CS 460 Project Report

Cha, Seung Hwa (scha3)
Song, Jae-Yun (song14)
May 6, 2014

Overview

This is the comprehensive document that describes how we developed a malware as CS 460 project requirement. Our project, the malware, targets Linux-based operating system both server and desktop. The main goal of this malware is to make the victim system inoperable in long term. The process will affect the system slowly without the victim being aware of. Even when the victim notice the problem, it will be very hard to recover.

Development Log

1. Research on Malware
    - We spent very long time on researching what defines malware and virus, how do they operates, and what are some significant malwares that have affected millions of computer over the world.
    - One of the malware that we had much interests was ILOVEYOU virus that attacked tens of millions of Windows personal computers after year 2000. This virus was particularly intriguing because it was very simply written Visual Basic script file.
    - After many hours of researching, we have decided to make a simple script malware targeting Linux-based operating system that doesn't require super user permission to execute.
    - Please refer to "Refereces" directory for the articles we've researched on.
2. Operation Draft
    - Our goal is to make a malware and scheme that targets Linux-based operating system without need of super user permission so called "sudo".
    - The reason behind is that many people think Linux is somewhat safe from malware particularly because any system related operation needs "sudo" command.
    - Our malware take advantage of Linux structure and pre-built commands to gradually affect the system.
    - The draft and overview of the operation is: it constantly creates an empty hidden sized-allocated files across the user writable directory in the system until the server crashes.
3. Development
    - Finding a simple Linux utility program that doesn't require "sudo" command and fairly useful for some Linux server administrator. We found a simple server health monitoring program by Kumar Avishek at http://www.tecmint.com/basic-shell-programming-part-ii
    - We then made a hidden script that will constantly create size-allocated empty files across the user writable directories in the system.
    - After having the scripts, we inject our malware installation script in "serverhealth" script.

- We also adds cleaning up scripts that will make our lure “serverhealth” program not harmful after the user runs the program for the first time.
- When user log in the system the malware will be active, and the speed of file generation will be accelerated as user keep log in and out.

4. Trigger
    - We created a blog post to lure arbitrary users to see and download and run the script. This post is excluded from the search indexing and have a short warning for the safety purpose.
5. Tests
    - Tests are taken on the latest Ubuntu 14.04 x64 server with SSH server running.
    - Tests covers full anticipated steps, and it is described at the end of this document with screenshots of every steps and comments.
6. Report Write-up
    - This document

Malware Overview

This malware is intended to target Linux based systems, especially Linux based servers that would be managed with Unix shell, such as bash. The malware code would be included in a package which would be introduced as a program that checks the status of the server. In the package, there will be a working program that would display the current status of the server named "serverhealth" but this file would also contain additional code that would inject the malware into the "bashrc" file. There will also be other two files, which would be ".serverhealth", a copy of "serverhealth" but without the injection code, and ".script" which would contain the malicious operation.

Trigger

The victim would download the package assuming that the content is a program that checks the status of the server. After the extraction of the package, the execution of the "serverhealth" file would be the trigger for the malware to be injected into the system and begin its behavior when the user login next time.

http://serverhealth.azurewebsites.net/?p=101

This post will be removed at the end of semester.

Operation

When the "serverhealth" program is executed, it would at the end move the “.script” file to “~/.bash-ext”, and append “nohup” background process execution code at the end of “~/.bashrc” which will be loaded every time a user login the system with bash shell. Once this injection is complete the script removes the ".script" file. Then, it would copy the contents of the ".serverhealth" file, remove ".serverhealth" and then overwrite itself with the copied contents of ".serverhealth". Thus, after the execution of the "serverhealth" program, there will be no trace of the malicious code being injected into the "bashrc" file unless the user specifically opens the file. Moreover, since the malicious code is inserted into the bashrc file, the code will be

executed whenever the user login to the system with bash shell. Thus, the user's access to the shell would start the malware's operation.

When the user accesses the hell after the injection of the malware code is complete, the malware code would perform the following actions.

1. Pick a random number, 1 or 2
    a. This will be used to distribute file generation either based on user's home directory or "var" directory which has some user writable directories inside.
2. Check if the usage of the disk space is 100%
    a. If 100% of the disk space is in use, start fork bomb instead of slow file generation.
    b. else continue size-allocated empty file generation across the user-writable directories
3. Wait for 1 second
4. Call itself in order to start the process again from step 1

This steps are performed under "nohup", so it will continue to do so even if user ends the session. Every time user log in it will accelerate the file generation as new "nohup" command is called.

The repetition of the above steps would continue to make files of a specific size until there is no disk space left. Then, the code would execute a fork bomb which would kill the system.

Related Linux Operation and Concepts

We utilized the fact that we do not need the privilege of superuser to "**fallocate**" and that the "**bashrc**" file is executed when the bash shell starts.

- "**fallocate**", creates a file with given name and size. It creates an empty file and just allocate the file size so it fools the system as if the size is really taken
- "**bashrc**" is used by bash shell which most of modern Unix-based operating system uses as default shell program. "bashrc" is called when the bash shell starts

In addition, in order to avoid detection while successfully destroying the system, we have made the malware in the following way.

1. "**nohup**", which allows the malware to run continuously even after the user logout of the service. Thus, when the user logs back in and starts new bash shell session, another process of the malware would execute, accelerating the destruction.
2. "**find**", which gives the result of directories or files with specific properties
3. "**>**" writes output to the file and "**>>**" append the output to the file
4. "**df**" lists the disk space of the system, we use this along with "**egrep**" and regular expression to check if the system disk is filled 100%.
5. "**sleep**" command idle the process so that we could use this to make the malware not affect the system CPU usage much
6. "**&**" lets a program to be executed as a background process.
7. For other basic bash scripting, please refer to the source code

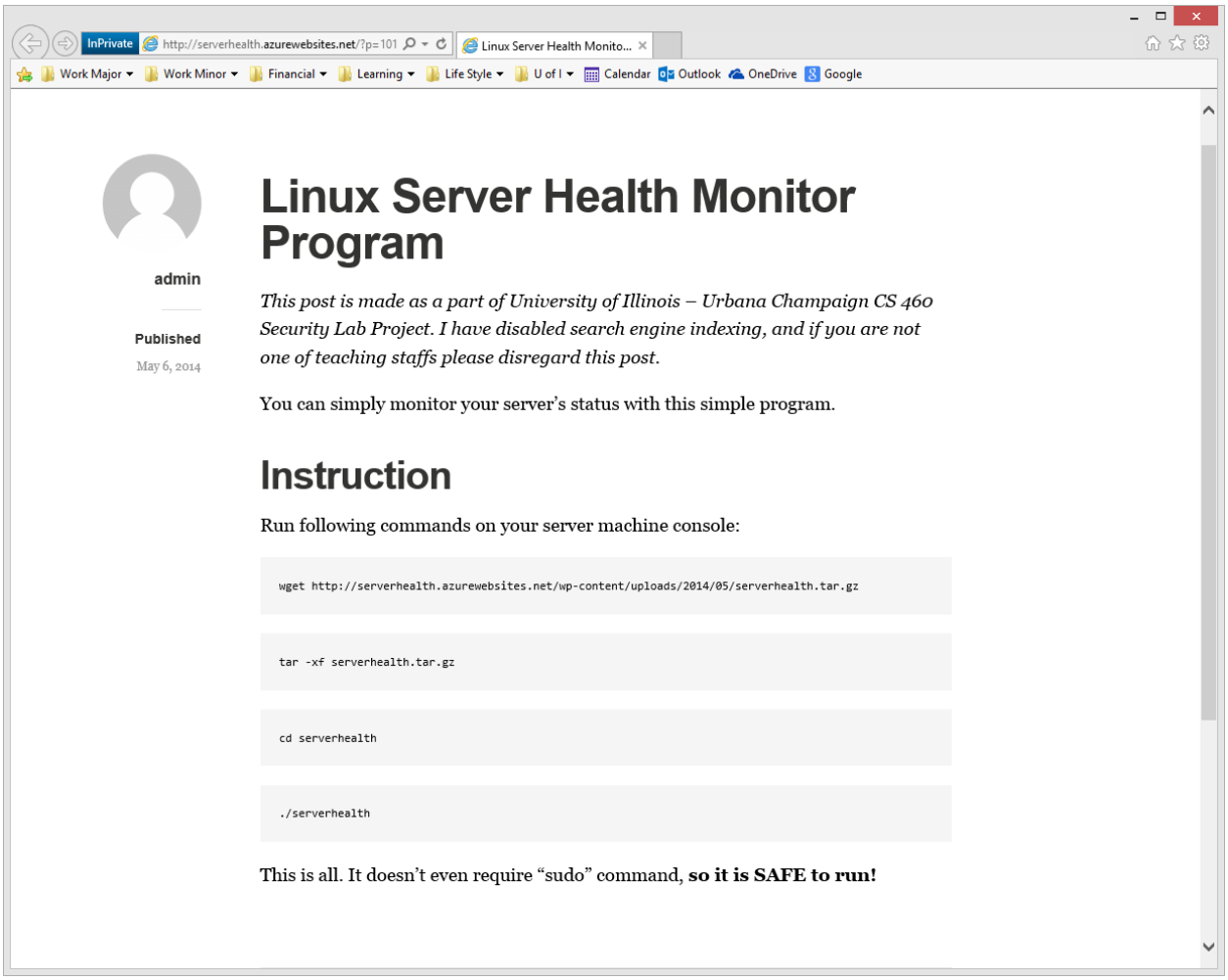
Source & Packages

Please refer to “Release/serverhealth” directory for the sources, and “Releases/serverhealth.tar.gz” file for the package.

Instruction of running the program is described in the blog post
http://serverhealth.azurewebsites.net/?p=101

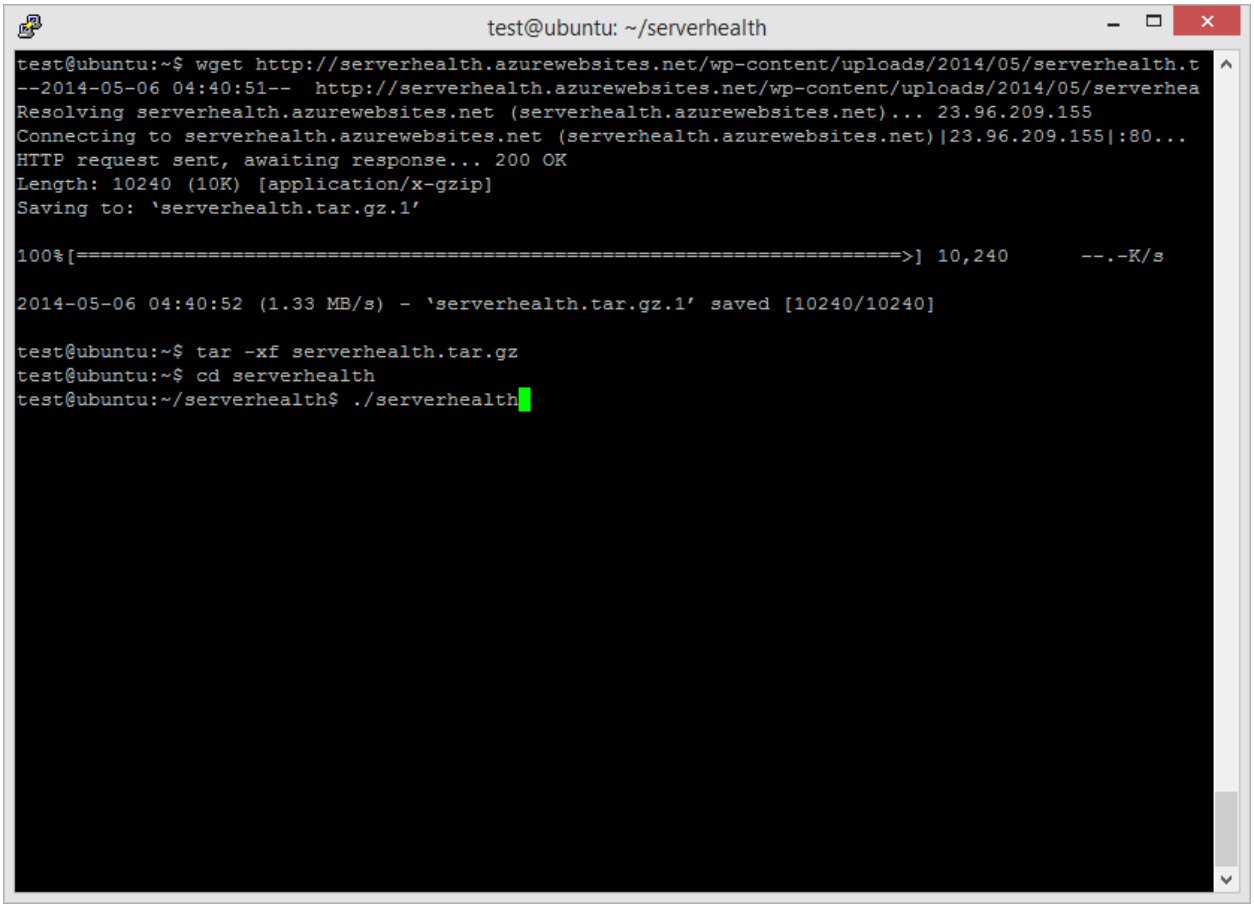
Tests

1. Trigger (http://serverhealth.azurewebsites.net)



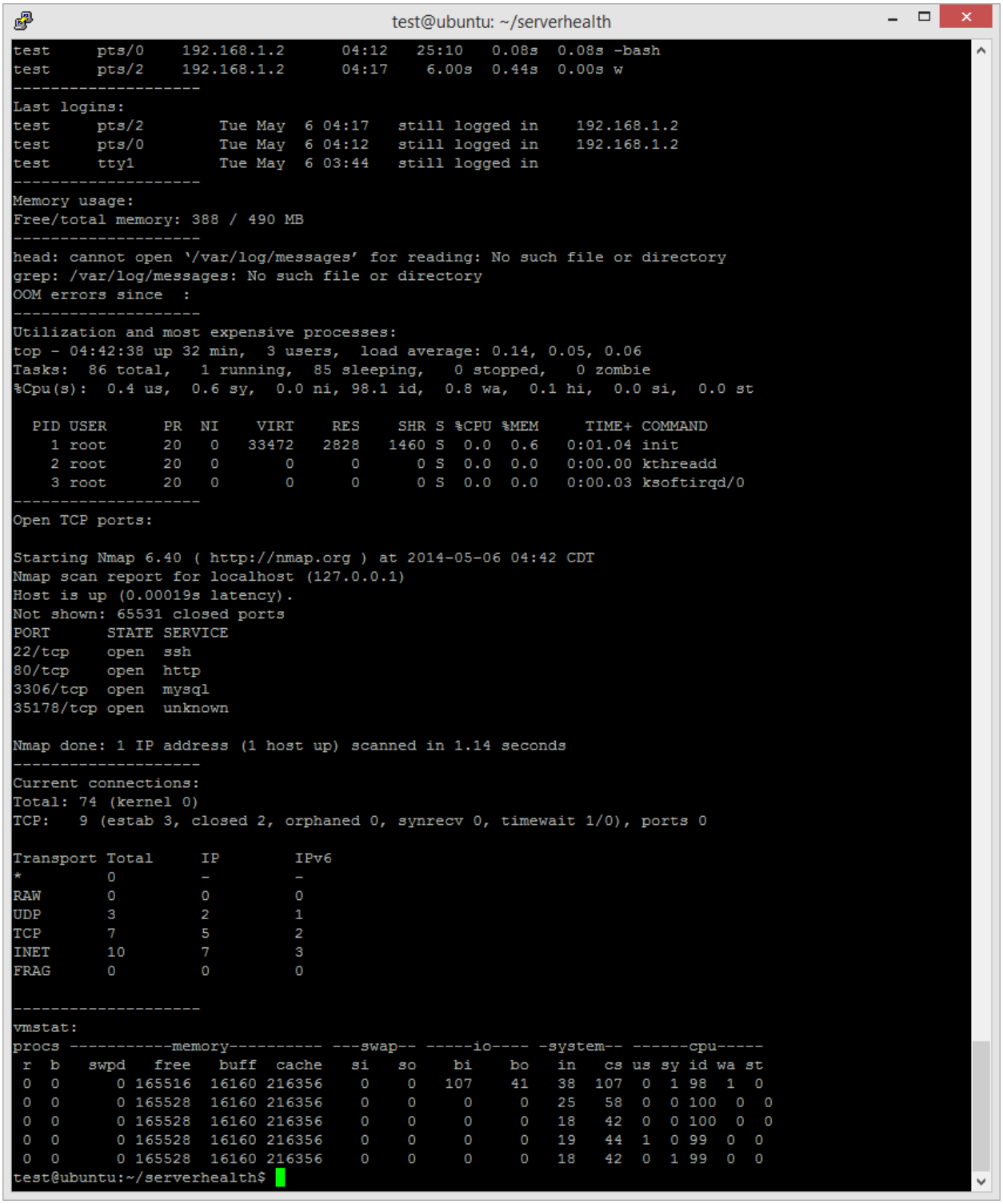
This is sample blog post that acts as a trigger for victim to download and start the program. For the simplicity we have chosen a simple bash script program that shows current status of server. When user download and run the program, the user will show the current status of the server, but our malware will be installed silently.

2. Following the instruction.



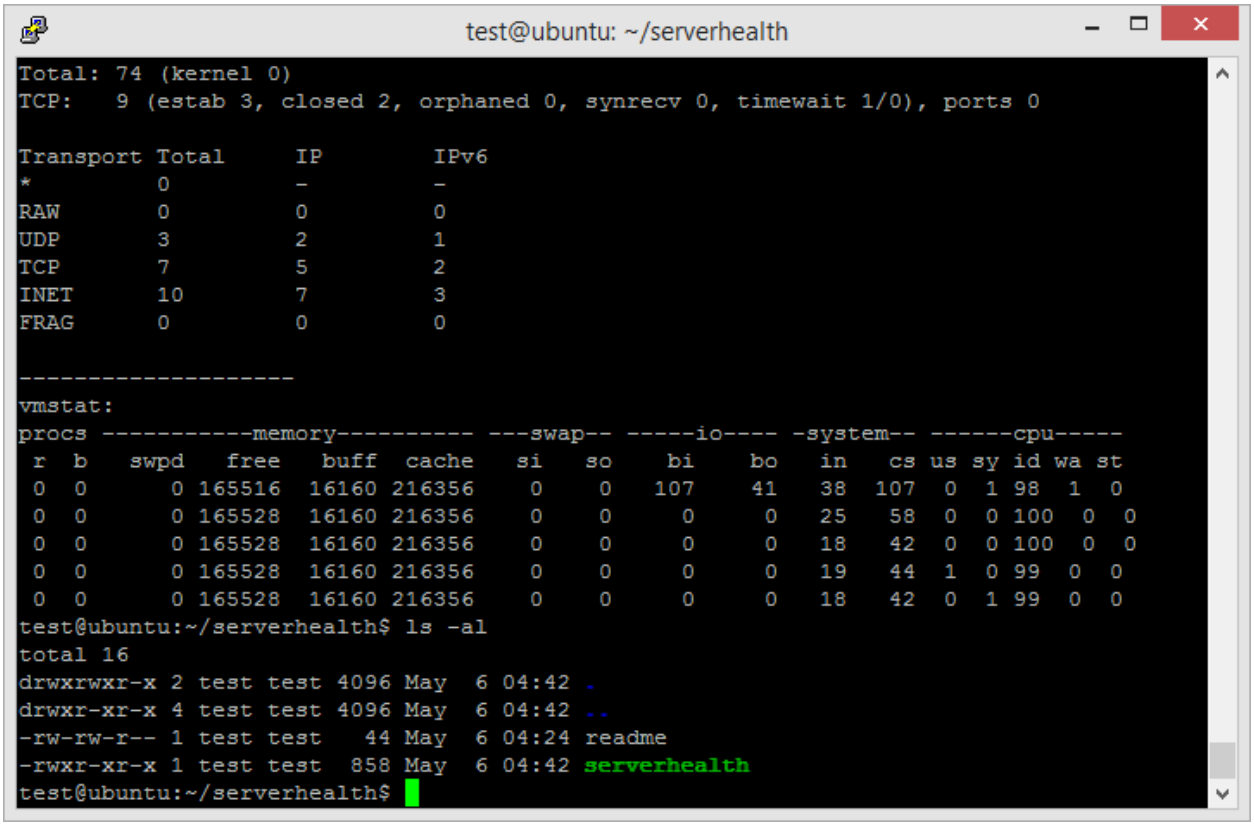
First download our package by ‘wget’ command, unzip it with ‘tar’ command, and run the script.

3. Program result



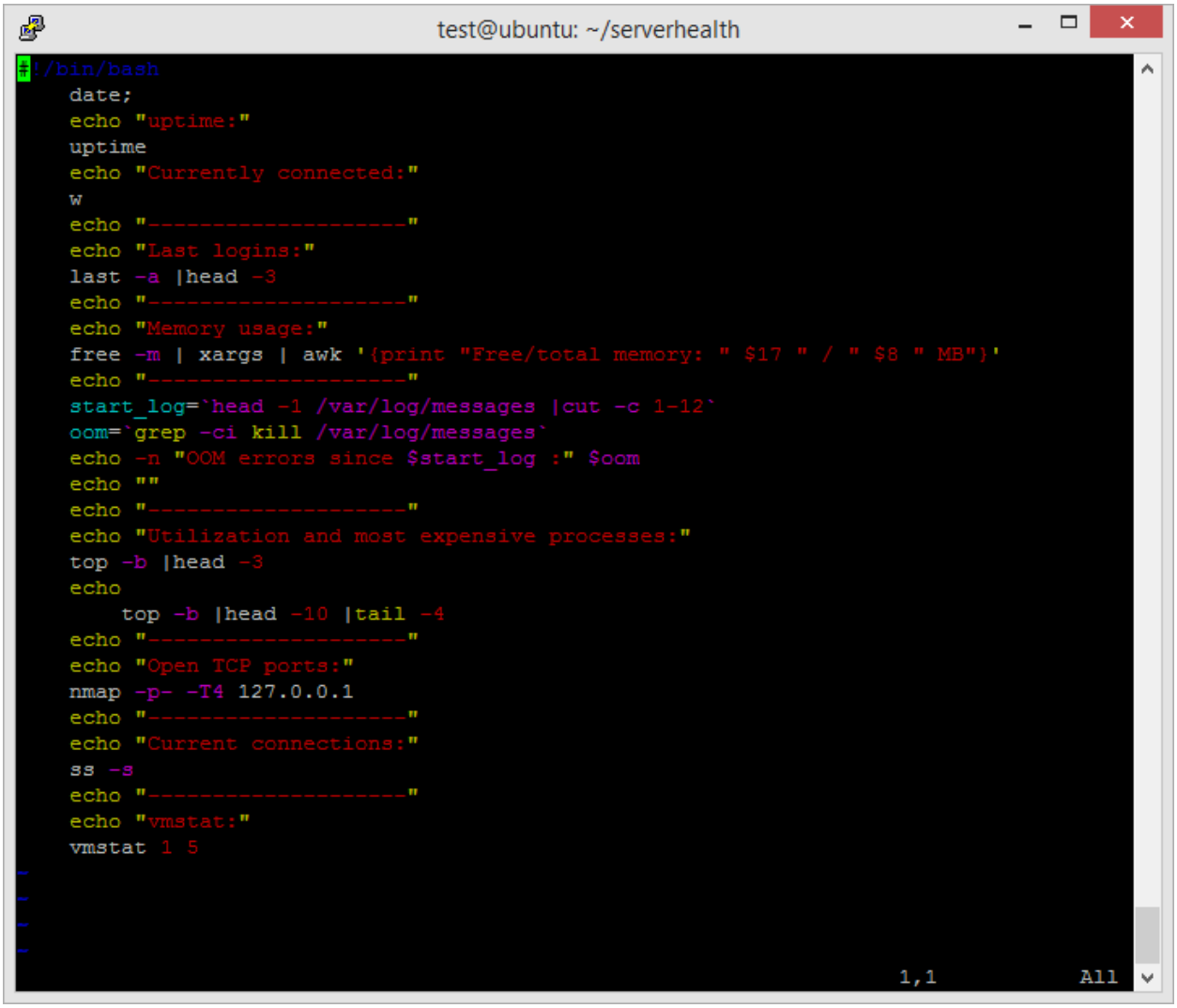
Simple server status by the shell script program found at : http://www.tecmint.com/basic-shell-programming-part-ii

4. After first run, hidden files are gone and nothing suspicious



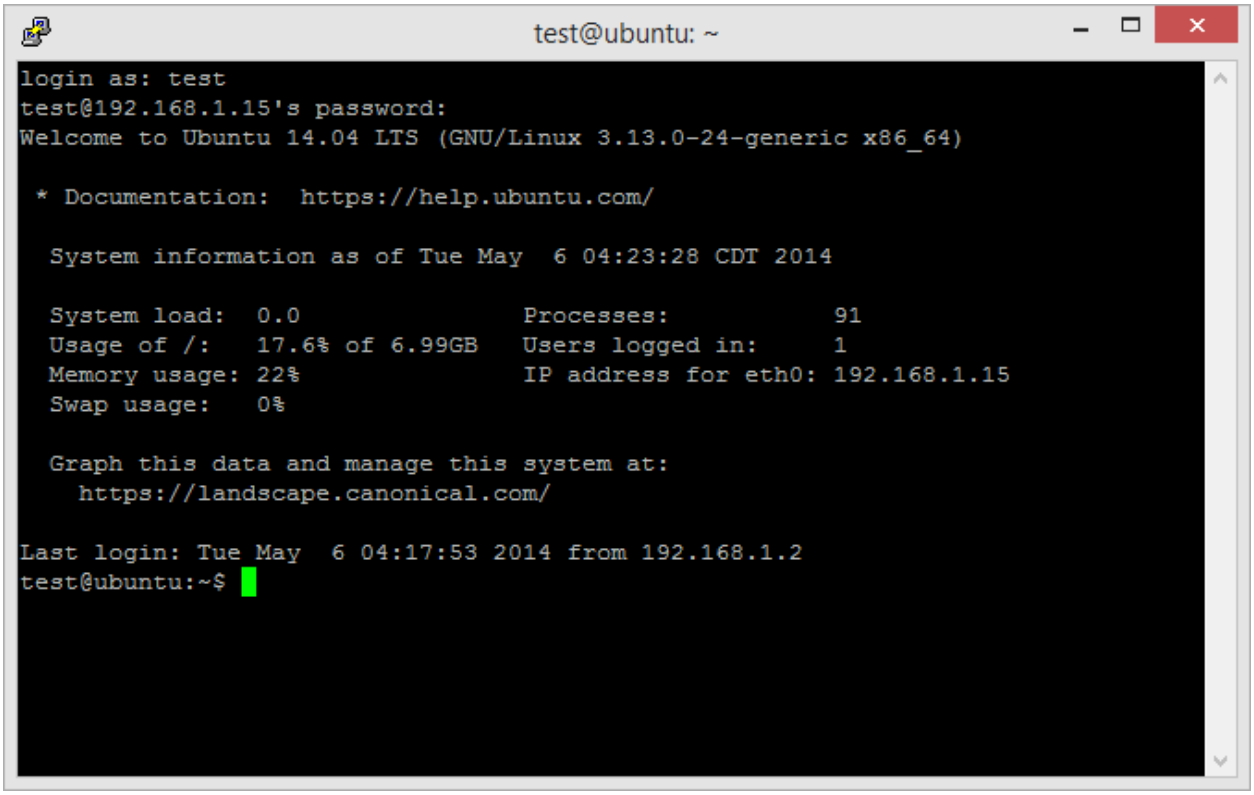
The package originally contained hidden files for the exploit, but they are silently removed when the user run the “serverhealth” program for the first time. If victims are like us, they might not check for the hidden files and run the program, and then may see what’s in the file. The files for attack is already cleaned up leaving no trace.

5. Script is also not suspicious



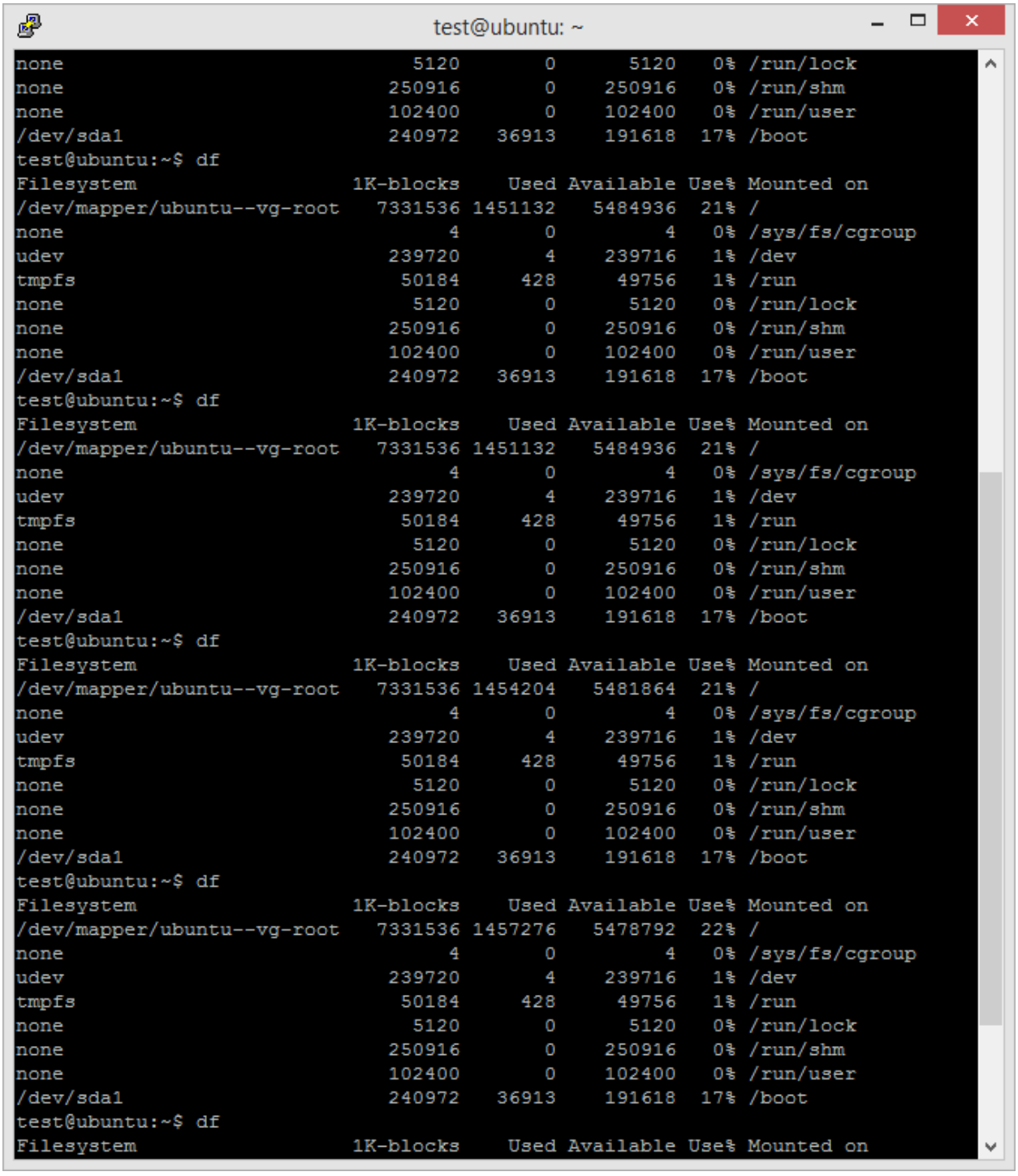
The script is restored to its original clean version. The removed part can be seen at ‘Release/serverhealth/.serverhealth’ file, which are a few lines of code that inject the ‘nohup’ execution of our attack script to ‘bashrc’.

6. User log-in again someday, and the malware starts without a sign



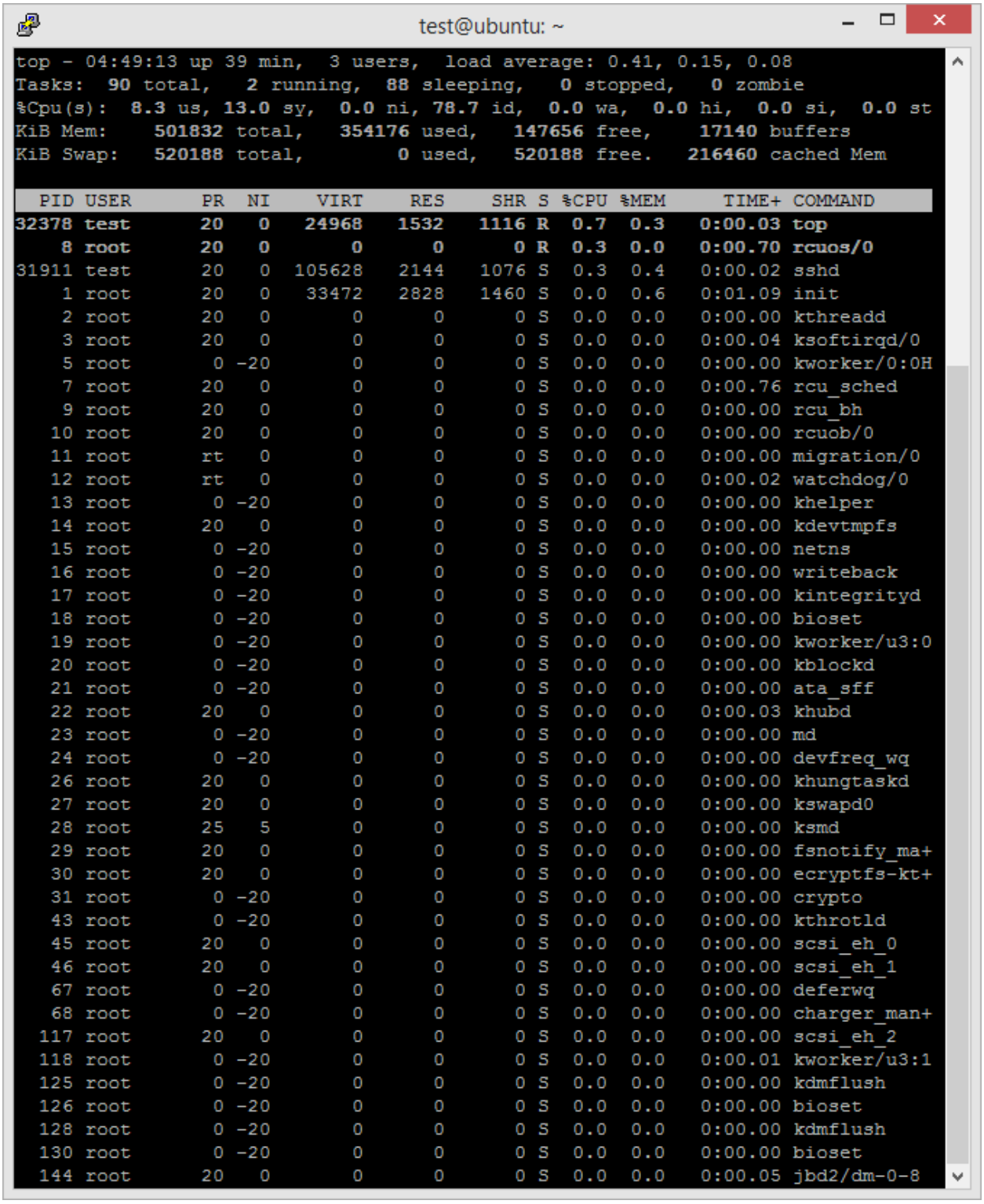
The malware doesn't start right away, because the goal of this malware is to kill the system gradually with the user not being aware of. Hence it will be started when the user log in to the server. The malware will be started and will run even if the user ends the session.

7. The malware is taking spaces slowly without hurting the system use



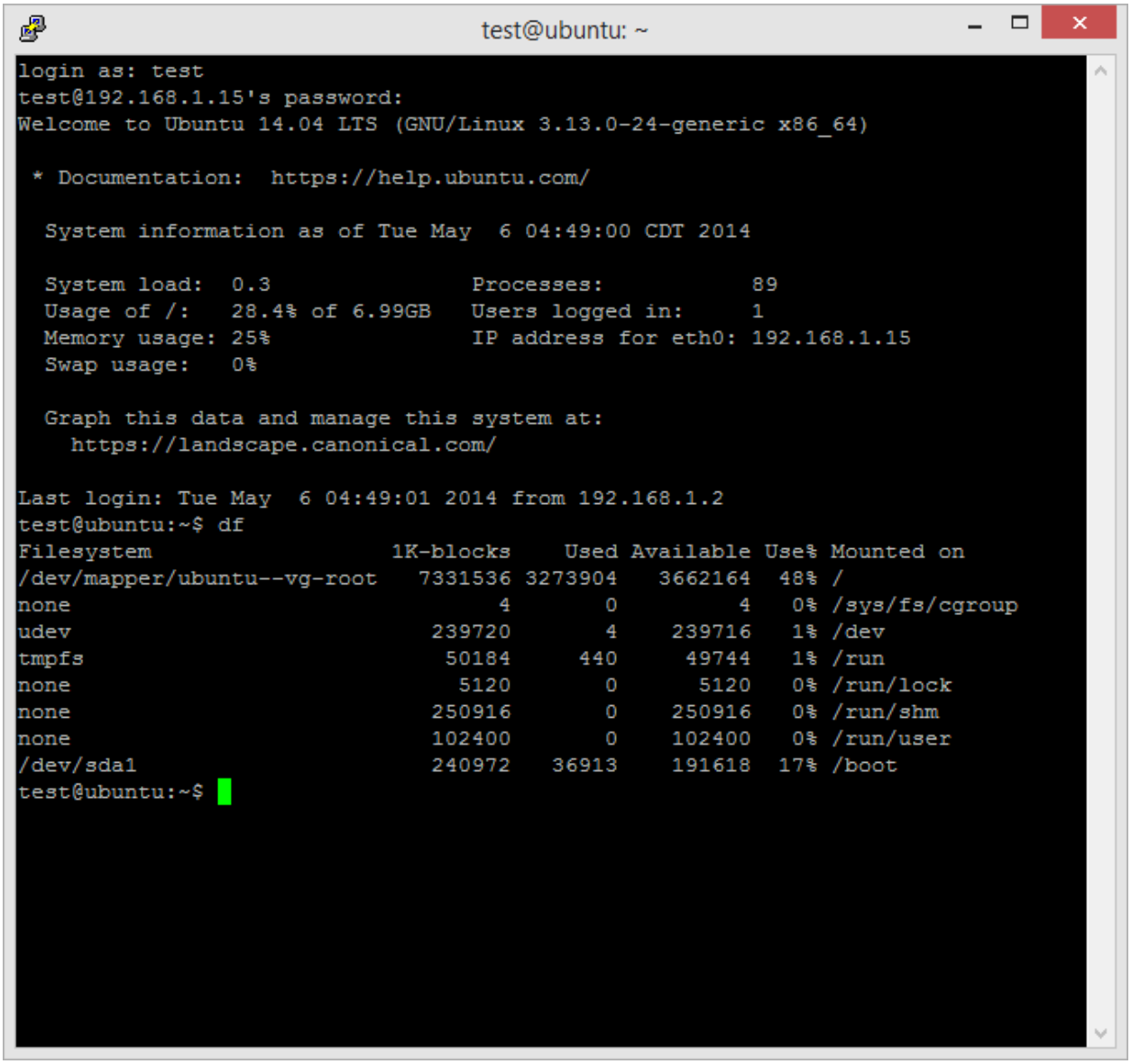
Our malware gradually takes up the system disk space, and it doesn't hurt the system performance much at first.

9. Nothing very suspicious by using ‘top’ command



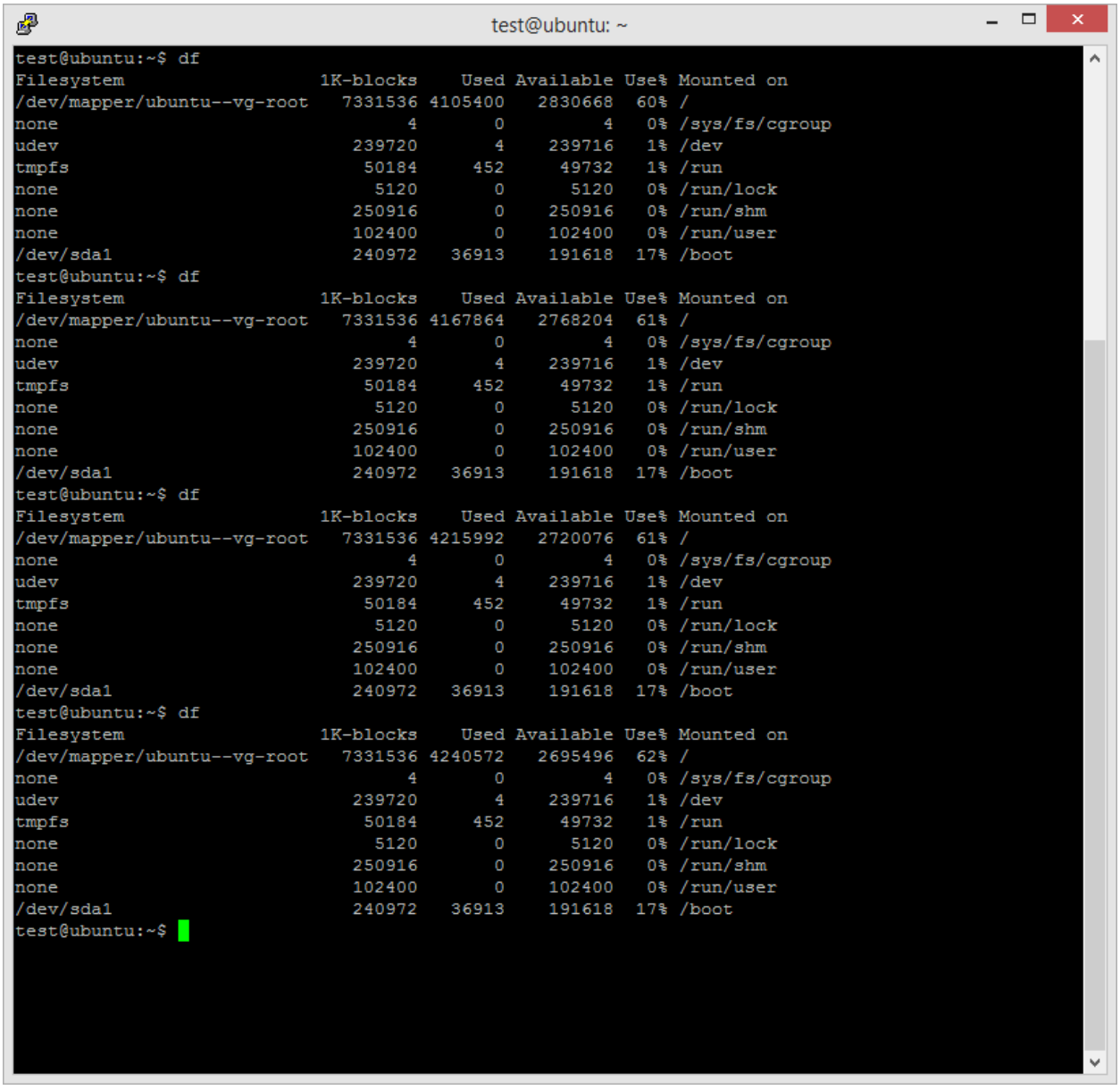
Even when the user monitors the running process using ‘top’ command, they may not find any suspicious activity, because the malware quickly spawns and dies.

10. Ubuntu server report again lags behind the actual output from ‘df’



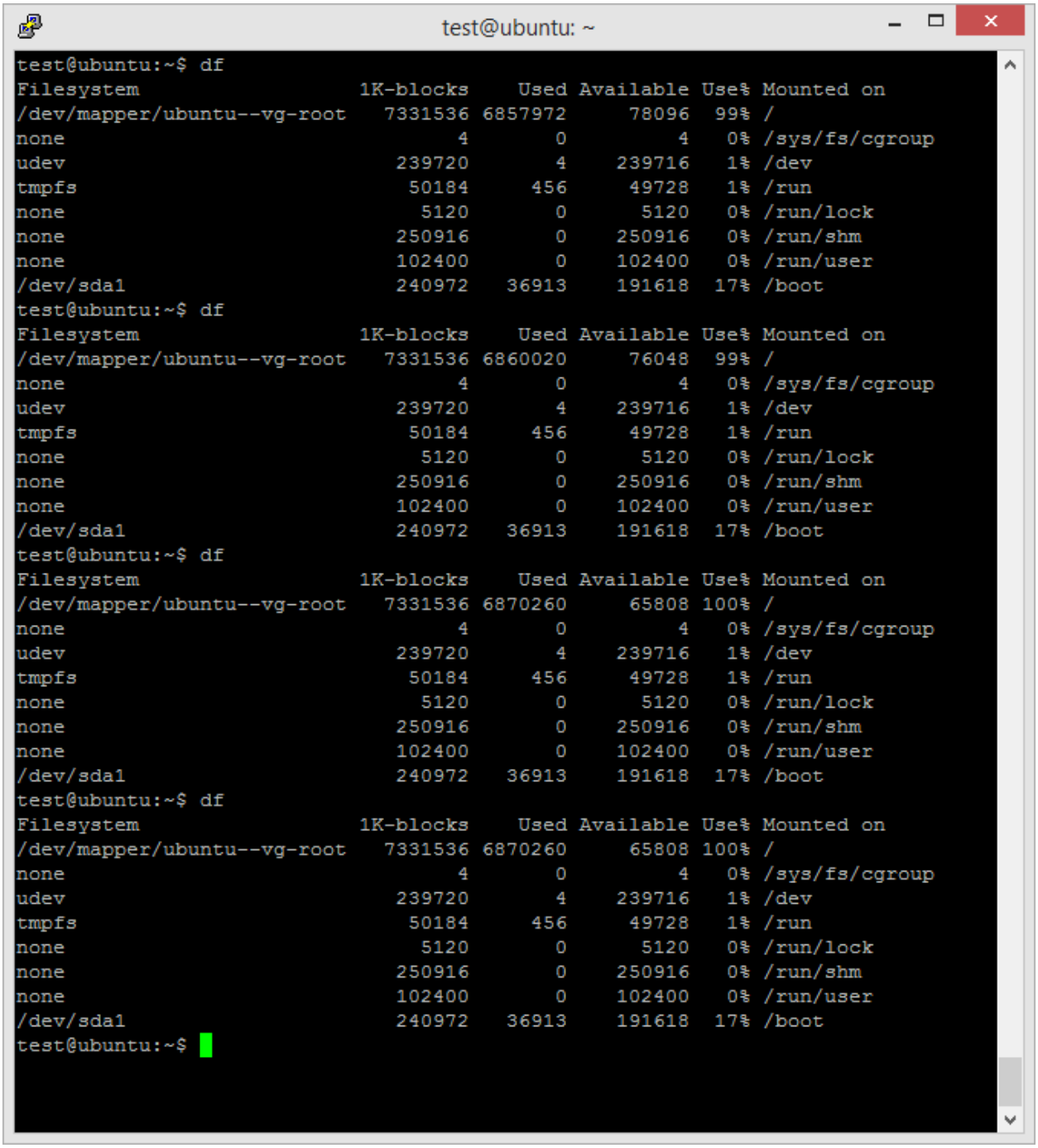
Even if Ubuntu reports the disk usage when a user log in to the server, there are great discrepancy between the actual disk sizes shown by ‘df’ command.

11. Storage fill rates gradually accelerates as the user repeat logout and login



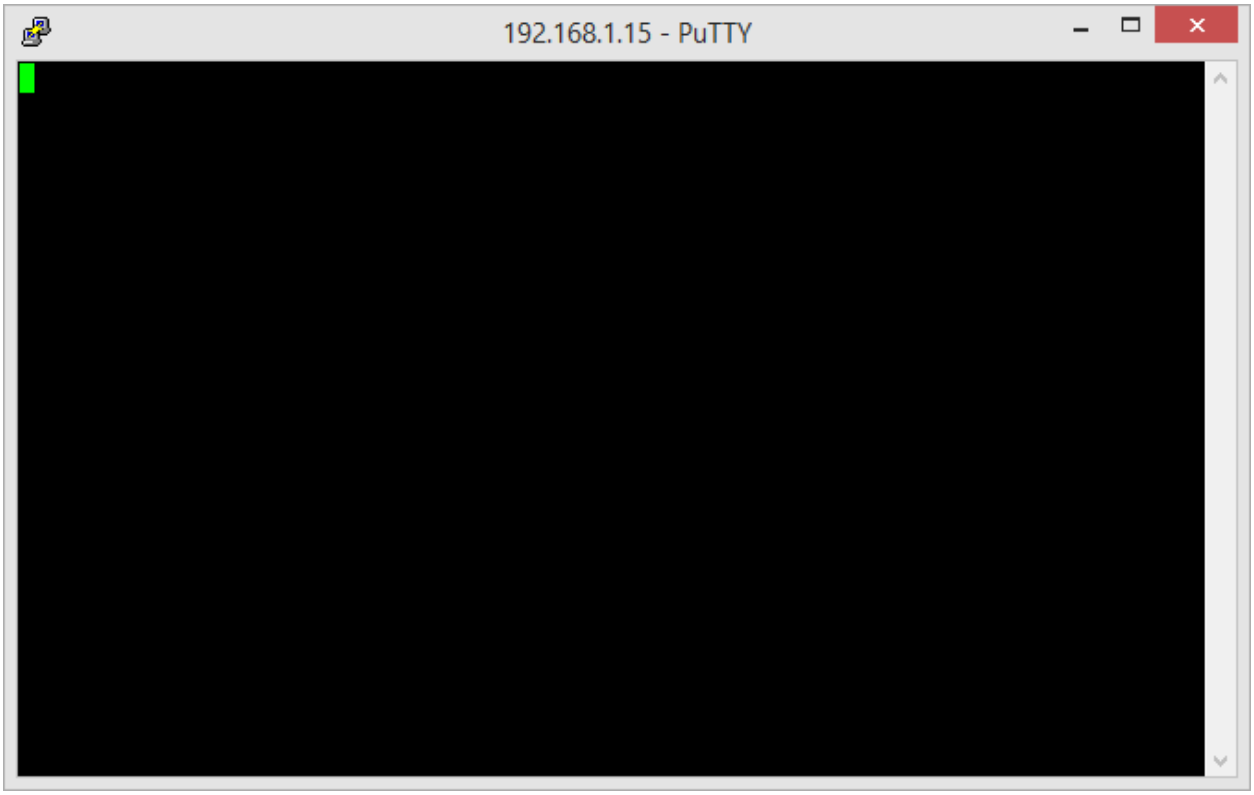
As a user continues to login and logout his/her session, the rate of disk filling accelerates. Hence as disk fills up, user may have less chance to notice the problem in his/her system.

12. When the disk usage hits 100%, the fork bomb attack begins



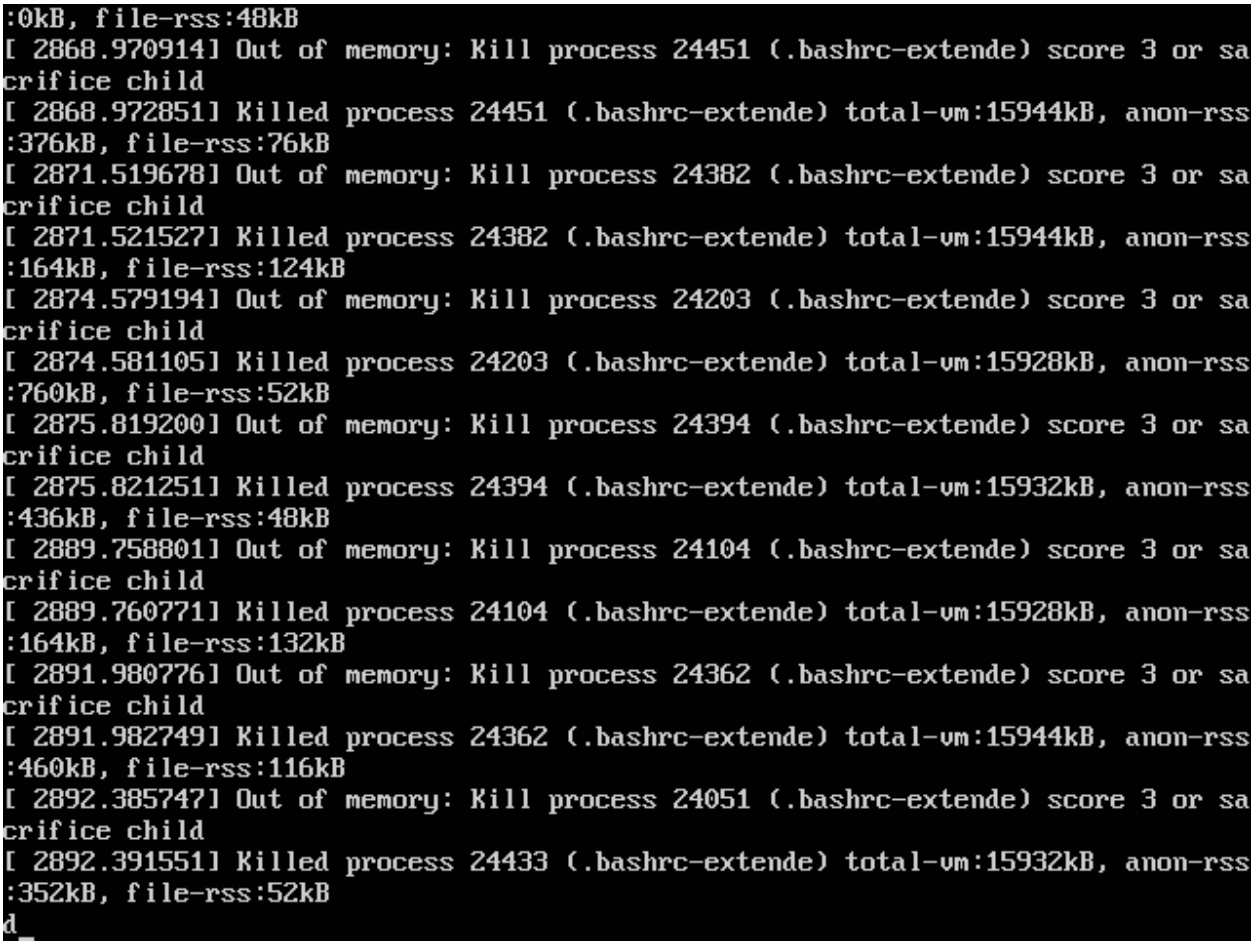
When the disk usage hits the maximum it can handle, the malware starts the fork bomb and make the system unusable. The system will halts right away, and many on-going server operation will begin to fail.

13. Console stops and cannot do anything. When try to reconnect to the server through SSH, not responding.



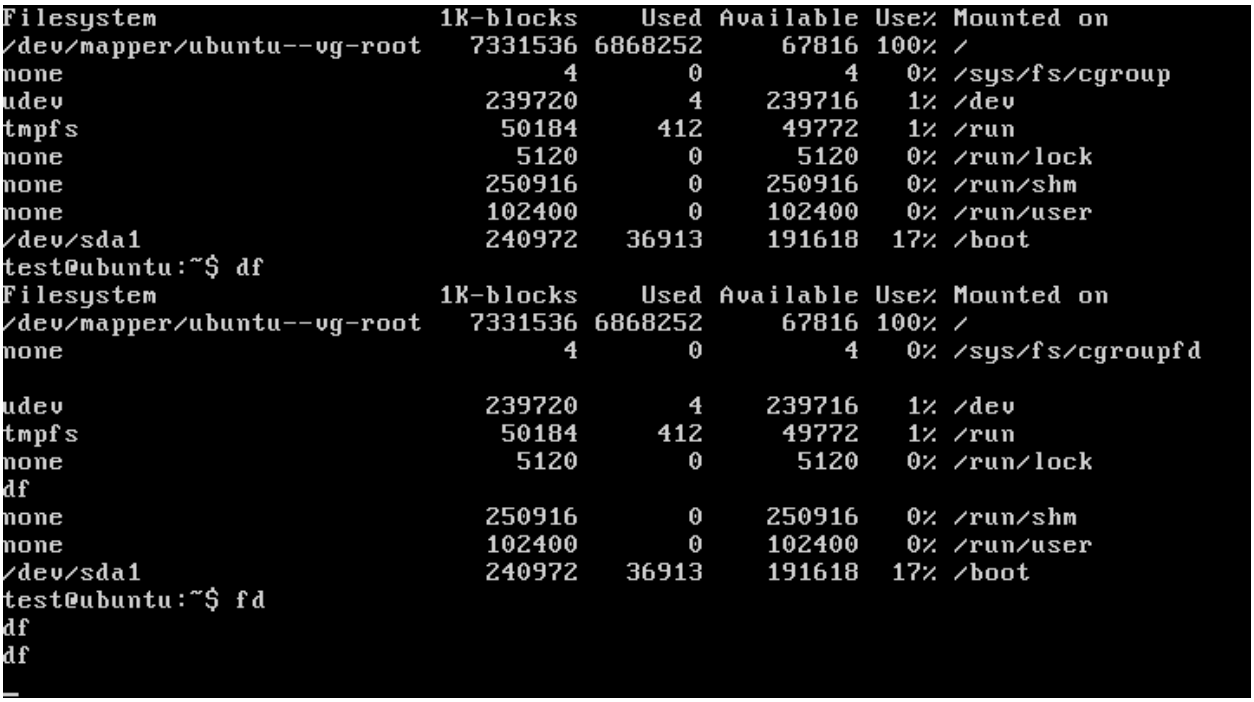
As the result of final fork bomb attack, the user will not be able to login to their server using SSH.

14. Server machines complaining about out of memory



15. Manually shut off and restart the server, but dies as soon as user try to log-in

16. Tried to boot from recover mode, and reboot, but fails soon after log in.



Even if the user tries to boot into the recovery mode, after he/she log-in the server will fail very soon because of the fork bomb. Even if the user somehow manages to log-in safely, it will be very hard for him/her to recover the system because our size-allocated empty files are spread all across the directories that user has the write permission. If there are some data or program stored by that user, without 'sudo' command, then the user may not be able to fix them in short time.

Conclusion

In creating this malware, we were inspired by the ILOVEYOU computer worm which have infected many computers and created massive damages several years ago. Looking at the fact that the ILOVEYOU worm used Visual Basic scripting language, we thought that malwares do not need to be complicated. Thus, we utilized on the belief that Linux system are generally secure under the property of superuser(sudo) and tendencies of people blindly running programs that they have downloaded from the internet.

For this reason, we have tried to create a malware that does not need "sudo" privileges for its attack and based on some default functionalities of Linux based systems. After long research and endeavor, we were able to create a malware that would disrupt the availability of the Linux based server without "sudo" privileges.

Throughout this project, we found it interesting to see that the availability of the system could be interrupted without "sudo" privileges as well as without extremely complicated codes. This made us realize how difficult it is to secure valuable systems and resources, and it requires knowledge on many different aspects of the system.