How the virus “Remote Shell Trojan” (RST) works

Marco Balduzzi <balduzzi@idi.ntnu.no>

University of Trondheim
Norwegian University of Science and Technology, Department of Telematics

Abstract

This paper introduces the concept of malicious software, spending more words on virus threats. Later it focuses on the “Remote Shell Trojan” virus, a well-known GNU/Linux code which spread across on all Internet in September 2001.

1. Introduction

Malicious programs are probably the most dangerous threats for computer systems; the literatures divides them into the following classes: viruses, worms and trojan horses.

The jargon file [1] defines a virus as a cracker program that searches out other executable files and infect them by embedding a copy of itself. When these programs are executed, the embedded virus is executed too, doing bad mistakes and propagating the infection. Finally the original program is called and run in normal way: thus the virus activity is invisible to the user.
Some viruses infect memory and boot records instead of simple files, becoming more aggressive and dangerous.
A worm is a program which doesn't require file transmission like viruses. It takes advantage of systems vulnerability to diffuse across the network and infect running processes. A very famous worm was coded by R. H. Morris. The story of “The Internet Worm of 1988” is at [2].
A trojan horse is a code which pretend to be a useful program but when executed it performs some unwanted or harmful function. Differently from virus and worms, a trojan horse doesn't auto-propagate but require user installation. When executed, a trojan horse grants to the cracker the access to the victim's computer.

Other minor classes of malicious programs are the backdoors/trapdoors (secret entry point that allows someone aware to obtain access to the system without going through the usual procedure), the logic bombs (code embedded in some legitimate program that's set to “explode” when certain conditions are met) and zombies (programs used in DoS attacks).

2. Overview of “Remote Shell Trojan” virus

The “Remote Shell Trojan” virus (or RST) has been discovered by Qualys at the 5th of September 2001 and named due to its backdoor functionality. It's also known with the name RST.a to distinguishes from its more recent variant (RST.b) appeared four months later. An anonymous has written on buqtraq mailing-list about its origin [3]: *“RST was developed by us as a research project and intended only for internal use on our systems. [...] An infected binary accidentally leaked out our research lab and came into the hands of so called script kiddies”.*

The source code of the virus has still not been released, thus the analysis have been made with the “black-box” techniques or disassembling the virus executable.

RST doesn't take advantage of the vulnerability of computer systems; differently from a worm, the infection is diffused with the exchange of file (email attachments and download). RST attacks GNU/Linux

ELF binaries and when run it infects all binaries in the current and in /bin directory (like cp, mv, ps commands). It also provides a backdoor functionality which allows the attacker to launch arbitrary commands.

3. RST analysis

The virus begins to spread by attaching itself to a “healthy” ELF binary using a variation of Silvio Cesare's technique described here [4]:

- the viral code inserts itself between code segment and data segment;
- the viral code is modified to jump to original entry point afterwards;
- the entry point of the executable is changed to run the viral code;
- some field of ELF header are adjusted (code segment size for example) and other data are moved to the end of the virus.

Afterwards the virus forks and the parent runs the original code while the child acts in evil way. It spawns a backdoor listening on UDP port 5503 (the RST.b variant use an EGP raw socket realizing a more hidden communication channel). Special packets enable the backdoor to execute remote commands with the privileges of the process.

It's interesting to note that the virus adopts a anti-debugging technique which consists in calling the ptrace(PTRACE_TRACEME) function to check if someone is debugging us [5]:

```
int main()
{
 if(ptrace(PTRACE_TRACEME, 0, 1,0)<0)
 {
   printf("Don't debug me!\n");
   return 1;
 }
 printf("Normal execution...\n");
 return 0;
}
```

A workaround is to open up the file in a hex editor and change the int 80 to NOP NOP which prevents ptrace from being called. Another possibility is to block the ptrace() syscall with a LKM (Loadable Kernel Module).

4. Detection and immunization

Many tools has been provided by antivirus software-houses and underground community, but nobody of these work perfectly because of the many variants of the virus and the difficulty of debugging.

For detection a reliable procedure is checking MD5, timestamps and size of /bin files (host-based IDS like Tripwire becomes very useful). In fact the viral code infection alters binaries.
Another technique it to verify whether the entry point is exactly 4096 from the end of the code segment. It'd mean that the executable is infected.
Some scanners parse `readelf -l` output for LOAD segments and calculate the distance between the start of a new LOAD segment and the end of the previous one. If the distance is less than 0x1000 bytes a warning is printed.

Immunization works by increasing the size of the text segment by 4096 bytes so that the hole between the text and data segments is gone. Thus there is no space for the RST to add itself to the binary anymore.
If it were known the original entry point, a simpler solution would be to restore the entry point. This technique requires a deep reverse engineering procedure since many viral codes store the original entry point at a non-intuitive address, in the middle of infective code.

References

[1] The Jargon File, http://www.catb.org/jargon/
[2] The Internet Worm of 1988, http://world.std.com/~franl/worm.html
[3] Remote Shell Trojan: Threat, Origin and the Solution
[4] Unix ELF Parasites And Virus, http://www.packetstormsecurity.com/9901-exploits/elf-pv.txt
[5]Linux anti-debugging techniques (fooling the debugger), http://vx.netlux.org/lib/vsc04.html