Proceedings of the 10th Colloquium for Information Systems Security Education
University of Maryland, University College
Adelphi, MD June 5-8, 2006

Automated Reverse Engineering Tool

Ramakrishnan Ravindran , Student, and Richard R. Brooks, Associate Professor

Abstract –: ***Network security plays an increasingly important role in technology. As the world gets more and more interconnected, the need for security increases. While there are several tools that offer a fair amount of security, it is still crucial that students are educated well on the design and operation of malware, and learn to develop countermeasures that prevent malicious activity. To assist in this, we developed a software package that studies the actions of known or suspected malware in a controlled environment, and provides information on the effects of malware on the system without actually compromising a system. By means of a virtual environment, this program collects data before and after the malware has infected the virtual machine. Comparison between the two aids in understanding the working of the malware and identifying system weaknesses. Several reverse engineering techniques have been implemented to collect vital information about the malware being simulated. This tool is invaluable to an educator seeking to show students the impact of a certain virus or the effects of a Denial of Service attack. It also acts as the starting step in the reverse engineering process, so that students realize the potential of malicious software and also learn the process of reverse engineering. This software is also a platform for further research in reverse engineering.***

Index terms – reverse engineering, educational tools, network security

I. Introduction

Computer viruses and related tools for disrupting security are known as malware. These programs deliberately harm computers, or steal information and spread to other computers through networks or storage media. There have been several thousands over the past decade and anti-virus companies are constantly updating their databases to reflect new threats. Some anti-virus software even have heuristics they use to detect suspicious activity and inform the user or take action themselves, even if the virus is not in its database.

Malware do not always operate in a predictable manner. This is why creating software to detect attacks is extremely difficult and may result in an excessive number of false positives. Software that analyzes virus behavior and operation can help find patterns in virus designs. The goal of this research is to design software that executes malware in a controlled environment and collects an operational profile. This software is by no means an anti-virus software to detect viruses and remove them, nor is it a software which can be used for extensive analysis and reverse engineering on viruses. It is however and educational tool designed to be used by students to study malware which have been popular and successful (from the perspective of the person who programmed the malware). This can also be a building block and hopefully, a foundation for something bigger. Using this model, software can be designed to be used for research purposes to find common traits in the operation of malware and find patterns in their working mechanisms.

There have been a few virus simulators in the past, for example the one discussed by Konstantin Rozinov [6], and the virus simulators available on websites like VxHeavens [26]. But these were all designed for simulation of one particular virus and they just simulate the execution of the virus and do not perform any reverse engineering. This software however not only simulates the execution of a given virus, but also performs some reverse engineering operations and presents it to the user for analysis.

Furthermore, it has can also simulate a Denial of Service attack. A denial of service (DOS) attack is a type of attack on a network, which is designed to bring down or slow down a network by flooding it with useless traffic. This is carried out by one computer targeting another computer or several computers (called Zombies) targeting a network. The DOS attack is simulated in a virtual environment and provides the user with packet information. Showing a Denial of Service attack in action will certainly be of great use to students.

Hence the simulator proposed in this paper provides a great starting point for developing a powerful research tool and also serves as an educational too which can be

Richard R. Brooks, Associate Professor, and Ramakrishnan Ravindran , Student
Holcombe Department of Electrical and Computer Engineering
Clemson University
P.O. Box 340915
Clemson, SC 29634-0915
USA
Tel. 864-656-0920
Fax 864-656-5910
Email: *rrb@acm.org*

used in institutions for students to actually see malware in action and get some experience with reverse engineering and study the working of Malware with a more practical and hands on approach rather than a theoretical one.

II. THE SIMULATOR

The simulator uses the following software

- VMware
- PERL
- Ethereal
- Shell

All processes are controlled by the PERL script and are child processes created during execution of the script. Ethereal is a network analyzing protocol used along with various command line options to capture packets during a Denial Of Service attack and display the packet information after the attack.

The reason for choosing VMware for the simulator is to provide a controlled environment for the simulation of malware. VMware is software that allows a completely independent operating system to exist within another operating system. It is like running a program like a game or any other application, only instead of an application, this software runs a whole operating system. There is no need to partition the hard disk or perform multiple installations of operating systems during boot up of the computer. Instead, once the host machine (the physical machine) is running, VMware is simply powered on, and the virtual machine can just be powered on while the host OS is still running. Within a screen, the display will be as though an independent physical machine is booting up, starting with the boot up screen. Installation of an operating system can be carried out just like on any other computer. The virtual machine tells the virtual operating system that it has physical devices like a real machine; however it just shares all the hardware devices of the host machine and fools the virtual machine into believing that it has its own devices. This then gives you the opportunity to have two operating systems on one single machine, maybe even more depending on whether the host systems processor and RAM can handle it. You can then connect the virtual machines to the host machine in several different ways.

- Bridged Networking
- Host Only Networking
- Network Address Translation
- Private network

Bridged networking just bridges the network connection of the host machine to the virtual machine and hence, the virtual machine can have access to the physical network via the host machine. Host only networking is where a separate network interface is created between the host machine and the virtual machine and this network is completely isolated from the network to which the physical network card of the host machine is actually connected to. Network Address Translation can be used when the network needs to be used and the virtual machine should have its own IP address. If multiple virtual machines exist, then all of them can be placed in a private network, each with its own IP address. In our case, we use a host only network titled vmnet1. It is necessary for the user to configure a new network interface with the title vmnet1 and give it the subnet 192.168.121.1 during installation. This is absolutely crucial for the simulator to run since the virtual machine provided with the installation disk is a preconfigured virtual machine which has the IP address of 192.168.121.128. Also Ethereal starts automatic packet capture during a Denial of Service attack, on the vmnet1 interface.

The biggest advantage of VMware is that the virtual machine is represented by a directory consisting of a few files with extensions like vmx. These are the vmware data files and all one needs to do to backup a virtual machine is to copy over these files to another location. If anything happens to the virtual machine while it is running, it can simply be powered off and its files can be overwritten with the backup files. This is a step that is crucial to the execution of the software since every time the simulator runs, the virtual machine is going to be infected with a virus. This needs to be overwritten before the next simulation or else the results will be false, or the virtual machine might fail to start.

III. DESIGN AND WORKING OF THE SIMULATOR

The simulator has two sections, the PERL script and the SHELL script. The PERL script controls all operations on the host machine and the SHELL script handles all operations on the Virtual Machine. A brief set of steps on how the simulator works for a virus is as follows:

1: The PERL script is run on the host machine
2: A list of options is displayed to the user; user chooses which Malware to simulate.
3: PERL copies over the required Virus into a directory, and the corresponding script file for the Virtual machine, resets the Virtual machine and then powers on the Virtual machine.
4: The Virtual machine reads the start up script and performs the instructions on startup. This involves copying over the virus from the host machine (using SCP) and running all the data collection instructions on the ELF file.
5: Once all the data is collected the information files are copied over to the host machine using scp. Both the host machine and the virtual machine are

preconfigured to run copy files using scp without prompting for a password.
6: The host machine times out the session and displays a list of results obtained.
7: If file system change option is selected, then a different script is copied over to the virtual machine after resetting it and it is powered on again.
8: The original version of the infected files are copied over to the host machine and host machine times this session out and extracts all original files to one directory and corresponding changed (infected) files to another and checks them for change of content.
9: Once all results have been displayed, user can return to main menu to simulate another virus.

The third step is one of the most crucial steps in the simulator. This is the step where the back up of the virtual machine overwrites the infected virtual machine to provide a clean installation of the virtual machine. Hence every time the simulator is run, the previous session's virtual machine is deleted and a fresh copy is created by copying over the files from the backup directory. This is a major advantage of using VMware as this is a matter of copying over a few files from one directory to another. There is no need for any form of user intervention and the whole process is automated. The virtual machine has been modified in such a way that it starts the files required by the DOS tool namely Masterserver (mserv) and Client(td) programs during start up and also runs another file called RUNME, which copies over the script file from the host machine. Once the script file is copied over, RUNME changes the privileges for the script file and executes it. From there on, the script file takes over and is also responsible for shutting down the virtual machine once it is done.

For simulation of a Denial of Service attack the steps followed are slightly different and as follows:

1: Once the user selects Denial of Service simulation, the PERL script resets the virtual machine and copies over a script and powers on the virtual machines.
2: The host machine waits for a finite duration and then begins an Ethereal packet capture session for a finite duration.
3: At the same time, the first virtual machine powers on, runs the script and begins a Denial of Service attack on the host machine.
4: The Ethereal session on the host machine times out and the captured packets are saved in to a capture file.
5: VMware is shut down and the user is given a menu with results obtained. Various capture filters are added to the capture file to display the different types of attacks that were simulated.
6: User is then given an option of returning to main menu to simulate another attack.

IV. PERL SCRIPT

The PERL script is the core of the simulator. This script controls and sequences all the events that the simulator takes. A visual representation of the structure of the simulator is in Fig. 1.

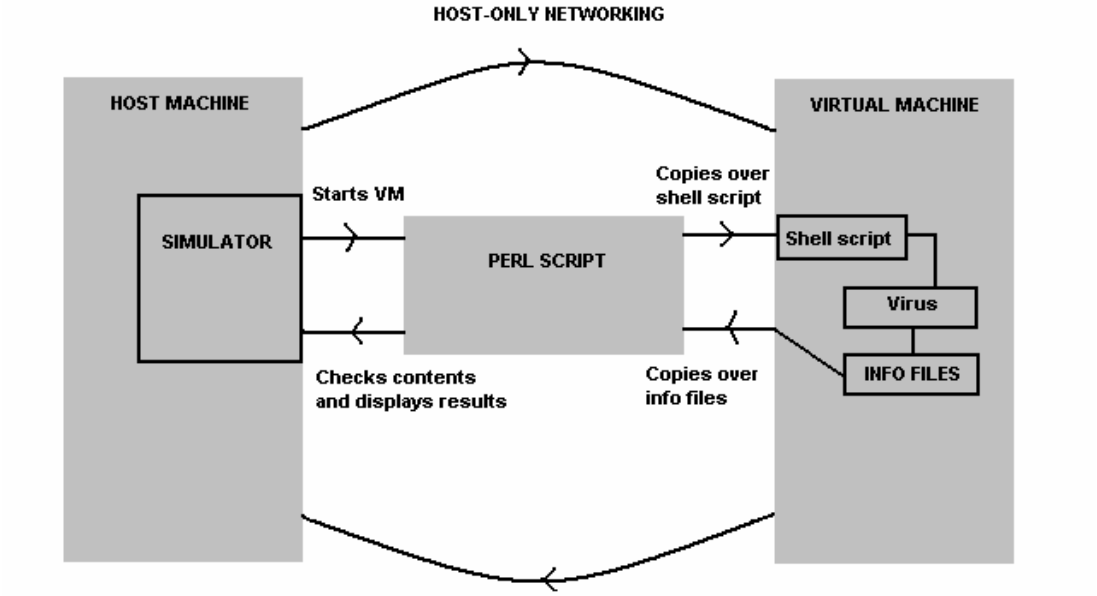

Figure1: A visual representation of the simulator

The script starts out by displaying a set of options to the user. These comprise of simulation of sample viruses, the option to simulate a Denial of Service attack and the option to simulate any given ELF file. The last option requires that the user place the ELF file in the path specified. Once the user makes the decision, the script does the following:

1: It begins by copying over the ELF file onto another ELF file called 'Virus'.
2: The shell script for the Virtual machine is copied over to a file called 'Script'.
3: The Virtual machine files are copied over from a backup directory so as to ensure that the previous session files are over written with fresh files.
4: The Virtual machine is powered on.
5: The Virtual machine has a start up script which copies over the 'Script' file from the host machine using the scp instruction and changes permission to run it, and runs it.

These are steps common to both Virus simulation and DOS attack simulation. Both the host machine and virtual machine are configured for passwordless login; hence the scp instruction will not prompt for a password when copying over the files. The steps followed after these five steps are the ones that vary. For a Virus or given ELF file simulation, the steps are as follows:

6: The shell script collects all necessary information and copies them over to a TEMP folder on the host machine.
7: User is presented with a list of results obtained and the selected result is displayed on screen by reading the respective .info file.
8: In the case of a file system change, the PERL script over writes the Virtual machine files with the back up and then boots the virtual machine after copying over a different script to the 'Script' file.
9: This script file copies over the necessary files to the TEMP directory.
10: The infected files and clean files are extracted to separate directories on the TEMP folder for further analysis.
11: Once the user has completed viewing all results, they may return to the main menu to run another simulation.
12: The user is also provided an option to retain the files containing the results. If the user decides to do so, the TEMP folder is not deleted on exiting the program.

For a Denial of Service attack, the following steps are executed:

6: The PERL script waits for a while after booting the virtual machine and begins a capture session using Ethereal on the host machine.
7: After the capture session has ended, the user is presented with the capture file which can be further analyzed using Ethereal.

V. SHELL SCRIPT

The SHELL script for simulation of Virus or given ELF file does the following:

1: It copies over the 'Virus' file to the virtual machine and changes permissions to run it.
2: Gets the list of running processes and saves it to file along with the disassembled code of the ELF file.
3: Runs the virus and saves the immediate STDOUT to a file.
4: Finds process ID of the Virus if it is running in the background and then finds the files in the /proc directory in relation to that PID and copies them over into a tar ball.
5: Finds if there have been any modifications to the bin directory and copies over all the modified files into a tar ball and saves the list in a .info file.
6: Copies over all the files from the /bin directory from the host computer so that further instructions are not hampered by the running of the virus.
7: Finds all files in the /etc, /usr, /root, /home directories that have been modified after running the virus and saves them into a tar ball and saves the list in a .info file
8: Performs Strace, Ltrace, Objdump, Hexdump, Netstat, and String search on the virus file and saves all results onto separate .info files.
9: Gets list of processes running and saves them into another .info file.
10: Copies over all the .info and tar files to the TEMP directory on the host machine.
11: Removes all the files in the virtual machine.
12: Shuts down the virtual machine.

In the case of a DOS attack, the shell script simply executes the client program to perform a DOS attack on the specified IP address. An ICMP attack and a UDP attack are performed once after the other with 5 seconds of each. Filters are used on the capture file to differentiate the two attacks.

To obtain the file system changes, there is another script that is copied over to the virtual machine. This script simply takes all the .info files created by the find command (i.e. Files modified in the /bin, /etc, /usr, /root, /home directories) and makes a tar ball of all the files listed in these info files, copies them over to the host

machine and then shuts down the virtual machine. Once this is done, the tar ball contents from the infected machine and the clean machine are extracted to folders in the TEMP directory.
The PERL script then presents the user with an option to run the diff command on the files in the corresponding clean and infected directories. Hence if the user selects to view the file changes in the /bin directory, then the output will be the comparisons between all the clean files and infected files.

During the simulation, there is a possibility that there aren't any changes made to any files in the folders which are checked for file modifications. If that is the case then, the list file will be empty and hence the tar ball for that particular folder will not be created.

Similarly, several viruses are non memory resident and will exit immediately after execution. In such a case, the /proc folder for the virus cannot be accessed. Hence only if the virus has a PID and is memory resident, the tar ball is created and copied over and extracted to a PROC folder in the TEMP directory.

VI. SUMMARY AND CONCLUSIONS

In this work, the primary objective was to develop software to safely simulate malware and present the user with information on the malware using reverse engineering techniques. The obtained data can be used for further analysis and research on the malware. A virtual machine installed using VMware is used for simulating the virus. A virtual machine is a fully independent machine which appears to have its own hardware devices although it actually just shares the devices of the host machine. This software is not guaranteed to work on all viruses, for example, the simulator will fail to provide results if the virus completely crashes the virtual machine.

The software uses PERL as the base for running and controlling all operations. It automates the process of backing up, restoring, powering on and shutting down the virtual machine. A shell script runs all the necessary commands and tools in the virtual machine and saves all collected information onto data files which are copied over to the host machine and presented to the user. Hence this is a completely controlled environment with no chance of the virus spreading on the host machine since it is not given executable privileges in the host machine and never triggered in the host machine. Thus the risk of harming the host machine is very low.

This tool was developed as an educational tool to provide a platform for learning reverse engineering and to observe common malware in action. It also serves as a starting step for the development of a more powerful research tool for reverse engineering. One of the possible extensions to this software is to add the ability to find common patterns and trends in malware. Similarly, adding a graphics interface and additional reverse engineering tools are other possible extensions.

1. REFERENCES

1. Computer Viruses as Artificial Life, Eugene H. Spafford, Department of Computer Sciences, Purdue University
2. White Paper: Distributed Denial of Service Attacks: Analysis and Partial Solutions March 2000
3. Defending against a Denial-of-Service Attack on TCP, Pars Mutaf, Department of Computer Engineering, Izmir Institute of Technology
4. White Paper on Reverse Engineering, *Spencer Rugaber, C*ollege of Computing and Software Engineering Research Center, Georgia Institute of Technology, March 9, 1994
5. Reverse Engineering: A Roadmap, Hausi A. M¨ uller, Jens H. Jahnke, Dennis B. Smith, Margaret-Anne Storey, Scott R. Tilley, Kenny Wong
6. Reverse Code Engineering: An in-depth Analysis Of The Bagle Virus, Konstantin Rozinov, August 12, 2004
7. Trends in Denial of Service, Attack Technology, George M. Weaver, Neil Long, Rob Thomas
8. Norman Book on Computer Viruses
9. Creating a Secure Computer Virus Laboratory, *John Aycock and Ken Barker*
10. Reverse Engineering Malware, *Lenny Zeltser*
11. Distributed Denial of Service, Trin00, Tribe Flood Network, Tribe Flood Network 2000, And Stacheldraht, CIAC-2319, Paul J. Criscuolo, February 14, 2000
12. Analysis of a Denial of Service Attack on TCP, Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, Diego Zamboni, COAST Laboratory, Department of Computer Sciences, Purdue University, 1398 Department of Computer Sciences, West Lafayette, IN 47907–1398
13. Introduction to Reverse Engineering Software, Mike Perry, Oskov, http://www.acm.uiuc.edu/sigmil/RevEng/
14. Reverse Engineering, Wikipedia, http://en.wikipedia.org/wiki/Reverse_engineering
15. PERL home page, http://www.perl.org/
16. Unix Shell Scripts, Norman Matloff, http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Unix/CShellII.html
17. UNIX Bourne Shell Scripting, Ken Steube, http://research.imb.uq.edu.au/~ksteube/Bshell/
18. The Complete Reference UNIX, Robert Petersen
19. Perl 5 Developer's Guide (Paperback), by Ed Peschko, Michelle Dewolfe

20. The "stacheldraht" distributed denial of service attack tool, David Dittrich, Dec 31, 1999
21. SANS Malware FAQ: Analysis on DDOS tool Stacheldraht v1.666, Geoffrey Cheng, http://www.sans.org/resources/malwarefaq/stacheldraht.php
22. Linux Advanced Routing & Traffic Control HOWTO, Bert Hubert http://lartc.org/lartc.html
23. The Protocols (TCP/IP Illustrated, Volume 1) (Hardcover), by W. Richard Stevens
24. SANS FAQ for Stacheldraht, http://www.sans.org/resources/idfaq/stacheldraht.php
25. VMware Manual and Documentation
26. VxHeavens (http://vx.netlux.org/)

.