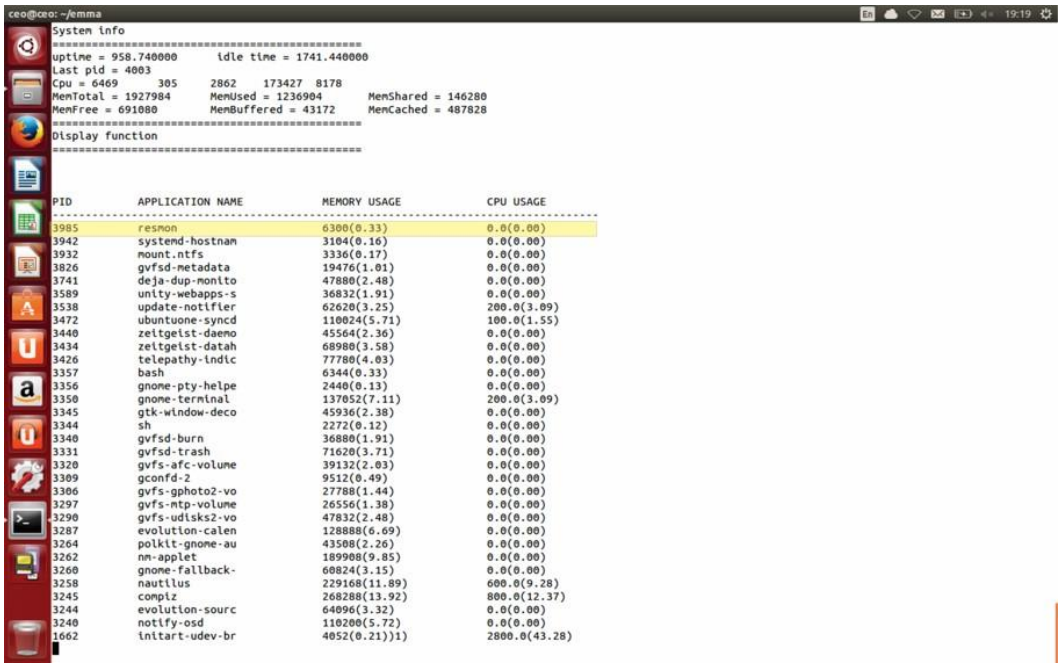

Figure 2: Figure showing "ResMon" running

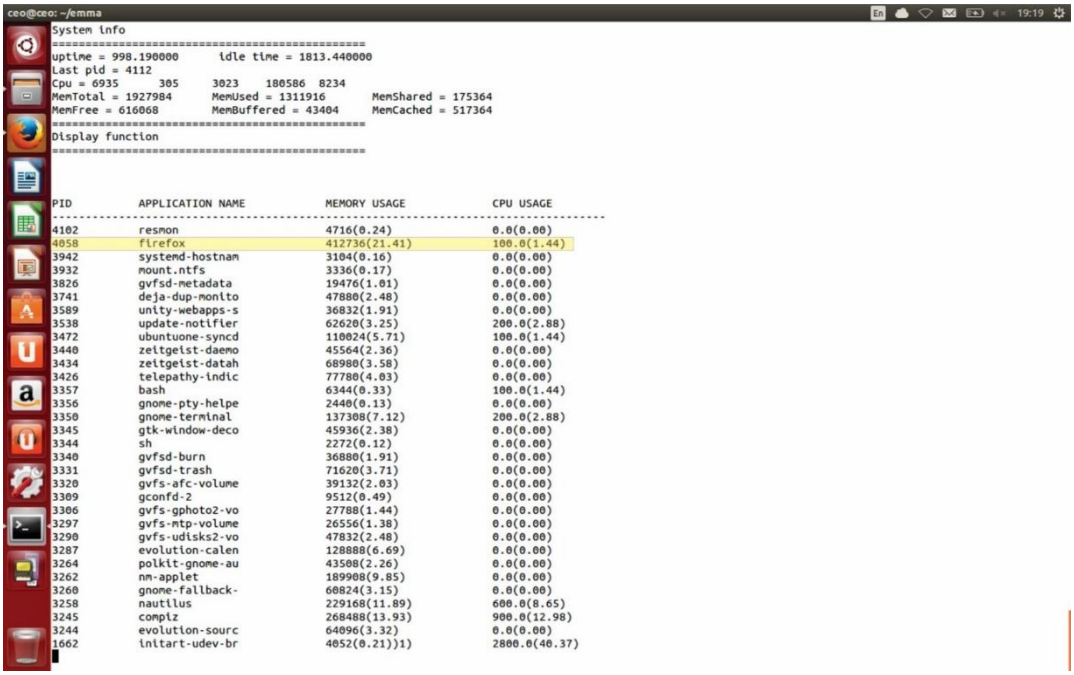

Figure 3: Figure showing resource consumption rate of Mozilla Firefox shortly before it was shut down by "ResMon"

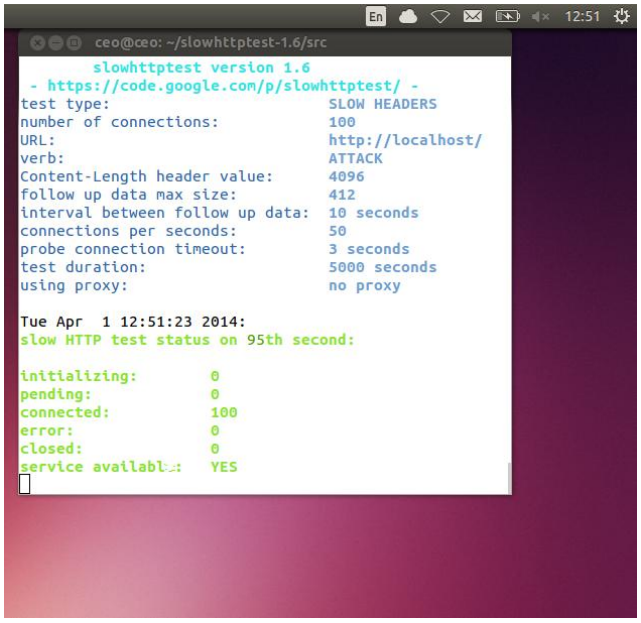

Figure 4: Figure illustrating parameters used to configure the slowhttptest tool for the experimental attack

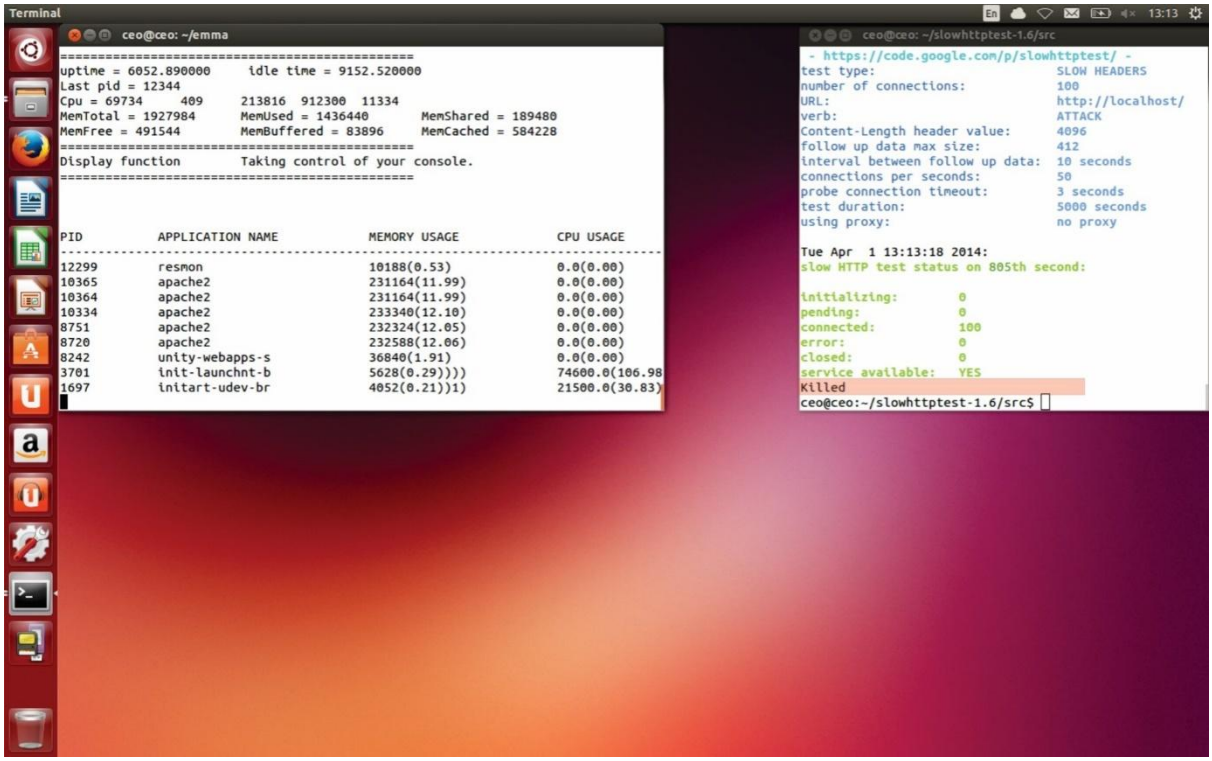

Figure 5: Figure showing that the slowhttptest attack tool is "killed" by "ResMon"