

JULY 2006

Sage™

Security Vision from McAfee® Avert® Labs

Vol. 1 • Issue 1



Paying a price for the open-source advantage

2 Security Trends and Events of the Last Six Months
6 Good Intentions Gone Awry
13 Money Changes Everything
21 Open-Source Software in Windows Rootkits

26 Building Better Bots
36 Is Open Source Really So Open?
38 Vulnerability Bounties
40 Will the Worm Eat the Apple?

CONTENTS

FEATURES

pg.6	<h3>Good Intentions Gone Awry</h3> <p>Open source was supposed to hinder malware. So what happened?</p>
pg.13	<h3>Money Changes Everything</h3> <p>Malware authors leverage open-source model for profit.</p>
pg.26	<h3>Building Better Bots</h3> <p>Open-source processes enable production-grade malware.</p>
TECHNICAL	
pg.2	<h3>Security Trends and Events of the Last Six Months</h3>
pg.21	<h3>Open-Source Software in Windows Rootkits</h3>
pg.40	<h3>Will the Worm Eat the Apple?</h3>
OPINION/EDITORIAL	
pg.36	<h3>Is Open Source Really So Open?</h3>
pg.38	<h3>Vulnerability Bounties</h3>



Editor's note

Welcome to the premier issue of *Sage*—a semi-annual objective forum of leading-edge security research, analysis, trends, and opinion. In this issue, we examine the darker side of open source. By open source, we refer to the free and unconditional sharing of source code and ideas. We look at how the social norms and tools of the open-source movement have been usurped by the malware-writing community and applied to the development of ever-more dangerous and virulent creations.

If one trend in security research has emerged in recent years, it is the rise of data. Data are everywhere, from ubiquitous security surveys to monthly or semi-annual vendor security-threat reports. This shift from fear, uncertainty, and doubt (FUD) to quantitative justification is a positive step, but reporting data is not the same as presenting information. In fact, the abundance of data only underscores the lack of crisp analysis that takes the data and turns them into truly useful information that can support sound decisions.

The goal of *Sage* is to help rectify this drought. We will publish predictive and incisive security research that helps you understand the current and evolving threat environment and, ultimately, empowers your security decisions. Whether you are a security decision-maker or a researcher, we hope you find *Sage* insightful and compelling.

Kevin J. Soo Hoo
Editor

For comments and inquiries, please contact the editorial staff at Sage-feedback@McAfee.com.

The views and opinions expressed in *Sage* are strictly those of the individual authors and in no way represent the views and opinions of McAfee, Inc.

McAfee and/or additional marks herein are registered trademarks or trademarks of McAfee, Inc. and/or its affiliates in the US and/or other countries. McAfee Red in connection with security is distinctive of McAfee brand products. All other registered and unregistered trademarks herein are the sole property of their respective owners. © 2006 McAfee, Inc. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or through any information storage and retrieval system, without written permission from McAfee, Inc.

Cover illustration by Mack Jolley.



In this Issue:

The Open Sourcing of Threats

Open source is an important and powerful force in today's networked world. From basic tools and utilities to applications and operating systems to the foundation of the Internet itself, open-source products have created tremendous value.

The fundamental tenets of the movement are quite simple:

"When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves. This rapid evolutionary process produces better software than the traditional closed model at a speed that, if one is used to the slow pace of conventional software development, seems astonishing."¹

Belief in the open source philosophy approaches an almost religious zeal in its most ardent proponents. However, like any powerful tool, open source can also be used for malicious purposes, particularly in security. Whether posting a terrorist training manual or a how-to guide for attacking infrastructure, there are consequences to the free and open sharing of information—especially in the realm of computer and network security, where the desirable degree of openness in the sharing of vulnerability and threat information and the role of open source in the production of malware are significant points of contention.

As Dmitry Gryaznov explains in "Good Intentions Gone Awry," malware authors have been collaborating and sharing source code, using books and bulletin board systems and, eventually, ftp sites and the Web, since soon after the first computer viruses appeared in the late 1980s. Gryaznov also quantifies the significant impact that such sharing has had on the production and proliferation of malware.

Igor Muttik continues the narrative in "Money Changes Everything," in which he presents ample evidence of a vibrant and sophisticated open-source community actively engaged in the development and dissemination of both new and repackaged malware. The bundling of threats and the use of obfuscating tools (to thwart security scanners) offer clear evidence that modern malware is the product of collaborative efforts.


The advent of bot herders and their botnets, however, signals a change in the character of and intent of malware. Though malware authors started sharing and collaborating 20 years ago, the degree of process maturity and quality of code in those early threats was never comparable to that of commercial software products. As a result, most malware was, by comparison, poorly written, prone to failure, and ultimately ineffective. Michael Davis' "Building Better Bots" confirms that this situation has changed. Bot malware is now developed with the same methodologies and tools used to produce marquee open-source products such as Firefox,

Apache, and MySQL. Driving this charge toward professional-quality code are the financial rewards that a large botnet can earn for its master, whether from sending spam, injecting adware, participating in a Distributed Denial of Service (DDoS) attack, or performing some other contracted nefarious activity.

Today's threat environment has materially changed from years past. The professionalization of malware coupled with the powerful open-source model is creating a formidable, profitable, and criminal adversary for security professionals. The fundamentals are in place for this new industry to thrive, virtually guaranteeing that malware will continue to become more robust, more sophisticated, more plentiful, harder to combat, and more dangerous.

Reactive security measures are unlikely to keep pace as these trends unfold. Though patching processes are improving and the window of malware opportunity between a patch's release and its widespread adoption is shrinking, targeted exploits that capitalize on previously undisclosed vulnerabilities are already uncomfortably common. Remediation—which by definition occurs after the fact—is also becoming more difficult. Mike Danseglio, a security program manager at Microsoft® shocked many at the InfoSec World Conference in April 2006 by stating flatly, "When you are dealing with rootkits and some advanced spyware programs, the only solution is to rebuild from scratch. In some cases, there really is no way to recover without nuking the systems from orbit."²

There are no easy solutions. Thankfully, current security measures have thus far managed to contain many of these threats. However, as malware continues to evolve and proliferate, prevention and proactive security may become the only ways to stop infections before they cause irreparable damage to systems and businesses.

Open source is not to blame for the current security trends, though it is a critical enabler for malware. In light of the ways malefactors use open source, perhaps the time has come to re-evaluate long-standing beliefs about full disclosure and absolute adherence to the open-source creed. Similarly, the security community may need to revise its traditional strategy of containing threats by controlling and restricting information, as it tries to compete with an open-source malware community that is becoming better organized, better funded, and more effective than ever. 

¹ <http://www.opensource.org/>

² Ryan Naraine, "Microsoft Says Recovery from Malware Becoming Impossible," *eWeek*, April 4, 2006. <http://www.eWeek.com/article2/0,1895,1945808,00.asp>



Security Trends and Events

of the Last Six Months | By Monty Ijzerman

TRENDS

The following descriptions cover prominent vulnerability and malware trends during the last six months (December 2005 through May 2006).

Monetization of Malware

For-profit malware emerged in 2003 and is now the overarching threat trend. Viruses, Trojans, bots, rootkits, worms, phishing attacks, spyware, spam, and other exploits are all used in criminal activity.

Exploits

A New Marketplace

Vulnerability bounty programs and the growing interest of criminals in the online world have created an army of for-profit vulnerability researchers and an increase in targeted zero-day attacks. Non-public vulnerabilities and exploit toolkits are for sale, enabling anyone to build these directed attacks, including companies that want to test their compliance levels and defenses.

Vulnerabilities

Increasing Numbers and Improving Patch Management

Discovered vulnerabilities are increasing about 30 percent annually. Vulnerabilities in Web applications comprised over two-thirds of total vulnerabilities disclosed in the second half of 2005.

Vulnerabilities are also growing in new areas. In the first five months of 2006, more than 80 vulnerabilities in Apple products were disclosed, compared to about 120 for all of 2005 and about 60 for all of 2004. The numbers of Firefox and Mozilla vulnerabilities are also increasing.

Fortunately, as vulnerabilities have increased, vendors' patch release cycles have shortened, reducing the vulnerability window. In the first half of 2005, the time between a vulnerability's disclosure and patch availability was 64 days; in the second half of 2005, the window shrank to 49 days. For Windows® vulnerabilities, the window in 2005 was 46 days, slightly shorter than the aforementioned industry average. The time between vulnerability disclosure and the availability of an exploit was almost constant in 2005 at around seven days.

Vulnerability bounty programs and the growing interest of criminals in the online world have created an army of for-profit vulnerability researchers and an increase in targeted zero-day attacks.

Several events in the last six months demonstrate the growing market for vulnerabilities and exploits:

- ▶ Two for-profit zero-day attacks involved non-public vulnerabilities. One exploited the Internet Explorer WMF vulnerability; the other exploited the Microsoft Word Code Execution vulnerability. The underground knew of both vulnerabilities, and the WMF vulnerability was put up for sale before it was used in an attack
- ▶ In December 2005, an attempt was made to auction an undisclosed Excel vulnerability on eBay. eBay pulled the listing before the auction ended
- ▶ The Zero Day Initiative vulnerability bounty program was launched at the end of 2005, joining iDefense in their efforts to purchase vulnerabilities from independent researchers and work with the affected vendors to fix and disclose them
- ▶ Argeniss Information Security recently launched an exploit toolbox with canned exploits, joining a handful of vendors in this market. Several exploit toolbox vendors offer a two-tiered service that, for a premium, provides zero-day exploits

The time that companies need to patch systems is falling as well. One study found that the time to patch half of a sample of externally facing systems was about 19 days in the second half of 2005. In 2003, the length was 30 days. This roughly corresponds with a November 2005 study that found 19 percent of survey respondents took one week or more to patch their systems after the release of a patch.

Finally, Oracle joined Microsoft® and began a monthly patch-release cycle in 2005. Apple, however, has not yet established a regular patch-release cycle.

Rootkits

Growth of Stealth Techniques in Malware

Due to increases in quantity and quality of stealth technology (rootkits), malware has a higher chance of remaining unnoticed.

The number of malware samples that use stealth techniques increased four-fold from the first quarter of 2005 to the first quarter of 2006. Possible explanations for the increase include the general availability of rootkit code and ready-to-use rootkit executables. McAfee found that, in a random sample of 24 generally available rootkits, at least 12 were found in malware samples collected from the wild.

In 2005, rootkits began migrating from Trojans (non-replicating malware) to viruses, bots, and potentially unwanted programs (or PUPs, which include adware and spyware). In the first quarter of 2006, McAfee® Avert® Labs found that one-quarter of submitted malware samples incorporating stealth techniques were viruses, bots, and PUPs.

Bots

Leveraging Open-Source Techniques Yields Higher Infection Rates

Botnets can be easily rented for Denial of Service (DoS) attacks, spam distribution, and pay-for-click scams. Without protective measures, your systems could become part of an active botnet, or your company could be targeted by a DoS attack launched from a botnet.

Mytob, the most recent bot family to emerge, launched in February 2005, is estimated to have increased the number of bot-infected machines by 150 percent. Four older bot families—Sdbot, Agobot (Gaobot), and Spybot—saw fewer new variants in 2005; however, they still accounted for more than 7,000 new variants in the second half of 2005.

Bot authors are increasingly using open-source development techniques, such as multiple contributors, releases driven by bug fixes, paid feature modifications, and module reuse. This form of collaboration is expected to make botnets more robust, creating a more reliable ROI for botnet customers.

Phishing

Still Growing, More Sophisticated

In May 2006, the Anti-Phishing Working Group reported a 90-percent increase in new unique phishing sites since the second half of 2005. The group says it has received an all-time high of more than 17,000 phishing reports per month in 2006, most of which were related to scams involving first- and second-tier financial institutions. The number of phishing Web sites that host keystroke loggers grew by 130 percent from January 2006 to April 2006.

Approximately 40 percent of phishing attacks are in languages other than English. Even small geographic regions, such as Catalan, have been targeted. The practice of incorporating knowledge about phishing recipients to target attacks is known as “spearphishing.”

REFERENCES

Monetization of Malware

“CSII/FBI Computer Crime and Security Survey,” 10th edition, Computer Security Institute, 2005, <http://www.usdoj.gov/criminal/cybercrime/FBI2005.pdf>

“McAfee Virtual Criminology Report,” McAfee, Inc., 2005, http://www.mcafee.com/us/local_content/misc/mcafee_na_virtual_criminology_report.pdf

Exploits: A New Marketplace

“eBay Pulls Bidding for MS Excel Vulnerability,” eWeek, 12/09/2005, <http://www.eweek.com/article2/0,1759,1899697,00.asp?k=EWRS03129TX1K0000614>

Zero Day Initiative, <http://www.zerodayinitiative.com>

“Argeniss Ultimate Oday Exploits Pack,” Argeniss Information Security,

<http://www.argeniss.com/products.html>

Vulnerabilities: Increasing Numbers and Improved Patch Management

“Symantec Internet Security Threat Report,” July 2005–December 2005, Symantec, <http://www.symantec.com/enterprise/threatreport/index.jsp>

Brian Krebs, “A Time to Patch,” *The Washington Post*

Phishing attacks are also becoming more sophisticated, moving beyond the traditional spoofed e-mail and simple Web link to submit confidential information. One recent phishing email enticed victims to call a telephone number affixed to a Voice over IP (VOIP) system that was set up by attackers. Last year, Netcraft reported 450 phishing attempts using HTTPS sites and security certificates. In one highly publicized phishing attempt this year, attackers managed to obtain a valid certificate from a certificate signing authority.

Unfortunately, a recent Harvard study found that 90 percent of the participants did not identify well-constructed, real-world phishes. Nearly one-quarter of the participants ignored security indicators, and more than half dismissed pop-up warnings about fraudulent certificates.

EVENTS

EXPLOITS, VULNERABILITIES, AND MALWARE

The events below were selected based on the significant damage they caused, or could have caused, or because they were indicative of a new trend.

Zero-Day Exploits for Undisclosed Vulnerabilities

Internet Explorer WMF Exploit

On December 27, 2005, an exploit that used a previously undisclosed vulnerability in the handling of WMF files in Internet Explorer was published. That same day, malware exploiting the WMF vulnerability appeared on Web sites. In the following weeks, it became clear that the vulnerability had been for sale since mid-December 2005. After its acquisition by cyber-criminals, the vulnerability was exploited in the wild. On December 31, 2005, a third-party patch became available. Due to public pressure and the ongoing exploitation of the vulnerability, Microsoft released an out-of-cycle patch on January 5, 2006.

Technology Blog, 01/11/2006, http://blog.washingtonpost.com/securityfix/2006/01/a_time_to_patch.html

“The Laws of Vulnerabilities: Six Axioms for Understanding Risk,” *Qualys*, 2/13/2006, <http://www.qualys.com/docs/Laws-Report.pdf>

“The Window of Vulnerability,” McAfee, Inc., November 2005 (not available online)

“SANS Top 20 Internet Security

Vulnerabilities—2006 Spring Update,” SANS Institute, http://www.sans.org/top20/2005/spring_2006_update.php

Francois Paget, “Open Source—Will the Worm Eat the Apple?,” *Sage*, Vol. 1, Issue 1, McAfee, Inc., July 2006, http://www.mcafee.com/us/threat_center/white_paper.html

Internet Explorer “createTextRange” Exploit

On March 22, 2006, an exploit targeting the Internet Explorer “createTextRange” vulnerability was released. This original exploit caused Internet Explorer to crash, but within a day an exploit that resulted in code execution was released. The vulnerability was disclosed to Microsoft on February 13, 2006, but no patch had been issued by the time the exploit surfaced. Two third-party patches were released prior to the release of the official Microsoft patch on April 11, 2006.

Mac OS X and Safari 0-Day Exploits

On April 19, 2006, nine previously undisclosed vulnerabilities and corresponding exploit code were published for Mac OS X and the Safari browser. Apple remedied most of these vulnerabilities in its May 2006 security patch.

Firefox Deleted Object Reference Exploit

On April 24, 2006, a proof-of-concept showing JavaScript parsing problems in Firefox was published. The original code caused Firefox to crash; however, experts did not rule out the possibility of arbitrary code execution. The vulnerability was patched on May 2, 2006.

Microsoft Word Code Execution Exploit

On May 16, 2006, employees of a large, unnamed company received emails with Word attachments that, when opened, installed a Trojan. The exploit used a previously unknown vulnerability in Microsoft Word. The vulnerability and corresponding exploit were not publicly known until Microsoft released a patch on June 13, 2006.

Zero-Day Vulnerabilities with No Corresponding Zero-Day Exploits

Oracle PL/SQL Gateway Unauthorized Database Access

On January 25, 2006, a researcher posted details of a vulnerability in the Oracle PL/SQL Gateway. This

vulnerability allows unauthorized administrative access to an Oracle database. A canned exploit has not yet surfaced, but details in the researcher’s post make exploitation possible. Oracle released its patch on April 19, 2006.

Permissive Windows Services DACLS

On January 31, 2006, an academic paper detailing the improper configuration of several Windows services was published. This vulnerability allows local attackers to elevate their privileges. The corresponding exploit was not difficult to construct and appeared on February 3, 2006. Microsoft released a patch on March 14, 2006.

Malware Events

Nixum/MyWife/Blackworm

This virus began spreading in mid-January 2006. A Web site counter updated by the virus recorded more than 300,000 infections by the end of that month. The virus was programmed to erase files on the third day of every month. On February 3, 2006, it did just that. However, the predicted global data meltdown failed to materialize.

OSX/Leap, OSX/Inqtana

In February 2006, two viruses for OS X emerged. OSX/Leap propagated initially through image archive downloads and subsequently through the AIM/iChat messaging system. OSX/Inqtana propagated via an old vulnerability in the file exchange service that uses Bluetooth. [Sage](#)

About the author

Monty Ijzerman is a Senior Intelligence Communications Manager at McAfee Avert Labs. He has been involved in the security field since 2000.

Rootkits: Growth of Stealth Techniques in Malware

Aditya Kapoor, “Rootkits, Part 1 of 3: The Growing Threat,” McAfee, Inc., April 2006, http://www.mcafee.com/us/local_content/white_papers/threat_center/lwp_akapoor_rootkits1.pdf

Aditya Kapoor, “Open-Source Software in Windows Rootkits,” Sage, Vol. 1, Issue 1, McAfee, Inc., July 2006, http://www.mcafee.com/us/threat_center/white_paper.html

Bots: Leveraging Open-Source Techniques Yields Higher Infection Rates

Igor Muttik, “Money Changes Everything,” Sage, Vol. 1, Issue 1 McAfee, Inc., July 2006, http://www.mcafee.com/us/threat_center/white_paper.html

Michael Davis, “The Bot Development Life Cycle: Production Grade Malware,” Sage, Vol. 1, Issue 1, McAfee, Inc., July 2006, http://www.mcafee.com/us/threat_center/white_paper.html

[mcafee.com/us/threat_center/white_paper.html](http://www.mcafee.com/us/threat_center/white_paper.html)

Phishing: Still Growing, More Sophisticated

“Phishing Activity Trends Report—April 2006,” Anti-Phishing Working Group, http://www.antiphishing.org/reports/apwg_report_apr_06.pdf

“More than 450 Phishing Attacks Used in SSL in 2005,” Netcraft, 12/28/2005, http://news.netcraft.com/archives/2005/12/28/more_than_450_phishing_attacks_used_ssl_in_2005.html

[archives/2005/12/28/more_than_450_phishing_attacks_used_ssl_in_2005.html](http://news.netcraft.com/archives/2005/12/28/more_than_450_phishing_attacks_used_ssl_in_2005.html)

Brian Krebs, “The New Face of Phishing,” The Washington Post Technology Blog, 2/13/2006, http://blog.washingtonpost.com/securityfix/2006/02/the_new_face_of_phishing_1.html

John Leyden, “Phishing Goes International,” The Register, 4/26/2006, http://go.theregister.com/feed/http://www.theregister.co.uk/2006/04/26/international_phishing_survey/

[theregister.co.uk/2006/04/26/international_phishing_survey/](http://go.theregister.com/feed/http://www.theregister.co.uk/2006/04/26/international_phishing_survey/)

Antone Gonsalves, “Phishers Catch Victims with VoIP,” InternetWeek, 4/25/2006, <http://internetweek.cmp.com/news/186701099>

Rachna Dharmija, J. D. Tygar, Marti Hearst, “Why Phishing Works,” Harvard University and UC Berkeley, http://people.deas.harvard.edu/~rachna/papers/why_phishing_works.pdf

Good Intentions Gone Awry

Open source was
supposed to hinder malware.
So what happened?

Proponents of the free and unrestricted dissemination of malware samples and source code argue that ready access to this information speeds the development of countermeasures. Reviewing the history of computer viruses, however, it is clear that exactly the opposite is true. Wide and open distribution of malware samples—especially source code—is a problem, not a solution.

MS-DOS Viruses

Twenty years ago, the computing and virus landscape was very different. Few institutions and companies had Internet access, and they typically connected at no faster than 9,600 bits per second.

Back then, viruses spread from one computer to another almost exclusively via infected floppy diskettes. Infections spread when users either booted from an infected diskette (boot/MBR infectors) or when they ran or copied an infected program from it (file infectors). As Dr. Alan Solomon noted at the time, the fastest speed at which a virus could spread around the world was that of a commercial jet.

Yet even during this early era, the distribution of virus samples and source code exacerbated, rather than alleviated, threats. Figure 1 shows the cumulative growth of malware from 1988 to 1998.

By Dmitry Gryaznov



The first personal-computer viruses appeared in the mid-1980s. In 1986, a boot-sector infector called Brain and the VIRDEM “demo” virus were discovered. In 1987, Vienna, Lehigh, and others emerged. Figure 2 shows a family breakdown of MS-DOS file viruses by 1994.

In 1987, Ralf Burger published *Computer Viruses: A High Tech Disease*, a book that detailed the operation of computer viruses and contained source code for the Vienna (a.k.a. DOS-62) virus, which was prominent that year, and for several other viruses.

Many virus authors used the book’s Vienna source code as the foundation for their own creations. Many more modified the Vienna code, thus producing over 500 Vienna variants by the mid-1990s. At a time when all known viruses numbered in the low thousands, these new variants represented a dramatic increase.

In 1991, the infamous Bulgarian virus writer Dark Avenger created and released the Mutation Engine, called MtE or Dark Avenger’s Mutation Engine (DAME). It was advanced for its time. For the first virus he built using MtE (called Dedicated:MtE), Dark Avenger modified the Vienna source code slightly by adding polymorphic capabilities.

Other viruses from Burger’s book were also copied, modified, and released, including a primitive

overwriting virus. Overwriting viruses destroy their host programs while replicating, making them easy to spot. The overwriting virus from Burger’s book was too obvious to succeed on its own. But after publication of the virus source code, numerous variants were soon spreading. This virus family was eventually named after Burger, and today more than 100 variants are known.

By the end of the 1980s, Virus eXchange Bulletin Board Systems (VX BBSes) appeared and were used by virus writers to share their creations, knowledge of MS-DOS, infection techniques and ideas, and source code. With the help of those VX BBSes, virus writers organized into virus-writing groups, working together on new viruses and freely sharing source code and disassemblies.

Some of the most well-known groups were Association for Really Cruel Viruses (ARCV), Immortal Riot, Youth Against McAfee (YAM), Falcon-Skism (PS), and 29A (hexadecimal for 666). Some groups still exist, having moved from VX BBSes to VX FTP and VX WWW sites.

Some virus writing groups also published e-zines to share expertise and source code. Well-known e-zines included *Infected Voice*, *40hex*, and *Insane Reality*. VX BBSes and e-zines published source code for early widespread viruses such as Cascade and Jerusalem, which led to relatively high numbers of their variants. In fact, most

Cumulative Growth of Malware, 1988–1998

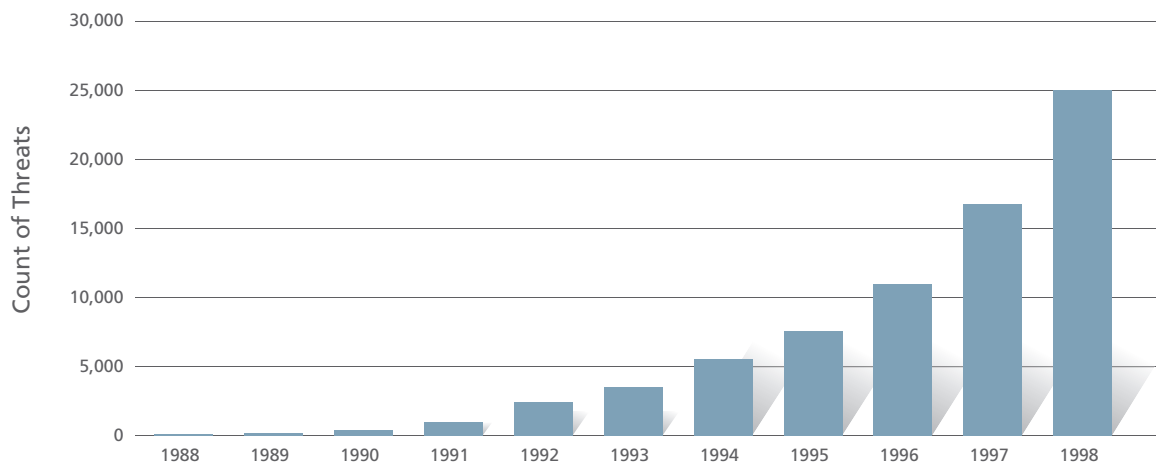


Figure 1: Cumulative growth of malware, 1988–1998

Source: McAfee® Avert® Labs

MS-DOS File Viruses by 1994

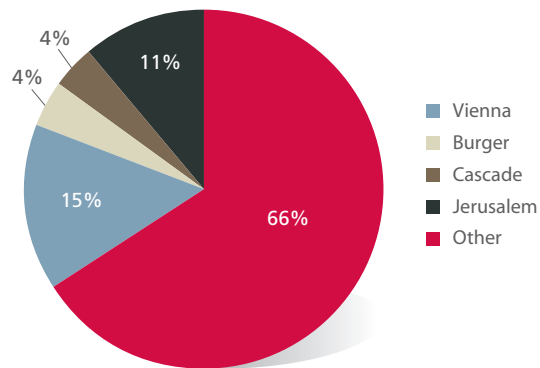


Figure 2: Breakdown of MS-DOS file viruses by 1994

Source: McAfee Avert Labs

viruses during this era originated either directly from virus-writing groups or were based on source code that was published in e-zines.

In the 1990s, virus-writing groups produced construction kits so that “wannabe” virus writers, who lacked the skill to write malware on their own, could create viruses in a semi-automated way. The construction kits, complete with instructions, were published in e-zines and on VX FTP and VX WWW sites and led to thousands of new virus variants.

With a typical virus construction kit, a user can select the type of the virus (overwriting, appending, etc.), target files (*.COM or *.EXE), trigger conditions (based on current date or time, infection counter, etc.), and the payload. The kit then produces a virus-assembly source code, which can be compiled immediately or further edited.

These kits generate template-based viruses with easily recognizable code, simplifying identification and classification. Popular kits included Virus Construction Set (VCS), Virus Creation Laboratory (VCL), and Phalcon-Skism Mass-Produced Code (PS-MPC). In the late 1990s, PS-MPC was responsible for the creation of nearly 15,000 viruses; all of which were uploaded both to VX sites and to anti-virus companies’ submission sites. Subsequent virus construction kits could produce Windows® viruses, macro viruses in Microsoft® Office applications, and script viruses. Figure 3 shows the family breakdown of MS-DOS viruses discovered by 1999.

By the end of the 1980s, Virus eXchange Bulletin Board Systems (VX BBSes) appeared and were used by virus writers to share their creations, knowledge of MS-DOS, infection techniques and ideas, and source code.

Macro and Script Viruses

With the release of Windows 95, MS-DOS was headed toward oblivion. At first, this seemed to herald a reduction in the malware threat. Most MS-DOS viruses—especially the prevalent boot/MBR infectors—were ineffective on the new operating system. Malware authors were still unfamiliar with the inner workings of the new operating system, and it took them several years to catch up.

MS-DOS File Viruses by 1999

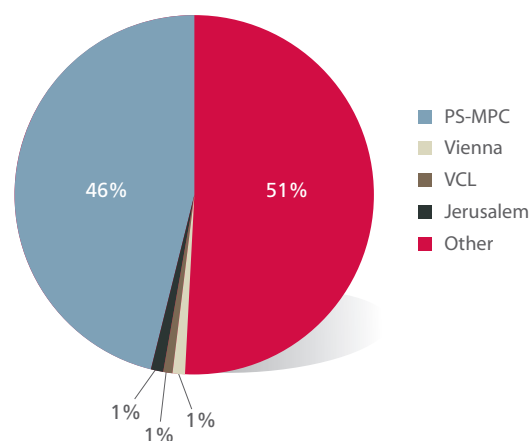


Figure 3: Breakdown of MS-DOS file viruses by 1999

Source: McAfee Avert Labs

Cumulative Growth of Macro Malware, 1995–2004

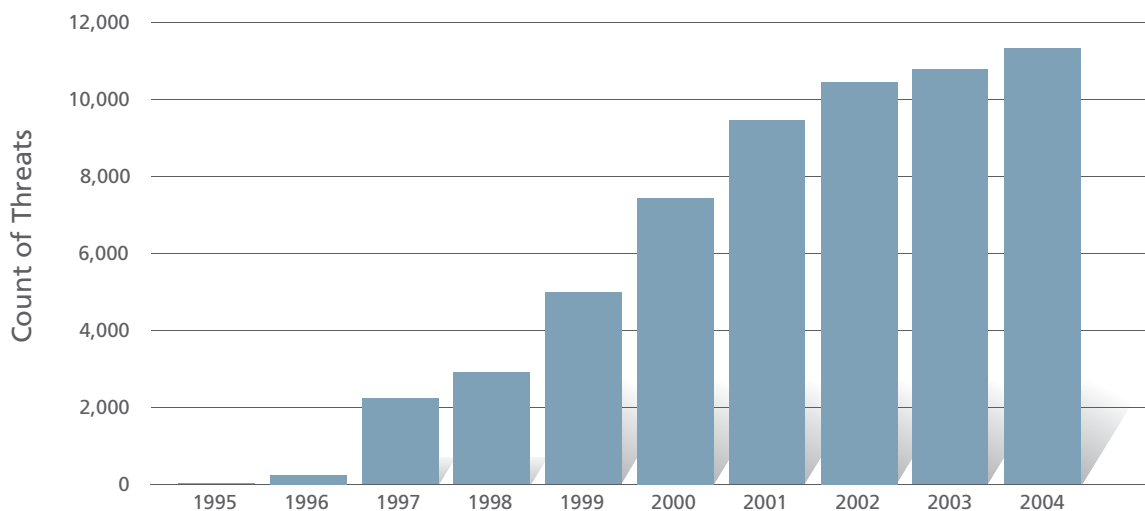


Figure 4: Cumulative growth of macro malware, 1995-2004

Source: McAfee Avert Labs

Unfortunately, around the release of Windows 95, a new threat appeared that would quickly become the most prevalent for several years. The Microsoft Word macro virus WM/Concept A signaled the beginning of a new era in malware development. The time had come for viruses that leveraged macro-capable products such as Microsoft Word, Excel, and PowerPoint.

Macro viruses, though predicted by the anti-virus community, posed a revolutionary threat. Previously “safe” inactive data files—documents, spreadsheets, and slide shows—were now executable and thus capable of carrying and spreading malware that was no less dangerous than traditional executable files.

More significant was macro malware’s ability to spread its own source code. Macro-capable applications were equipped with macro editors that could display a macro’s source code (including malware), modify it, export it as a text file, import a macro from a text file, copy macros between documents, etc. All of this could be accomplished automatically, using the macros themselves.

Easily created and modified macro malware effectively contained its own source code. Thus, it could be used as a virus-writing guide for

inexperienced programmers who lacked an in-depth knowledge of operating systems or computer architecture. Macro-capable software quickly became nirvana for virus writers and a nightmare for computer users. With the exception of viruses produced by construction kits, the computing world had never seen such huge numbers of variants created in such a short time (See Figure 4).

In spring 1999, macro virus writers discovered that the Office macro language, Visual Basic for Applications, which Microsoft also licensed for use in other applications, enabled control of the Outlook email client. The resulting worldwide outbreak of the Melissa mass-mailing virus encouraged copycat virus writers, and soon hundreds of Melissa variants and descendants followed.

The next generation of threat arrived with the advent of scripting languages such as JavaScript (JS) and Visual Basic Script (VBS). A program in these scripting languages is simply a text file of its own source code. The VBS Loveletter mass-mailing virus outbreak in May 2000 paved the way for hundreds of aspiring virus writers, and the term *script kiddie* was born.

Because script viruses were even easier to create and modify than macro viruses, a middle-school child

Since the early days of personal computing, it is clear that the widespread dissemination of malware source code has led to the spread of malware, rather than its diminishment.

with no knowledge of programming could trivially modify an existing script virus, using nothing more than the Notepad editor, and create a new variant.

To make virus generation even easier, virus writers produced construction kits for macro and script malware. Kits such as Word Macro Virus Construction Kit (WMVCK), Odysseus Macro VCK, VicodinES Macro Poppy Construction Kit (VMPCK), Senna Spy Internet Worm Generator (SSIWG), and Visual Basic Script Worm Generator (VBSWG) produced seemingly endless variants, any of which could cause an outbreak and wreak havoc. For example, one well-known mass-mailing VBS virus called Kournikova was created with VBSWG.

Scripts are also easy to embed into HTML. Users could now infect their computers simply by browsing to a malicious Web page or by previewing an email. Email attachments were no longer required to infect systems.

Present day

Improved security in Office applications, email clients, Web browsers, scripting hosts, and the Windows operating system in general would eventually diminish the number and severity of macro and script viruses. However, the ever-resourceful malware developers have, in the

ensuing time, improved their knowledge of Windows and other widely used Microsoft products, setting the stage for new generations of malware.

Other articles in this issue of *Sage* explore how contemporary malware is being developed, improved, and disseminated as the result of collaborative efforts. These modern-day, open-source projects are simply the latest incarnation of an enduring culture of sharing that exists among malware authors. From the time of Vienna and Burger viruses to the present day, the destructive cycle of variant outbreaks associated with malware source-code publication has demonstrated an acute weakness in the full disclosure security model. Unfortunately, in the malware world, good intentions can be just as damaging as bad ones. Since the early days of personal computing, it is clear that the widespread dissemination of malware source code has led to the spread of malware, rather than its diminishment. [Sage](#)

About the author

Dmitry Gryaznov is a Senior Research Architect at McAfee Avert Labs. He has been involved in anti-virus research since 1987 and is a regular participant and speaker at computer anti-virus conferences.



Money

Changes Everything

Malware authors leverage open-source model for profit.

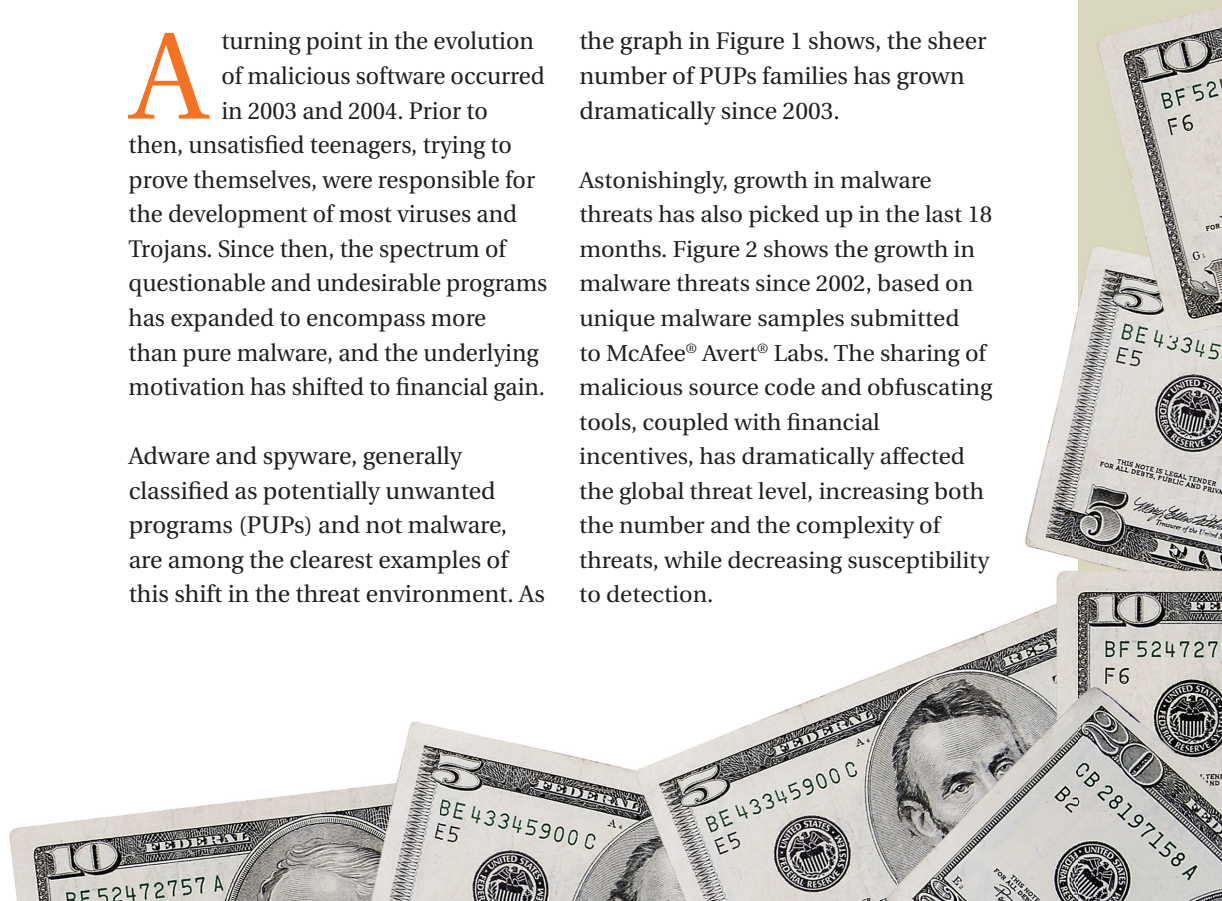
By Igor Muttik

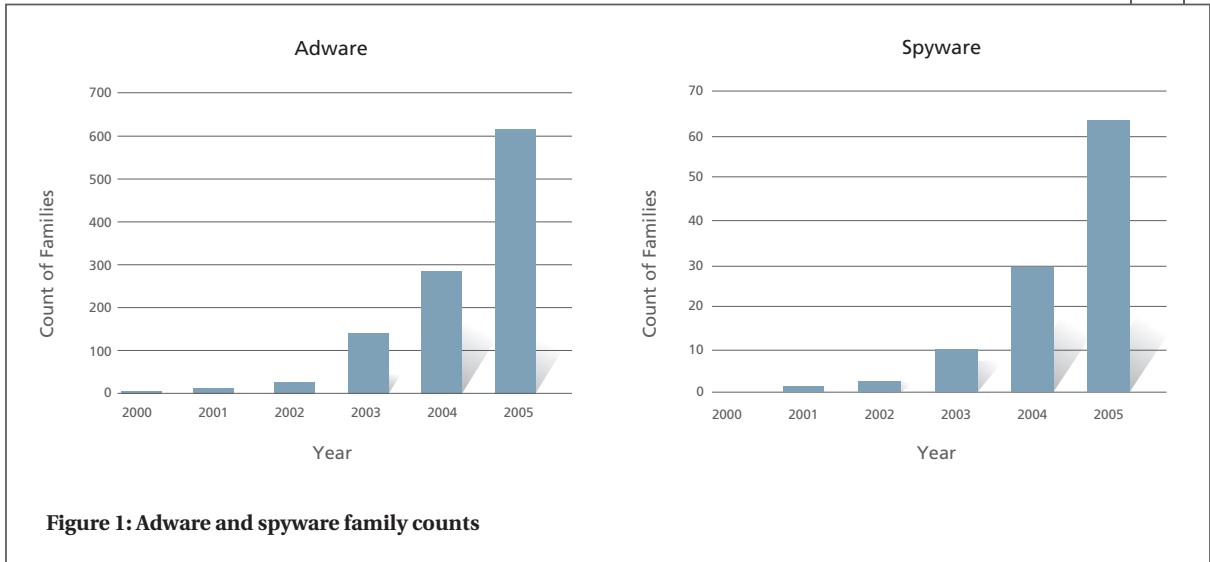
A turning point in the evolution of malicious software occurred in 2003 and 2004. Prior to then, unsatisfied teenagers, trying to prove themselves, were responsible for the development of most viruses and Trojans. Since then, the spectrum of questionable and undesirable programs has expanded to encompass more than pure malware, and the underlying motivation has shifted to financial gain.

Adware and spyware, generally classified as potentially unwanted programs (PUPs) and not malware, are among the clearest examples of this shift in the threat environment. As

the graph in Figure 1 shows, the sheer number of PUPs families has grown dramatically since 2003.

Astonishingly, growth in malware threats has also picked up in the last 18 months. Figure 2 shows the growth in malware threats since 2002, based on unique malware samples submitted to McAfee® Avert® Labs. The sharing of malicious source code and obfuscating tools, coupled with financial incentives, has dramatically affected the global threat level, increasing both the number and the complexity of threats, while decreasing susceptibility to detection.

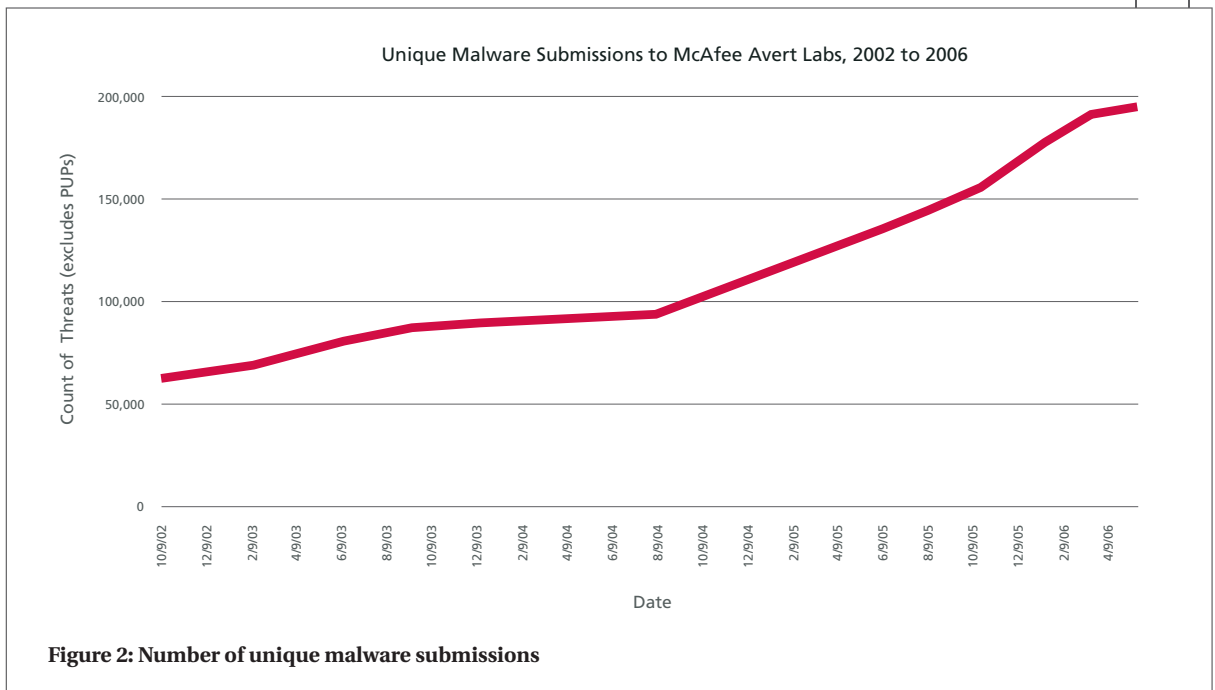




Source: McAfee Avert Labs

The prevalence of Internet Relay Chat (IRC) bots is a substantial component of the malware growth shown in Figure 2. The term *bots* is adapted from *robots* because bots are typically automated scripts or programs that execute a series of instructions based upon pre-defined stimuli. Contemporary malware bots are similar in that they autonomously crawl the Internet, looking for vulnerable computers. Once one has been found, a bot launches a scripted exploit against it, typically obtaining root-

level privileges. The bot is then able to start processes, inject malicious code, and turn the computer into a new drone for its master. Drones periodically communicate back to the master, or bot herder, for further instructions. These bot herds are often used for a variety of nefarious purposes, including Denial of Service (DoS) attacks, spamming, disseminating malware, and collecting confidential information. IRC bots are particularly popular because they communicate through a distributed network of IRC servers that



Source: McAfee Avert Labs

Financial Motivation

There are several motivations for bot herders, but only the last two in this list are not implicitly driven by financial gain.

- Launching a Distributed Denial of Service (DDoS) attack on a Web site (for extortion)
- Generating revenue through adware installations on controlled PCs
- Generating revenue through pay-per-click schemes and spoofing affiliate schemes
- Installing other malware, such as backdoors and password- or data-stealing Trojans (including credit card data, banking login details, PayPal, Nohex, etc.)
- Installing SMTP proxy software to propagate spam
- Stealing data from computers and networks (often for identity theft)
- Reselling botnets (for DDoS attacks or spam relaying)
- Using a distributed network of computers to crack passwords
- Controlling a large number of computers to participate in a bot war

There is significant evidence to confirm that financial motivation underlies bot development.¹

- McAfee Avert Labs' analysis confirms that certain variants of W32/Mytob have been involved in downloading the following adware: Adware-ISTbar, Adware-BB, Adware-Websearch, Adware-RBlast, Adware-SAHAgent, Adware-WinAd, Adware-SideFind, Adware-180Solutions, Adware-DFC, and Adware-ExactSearch
- Many W32/Sdbot variants are distributed along with SMTP proxies (Proxy-FBSR and Proxy-Piky)
- The German magazine *c't* reported in February 2004 that it was able to buy access to a botnet from Peter White, a.k.a. "iss," a developer of chat-server software. White offered the services of his botnet to distribute spam for U.S. \$28,000 a month²
- The same story also quotes a Scotland Yard source who said, "Small groups of young people are creating a resource of 10,000 to 30,000 computers networked together and are renting them out to anybody who has the money." According to a Reuters report from July 2004, the going rate is as low as U.S. \$100 an hour. There is also evidence that the IP addresses of

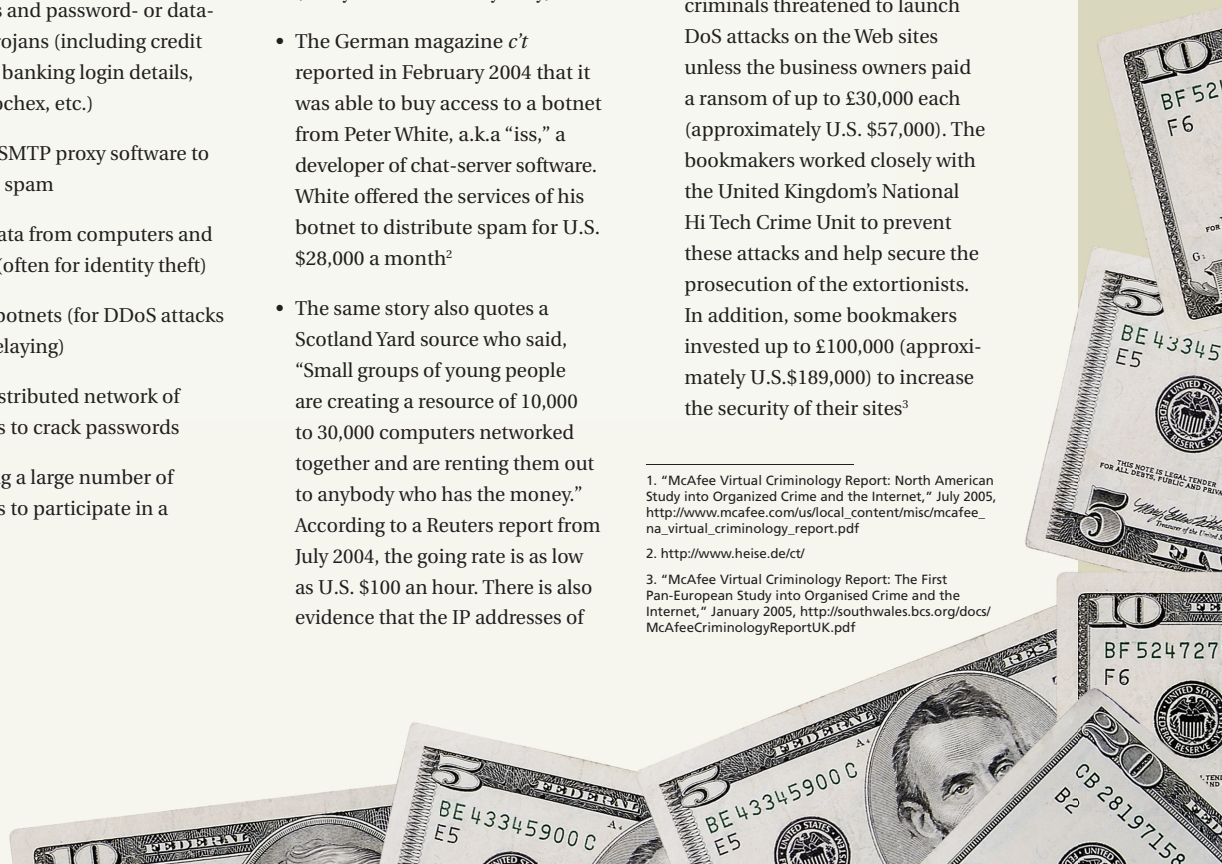
these machines are traded in chat rooms. Virus authors and middlemen administer payment through anonymous accounts or by using Western Union money transfers. Organized criminals can use these networks to run huge DoS attacks or distribute spam with viruses that steal credit card and/or bank details from unsuspecting users

- In September 2004, Norwegian Internet provider Telenor shut down a botnet consisting of approximately 10,000 machines. The botnet was ready to perform DoS attacks and hacking attempts on numerous computers and networks³
- In March 2004, criminal syndicates operating from Russia targeted betting Web sites worldwide. The criminals threatened to launch DoS attacks on the Web sites unless the business owners paid a ransom of up to £30,000 each (approximately U.S. \$57,000). The bookmakers worked closely with the United Kingdom's National Hi Tech Crime Unit to prevent these attacks and help secure the prosecution of the extortionists. In addition, some bookmakers invested up to £100,000 (approximately U.S. \$189,000) to increase the security of their sites³

1. "McAfee Virtual Criminology Report: North American Study into Organized Crime and the Internet," July 2005, http://www.mcafee.com/us/local_content/misc/mcafee_na_virtual_criminology_report.pdf

2. <http://www.heise.de/ct/>

3. "McAfee Virtual Criminology Report: The First Pan-European Study into Organized Crime and the Internet," January 2005, <http://southwales.bcs.org/docs/McAfeeCriminologyReportUK.pdf>



effectively cloaks the bot herders from detection. Since 2004, IRC bots have grown from 3 percent to 22 percent of all malware threats (see Figure 3). A raw count of active IRC bots is equally alarming (see Figure 4).

The use of IRC bots in for-profit ventures, such as spamming and collecting confidential information, has forced malware authors to improve the quality and features of their code. To those ends, malicious programmers are adopting coding practices and controls akin to those in the legitimate software development world.¹ They are also adapting social and professional norms, established by the open-source community, to develop malware, and capitalizing on the widespread availability of source code for many of the Internet's popular malware families. There is little doubt that this availability has dramatically increased the general threat level and pervasiveness of specific threats. Ironically, this same dynamic led to the proliferation of open-source software and its resulting benefits.

IRC-Sdbot

The IRC-Sdbot Trojan demonstrates how a single piece of malware can have a ripple effect. IRC-Sdbot originally appeared in the first quarter of

2002,² but its source code has become the basis for thousands of other IRC bots as programmers have added functionality.

Although W32/Sdbot began as a non-propagating Trojan, later strains employed spreading mechanisms, such as share-hopping, the use of numerous exploits, and weak password detection. Some of these strains received separate names such as Rbot, Forbot, and Wootbot.

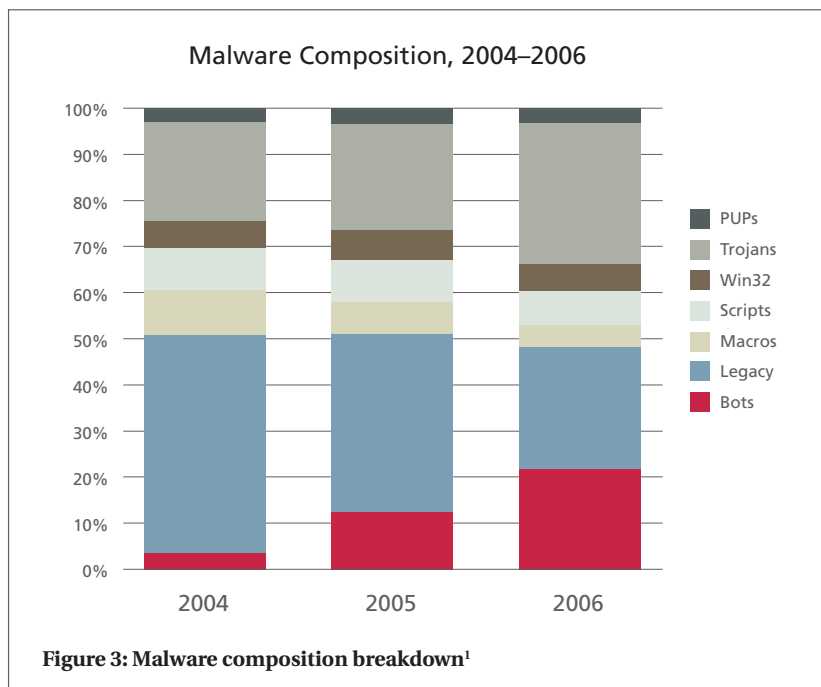
Two other large IRC bot families experienced similar shared development: W32/Spybot (first appearance in May 2003) and W32/Gaobot (first appearance in October 2003, with a latter strain dubbed Phatbot). New versions of source code for these families are readily available on the Internet. By adding new functionality, programmers fuel the generation of even more variants. Each of these bot families now includes thousands of variants.

Bolting Threats Together

There is a growing trend toward compound threats, or bundled malicious tools. For example, W32/Mytob was created by bolting a mass-mailing routine from W32/Mydoom to the W32/Sdbot.

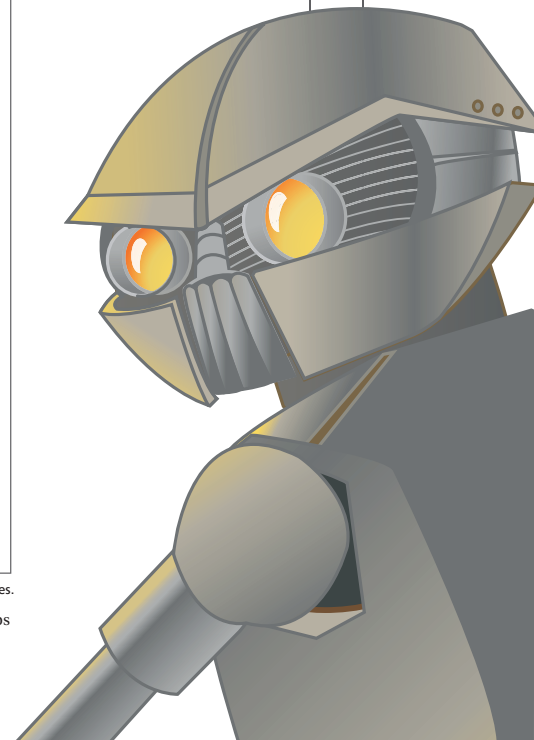
1. See Michael Davis' "Building Better Bots" in this issue.

2. http://vil.nai.com/vil/content/v_99410.htm



1. In the graphs, "legacy" denotes DOS, boot-sector, and Windows 3.1 viruses.

Source: McAfee Avert Labs



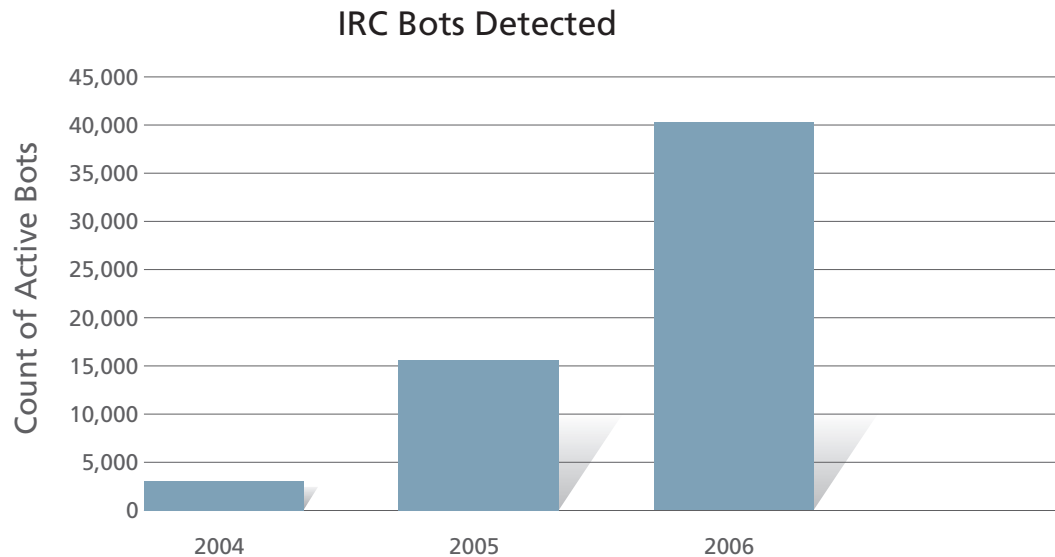


Figure 4: IRC bots catalogued as of March 2004, 2005, and 2006²

2. These numbers include field reports, samples from honeypots, and submissions from anti-virus companies. It is not possible to determine if all samples originated from true field infections.

Source: McAfee Avert Labs

The W32/Mydoom family

has hundreds of variants, significantly more than a typical malware family, and that is most likely due to the widespread availability of its source code.

In January 2004, the mass-mailing W32/Mydoom virus created one of the worst virus outbreaks ever.³ Another virus, W32/Doomjuice, distributed W32/Mydoom's source code in February 2004.⁴ Undoubtedly, when bot writers sought to add a mass-mailing feature, the field success of W32/Mydoom and the easy availability of its source were important factors in its selection. This started the new W32/Mytob family in March 2005.

Figure 5 illustrates the ebb and flow of new variants in the W32/Mydoom and W32/Mytob families. The graph shows that the W32/Mydoom family has hundreds of variants, significantly more than a typical malware family, and that is most likely due to the widespread availability of

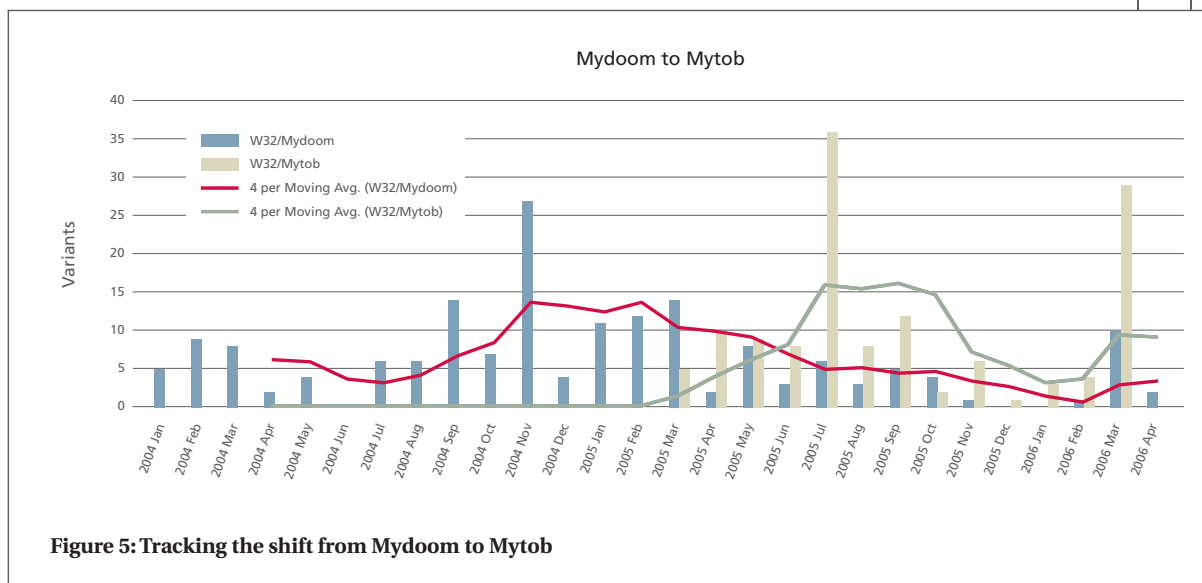
its source code. The graph also shows that the creation of W32/Mytob (achieved by merging the W32/Mydoom source into W32/Sdbot) revitalized the generation of new variants.

Around the spring of 2005, W32/Sdbot went through yet another incarnation when it acquired the FURootkit backdoor to cloak itself from users and from anti-virus scanners. Statistics from the Microsoft® Malicious Software Removal Tool show that FURootkit was the most common non-replicating malware at the end of 2005,⁵ due to its inclusion in W32/Sdbot. On its own, FURootkit does not have a propagation mechanism; thus, W32/Sdbot served as the principal springboard for its dissemination.

3. http://vil.nai.com/vil/content/v_100983.htm

4. http://vil.nai.com/vil/content/v_101002.htm

5. J. Garms, "An Accurate Understanding of On-Going Malware Presence," AVAR 2005, Tianjin, PRC.



Source: McAfee Avert Labs

Multi-component threats are growing, with obfuscation tools being the latest added feature. For example, the Swizzor Trojan was packed with the packer UPC to protect it from detection. The perennial Hacker Defender rootkit has a special relationship with a packing tool called Morphine. Packing tools, as we shall see, are another obfuscation technique that is shared within the malware community. These tools help malware evade anti-virus scanner detection, extending the useful life of both new and old malware.

Packers and Security Envelopes (Protectors)

In general, packers are a class of software that compress files to save disk space, such as Winzip. Malware developers, however, are not interested in compression so much as a by-product of packers' use. Packed files no longer resemble the originals on a binary level, but are still recoverable and executable, thus making them effectively hidden from signature-based anti-virus scanners. For anti-virus scanners to detect packed malware, they must be able to inspect compressed data. Applying a packer to 1,000 worms effectively creates 1,000 new, undetectable worms, unless the scanners can see through the packing.

Security envelopes, also known as protectors or cryptors, are another set of tools used by malware authors for obfuscation. Though they have legitimate uses, such as protecting games from

copyright abuses, these tools extend the useful life of malware by hindering analysis and reverse engineering of samples through numerous anti-debugging and anti-emulation tricks.

Applying packers and protectors requires little skill. Combined with their relative effectiveness, it is a recipe for rapid malware growth. Malware authors often employ several layers of packers and protectors. They also use commercial software installers such as InstallShield, Wise, and Nullsoft to obfuscate malware. Figure 6 shows the growth in the number of packers (counting only major versions, protectors included) and the proportion of packed malware. Today, packers and/or protectors obfuscate more than 50 percent of malware.

Figure 7 shows the life cycle of the Morphine packer (discussed later). As with other malware, packers have a predictable life cycle with a popularity window that eventually tails off as they become less effective at obfuscating malware. However, packers have a longer useful lifespan than most malware because engineering a solution for each packer is more complex than adding a malware signature to an anti-virus scanner.

Most packers are used almost exclusively for malware concealment. Figure 8 shows the prevalence of seven obfuscating tools (packers and protectors) as a percentage of all packed malware samples submitted to McAfee Avert Labs.

Packers on the rise include PE-Compact2, NSPack, and UPack v>2. Although a few packers experience stable usage, the most popular packers tend to gradually vanish as they lose their effectiveness.

Let's examine an example of how the desire to prolong the useful life of malware led to the development of an obfuscating tool and an entire community to support it.

Malware Development Communities

One popular Web site devoted to rootkit development is the Slovakian "Hacker Defender" site,⁶ which carries many versions of the Hacker Defender rootkit source code and hosts development discussions.

6. <http://www.hxdef.org/>

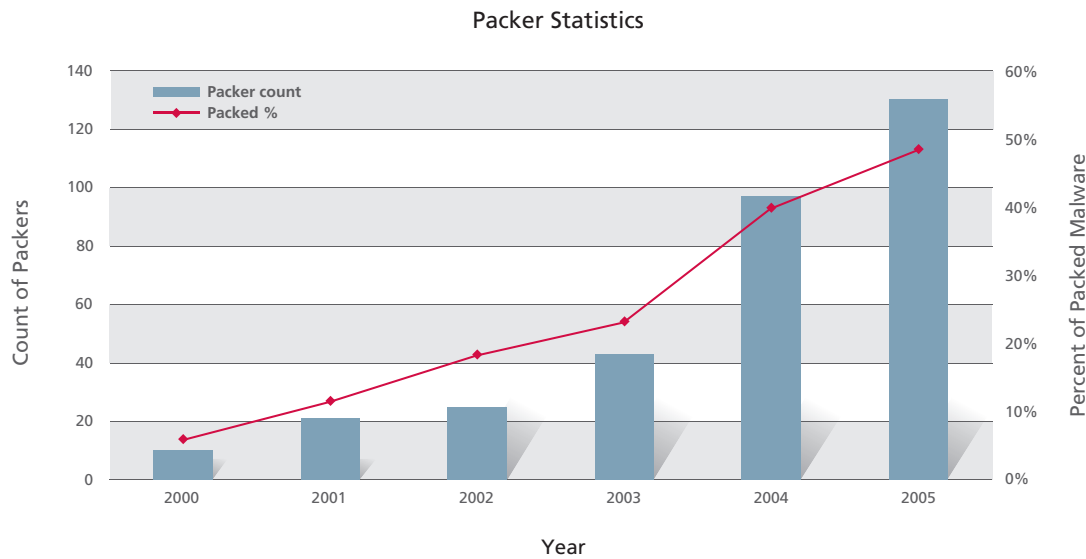


Figure 6: Count of packers and prevalence of packed malware

Source: McAfee Avert Labs

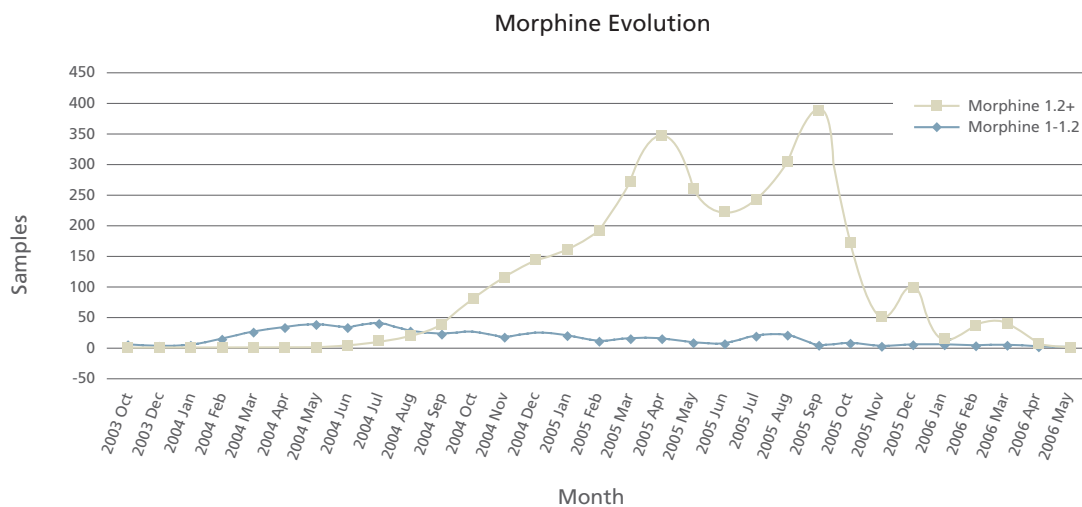
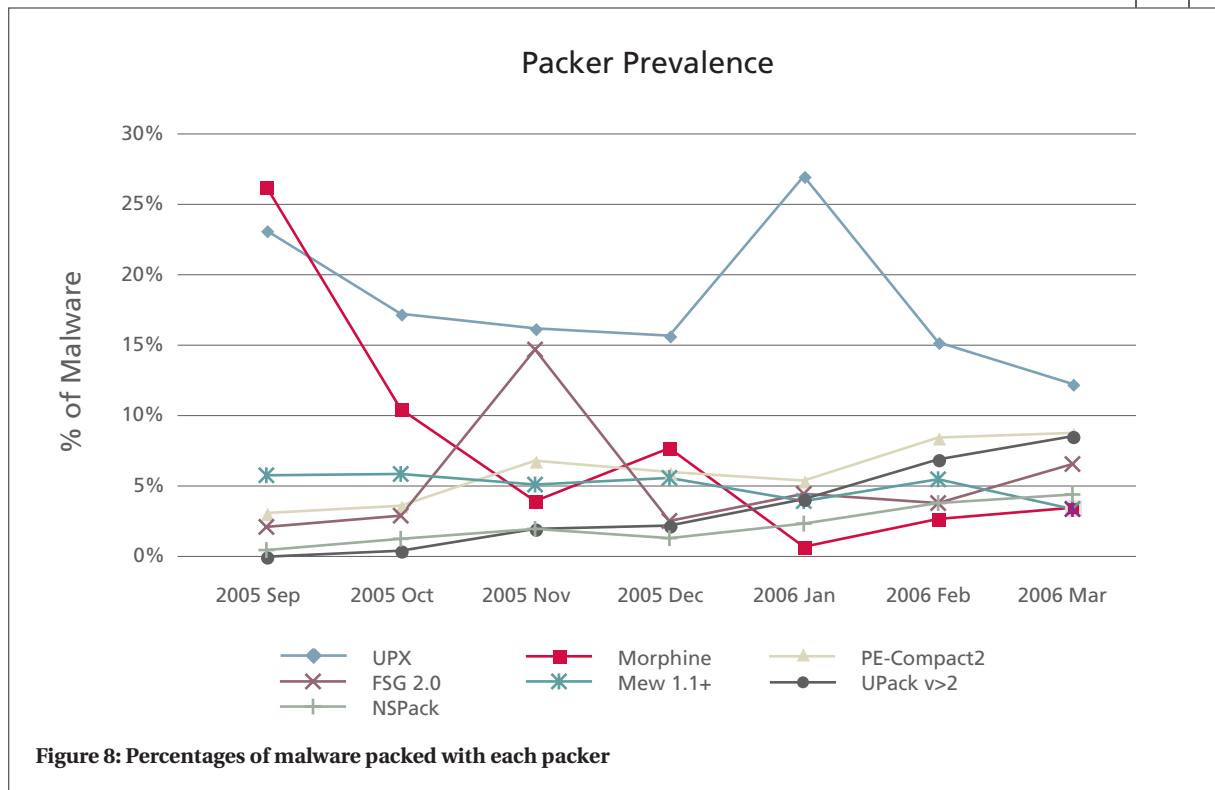


Figure 7: Evolution of the Morphine packer

Source: McAfee Avert Labs



Source: McAfee Avert Labs

Below is a statement from the Hacker Defender Web site.

Welcome to www.hxdef.org - home of the Hacker defender project. This Project was started in 2002. Many things has [sic] changed since then. There are two main subprojects on this page. First is Hacker defender Windows NT rootkit. The second is PE module encryptor for Windows NT called Morphine, its purpose is to protect PE modules against antivirus detection. Both of these programs are free and open source but both have their paid versions which can be ordered in antidetection section as antidetection service. There are also other open source programs in download section. They mostly implement some undocumented NT stuff for which is hard to find a good source code on the net.

This statement is revealing. It is a clear example of an open-source threat development project, and demonstrates how malware authors have embraced the open-source model and are actively sharing and contributing to each others' development efforts. It also shows that this loosely organized community and its polymorphic Morphine packer project is

engaged in an arms-race style of competition with commercial anti-virus software vendors.

Conclusion

The growth of bots is due to two factors—financial motivation and the availability of source code. Without financial incentives, there would be far fewer variants. The financially neutral Mydoom family, for example, has far fewer variants than any major bot family. Also, without large-scale source-code sharing, we would not see the handful of massive families that we have today. Rather, we would expect to find many small families, reflecting the individual efforts of separate researchers.

But the malware community is no longer a scattered army of individual hobbyists. The addition of funding from successful botnet deployments and the leveraging of open-source tools and techniques have created a formidable machine for the creation, modification, and distribution of threats. [Sage](#)

About the author

Igor Muttik is a Senior Research Architect at McAfee Avert Labs. He speaks regularly at security conferences and has been researching viruses since 1987. Igor received his PhD in Physics and Mathematics from Moscow University in 1989.

Open-Source Software in Windows Rootkits

By Aditya Kapoor

The open-source environment has always offered a wealth of information to researchers and enthusiasts. It is a common ground for candid criticism and in-depth research by beginners and experts. However, like any other powerful tool, it can be misused.

During the last few years, many security companies and researchers have turned their attention to the problem of rootkits, which is a fast-growing real-world threat.¹ Rootkits are malware that employ stealth techniques to conceal their malicious files and processes from users and, in most cases, from anti-virus scanners.² Finding new rootkit techniques is broadly analogous to finding new system vulnerabilities because a new stealth technique, like a vulnerability, can be bundled into a threat, making it more effective and potent. For example, in the case of FURootkit,³ once the author announced details of his newly discovered technique for hiding files and processes, malware writers quickly adapted their software to incorporate this new hiding technique.⁴

Security research is a sensitive field, and the public disclosure of an unsecured environment or of a new method for breaking into computer systems has a significant negative impact. As a result, most vulnerability researchers reduce the chance that malicious code will exploit their discoveries by contacting in advance the vendor whose product has the vulnerability. This cautious step gives businesses a chance to protect their users before a vulnerability is made public. Unfortunately, on more than one occasion, such prudent measures were not followed, resulting in the release of Trojans or viruses that exploited unpatched vulnerabilities as soon as the demonstration source code was published.⁵

The McAfee® whitepaper *Rootkits, Part 1 of 3: The Growing Threat* discusses how rootkits are proliferating at a rapid rate and speculates that their sophistication will increase due to open-source collaboration. In this article, I will substantiate that claim by analyzing the prevalence of open-source rootkits in known malware and also encourage a

1. Aditya Kapoor, "Rootkits, Part 1 of 3: The Growing Threat", McAfee Inc., April 2006, http://download.nai.com/products/mcafee-avert/WhitePapers/AKapoor_Rootkits1.pdf

2. Ibid.

3. Fu, <http://rootkit.com/project.php?id=12>

4. Furootkit, vil.nai.com/vil/content/v_127131.htm

5. W32/Mydoom.AH@MM exploited Microsoft Internet Explorer IFRAME buffer overflow vulnerability (KB889293) six days after it was discovered. http://vil.nai.com/vil/content/v_129631.htm

Fifty percent of the studied open-source rootkits can be found in malware from the wild.

candid discussion of prudent public disclosure policies for stealth techniques as their role in malware continues to grow.

Background on Finding Rootkits in the Wild

Any time an anti-virus company collects a new malware program or has one submitted for analysis, it is characterized as found *in the wild*. One approach to test the prevalence of publicly available stealth techniques in malware is to see how closely they compare to known malware.

Comparing any two programs can be considered a program equivalence problem.⁶ Two programs are considered equivalent if, given a set of identical input parameters, the two programs produce identical results. However, because we are dealing with stealth modules that are generally embedded within rootkit malware, equivalence is not particularly applicable.⁷ Rather, we elect to treat this problem as a clone detection problem.⁸

We take two approaches:

1. Compare parts of stealth-technique source code to known malware binary files.
2. Compare parts of stealth-technique compiled in binary form to known malware binary files.



Comparing a piece of source code with a binary, or two binaries, using static-analysis techniques may seem trivial, but the ease is deceptive. Most binaries today are compiled with modern-day compilers that employ various optimization techniques, such as instruction reordering, live variable analysis, and inline function analysis. The output binaries from these compilers can vary greatly for the same piece of source code, depending upon the settings and the individual compiler used. Thus, compiling publicly available rootkit source code may not yield a binary file that is recognized as a clone of a known malware file, even if both share the same original source. Complicating matters further, malware authors sometimes add obfuscations into their code to make matches unlikely.⁹ These compiler issues and obfuscations make the problem of clone detection time-consuming and challenging.¹⁰

In *Evaluating Clone Detection Techniques*, the authors explain that using basic string and literal matching may be a quick and crude, yet effective, way to compare two programs. Because most anti-virus scanners base their detections on sequence or string matching, the scanners are a good starting point for comparing two binary files. However, we fall back on basic string and literal matching techniques when dealing with uncompiled source code because of the issues that surround comparing the binaries when they are derived from different compilers.

6. <http://www.cs.princeton.edu/introcs/77/computability/>

7. "The Complexity of the Equivalence Problem for Simple Loop-Free Programs." http://locus.siam.org/SICOMP/volume-11/art_0211002.html

8. *Evaluating Clone Detection Techniques*. <http://prog.vub.ac.be/FFSE/Workshops/ELISA-submissions/04-VanRysselberghe-full.pdf>

9. *Obfuscation of Executable Code to Improve Resistance to Static Disassembly*. <http://www.cs.arizona.edu/solar/papers/CCS2003.pdf>

10. *Evaluating Clone Detection Techniques*. Op. Cit.

Our Approach

To test rootkits, we visited a number of well-known sites to gather our open-source collection:

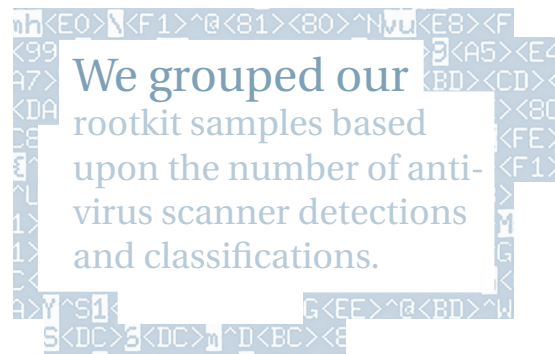
- 1 <http://www.ntkernel.com/wprod.php?ids=3>
- 2 http://www.megasecurity.org/Tools/NT_rootkit_all.html
- 3 <http://www.hxdef.org/download.php>
- 4 <http://www.rootkit.com>
- 5 <http://yythac.com>
- 6 <http://www.tibbar.org>
- 7 <http://www.iamaphex.net>
- 8 <http://www.egocrew.de>
- 9 <http://www.nuclearwinter.mirrorz.com>

Most of these Web sites host a subset of rootkits from other, more popular sites, giving us plenty of rootkits from which to choose. We downloaded 24 packages containing rootkit code or data, and then searched for their presence in the wild.

We gathered our results using two methods. We downloaded compiled binaries from these public Web sites and scanned them using a number of popular anti-virus scanners. In the event that a binary file was unavailable, we took the source code, compiled it, and ran the same battery of anti-virus tests against the compiled code. If no match was found, we engaged in a more detailed analysis, comparing the strings or literals of the source code with known malware samples from McAfee Avert® Labs. If we found a match, then we passed the associated malware sample through a battery of anti-virus products to see how many and what kind of detections occurred.

Classifying Compiled Binaries

If a downloaded package contains a binary file detected by more than three anti-virus scanners, it has a fair chance of being found in malware in the wild. As shown in Table 1, we can group open-source rootkit code based upon the numbers of anti-virus scanners that detect them.



Detection, however, does not guarantee that the sample is malware. Anti-virus scanners often detect binaries that are not necessarily confirmed malware, but that may nevertheless pose a security risk. Such files are generally called *applications*, as opposed to *Trojans* or some other malware category.

In Table 1, we grouped our rootkit samples based upon the number of anti-virus scanner detections and classifications. We assigned an overall classification from the most severe scanner rating. For example, if more than two products classified a sample as a Trojan, then we assumed that the rootkit was seen as a Trojan in the wild and therefore considered it a Trojan. When the source code was not traceable—that is, only binaries were available—we noted this and used it as a secondary distinguishing criterion, reasoning that the lack of available source would make it difficult to modify and incorporate into malware.

Results

Of the 24 packages downloaded and analyzed, the 12 found in Group 1 are most likely to have been used in malware because both binaries and source code are available; 10 or more anti-virus scanners detected them; and at least two anti-virus products classified them as a Trojan. These packages include well-known malware building blocks, such as AFXrootkit, PWS-Progent, and HackerDefender, all of which have been found to be highly prevalent.¹¹

Group 2 contains samples that, though recognized by several anti-virus scanners, were generally

11. Aditya Kapoor, Op. Cit.

Source-file analysis

Package Name	Detection Name	Binary Available	Source / Configuration File	Malware Classification	Number of Anti-Virus Detections
Group 1					
FU_Rootkit.zip	FURootkit	YES	YES	Trojan	20
hxdef100r.zip	HackerDefender.gen	YES	YES	Trojan	20
ntrootkit122.rar	AFXrootkit.gen / Trojan.Win32.Madtol.a*	YES	YES	Trojan	20
Rk_044.zip	NTRootKit-F / Backdoor.Win32.NTRootKit.044*	YES	YES	Trojan	20
He4Hook215b6.zip	He4Hook	YES	YES	Trojan	19
AFXRootkit2005.zip	AFXrootkit	YES	YES	Trojan	18
OpPorts12.zip (co-written by a member of the 29A group)	HackerDefender	YES	YES	Trojan	18
FUTo_enhanced.zip	FURootkit	YES	YES	Trojan	15
HideProcessHookMDL.zip	Backdoor-CSS	YES	YES	Trojan	15
JiurlPortHideDir_EN.zip	PWS-Progent	YES	YES	Trojan	15
Uay.zip	NTRootKit-V	YES	YES	Trojan	15
migbot.zip	Backdoor.Win32.Agent.uq* / Troj/RKProc-E **	YES	YES	Trojan	10
Group 2					
mjsrpk.zip	Generic BackDoor.m	YES	NO	Trojan	15
Procmagic.zip	HideApp	YES	YES	Application	16
Vanquish-0.2.1.zip	Vanquish	YES	YES	Application	15
faker11.zip	Tool-FileFake	YES	YES	Application	10
Ntillusion.zip	Application / Generic PUP.b	YES	YES	Application	9
Group 3					
Kircbot.zip	RootKit-KIrcBot	YES	YES	Trojan	1
cfsd.zip	Hacktool.Rootkit ***	YES	YES	Application	2
basic_hook_hide_proc.zip	New malware.z / Troj/RKProc-Fam**	YES	YES	Application	2
phide.zip	Demo-Prochide	YES	YES	Application	1
Group 4					
winlogonhijack-v0.3-src.rar	-	NO	YES	-	0
arcbot.zip	-	NO	YES	-	0
Bytehook.zip	-	YES	YES	-	0

* Kaspersky's detection name

** Sophos' detection name

*** Symantec's detection name

Table 1: Classification of 24 open-source rootkits

classified as applications instead of Trojans. This likely means that though the source code is public, no malware samples containing them have been found in the wild. The one exception is mjsrpk.zip, which was detected as a Trojan, but its lack of generally available source code calls into question whether it has been shared broadly for use in malware.

Based upon this approach, the packages in Groups 3 and 4 are unlikely to have been incorporated into malware. Group 3 consists of those stealth techniques that are at least recognized by anti-virus scanners, but for which malware has not yet been detected. Finally, Group 4 members are not detected by any scanners and are, in most cases, only available in source-code form.

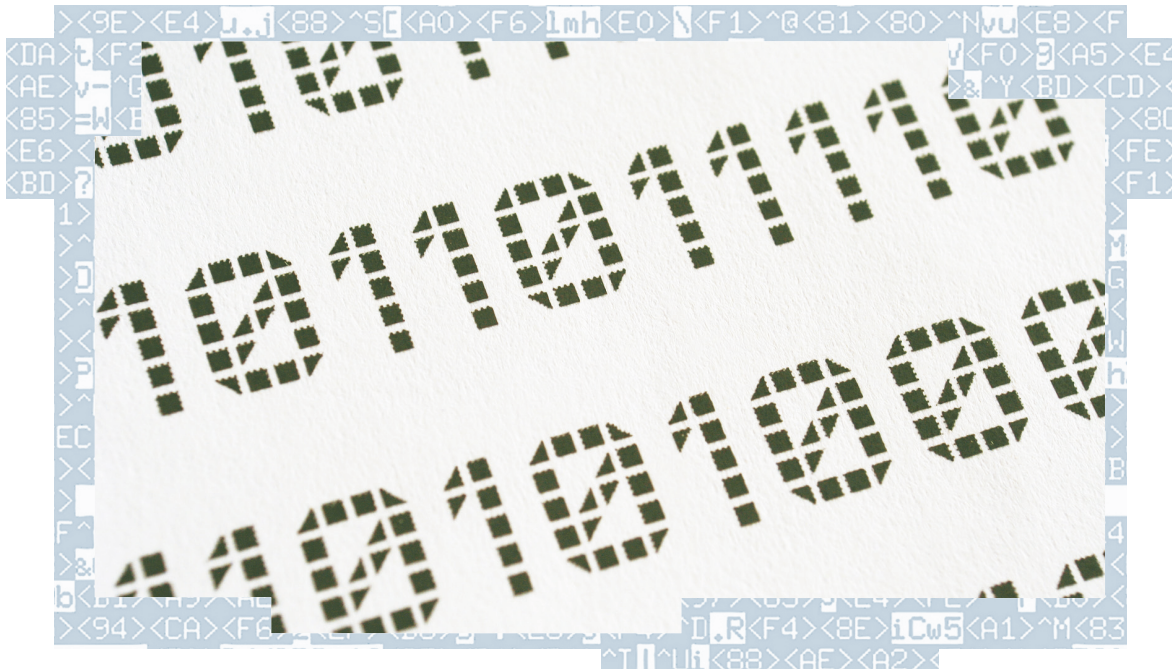


Table 2 shows examples of four packages that we successfully traced back to at least one Trojan family by analyzing their source code and comparing it to the strings or literals in Trojan samples. The rootkit code sharing is evident with very similar source code found in Backdoor-CPX and PWS-Progent variants. HideProcessHookMDL.zip appears to have been used as device drivers in Backdoor-CSS. The two cases have similar strings, both use kernel system service descriptor table (SSDT) patching techniques, and both use process-hiding techniques.

Package	Maps To
HideProcessHookMDL.zip	Backdoor-CSS
winlogonhijack-v0.3-src.rar	HackerDefender
FU_Rootkit.zip	Sdbot
JiurlPortHideDir_EN.zip	Backdoor-CPX and PWS-Progent

Table 2: Publicly available packages traced to known malware

Conclusion

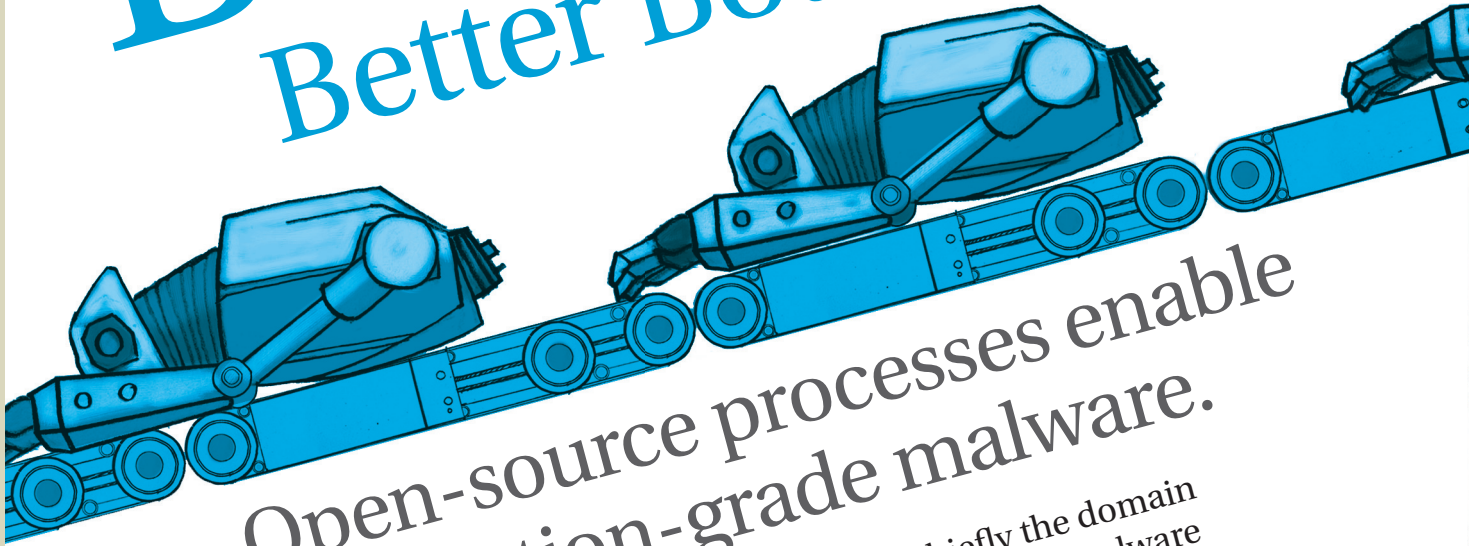
Fifty percent of the studied open-source rootkits can be found in malware from the wild. Our method also shows the relative popularity of the different rootkits in the wild—that is, the more scanners that detect a package, the more popular it is among malware authors.

A researcher gains credit for discovering a new stealth technique, just as with a new vulnerability, by publicizing the finding and being generally recognized as the first to do so. As this study demonstrates, publicized stealth technologies are quite common in malware today, and the trends indicate that this role will only grow. Because these technologies can be used to enhance malware, they should be handled with care and potentially be covered by similar disclosure guidelines as those used for vulnerabilities today. The virulence and invasiveness of rootkits has even caused Microsoft® to warn that a day may soon come when the only way to cure an infection is to reinstall a clean operating system. Given such high remediation costs, to say nothing of the direct consequences of an infection, the industry should quickly establish procedures for the disclosure of new stealth techniques to preserve security. Sage

About the author

Aditya Kapoor is a Research Scientist at McAfee Avert Labs. He has expertise in program analysis and disassembly techniques, and his research interests include program comparison, rootkit analysis and mitigation, and code behavioral analysis.

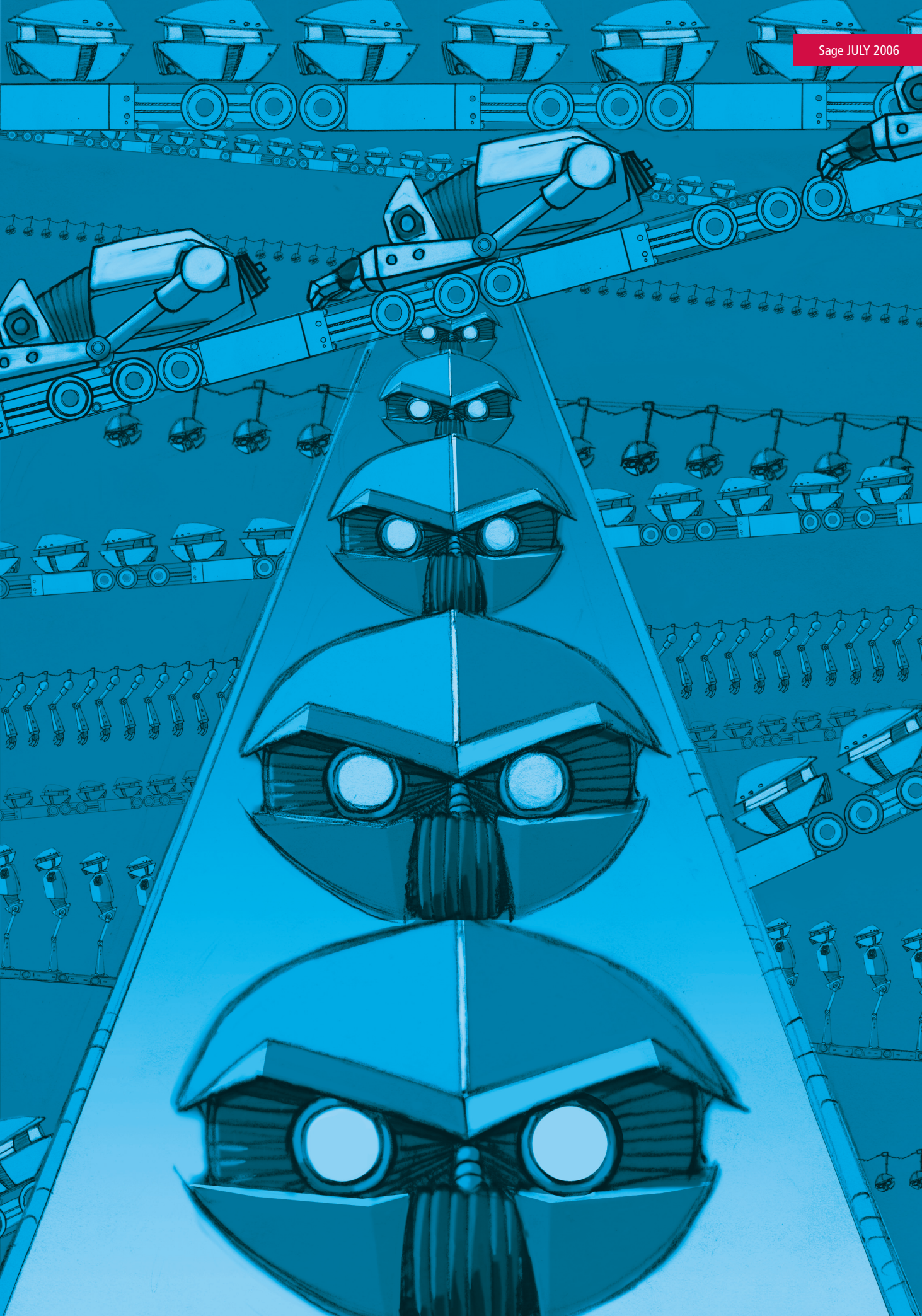
Building Better Bots



Open-source processes enable production-grade malware.

Until recently, Internet threats were chiefly the domain of amateur enthusiasts and hobbyists. Most malware was poorly written and was intended, more than anything, to establish bragging rights. As Igor Muttik points out in “Money Changes Everything,” however, this landscape has changed dramatically. The advent of financial incentives appears to have altered the character of malware authors and the malware itself. The newcomers are professionals in the traditional sense of the word; that is, they are paid to do what they do. They bring a level of sophistication, organization, and process to malware development that has traditionally been observed only in legitimate commercial and open-source software development. This shift is most visible in the relatively new class of malware robots, called *bots*.

By Michael Davis

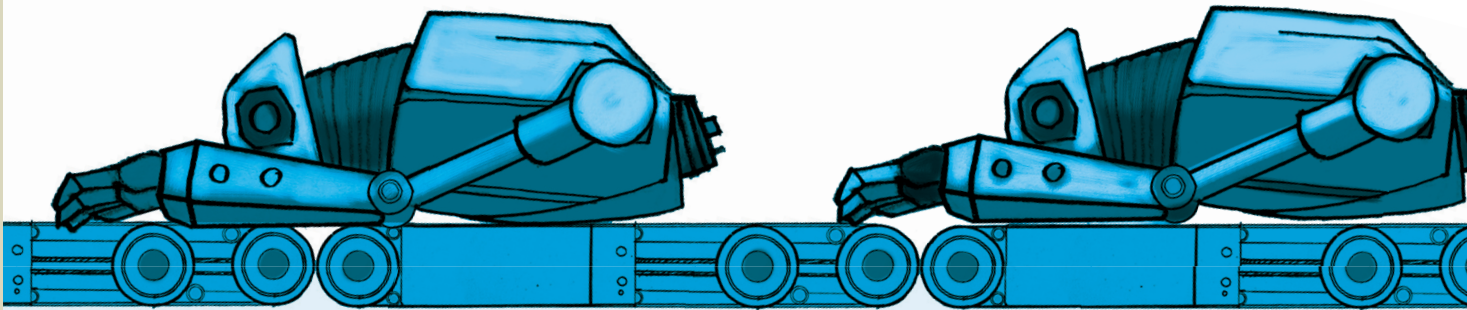


These bots are autonomous agents that crawl through the Internet, searching for vulnerable devices that they can infect with their malicious payloads.

When a target machine is identified, the bot either tricks its user with social engineering or exploits vulnerabilities in the platform to gain root-level privileges. Once that is accomplished, the bot is free to start processes and save its payload onto the system, thus turning the machine into another drone for its master, the *bot herder*. These bot-herding professionals, fueled by monetary incentives, deploy their botnets across the Internet, sometimes

creating networks of more than 100,000 *owned nodes*, or drones.

Bot herders then use these nodes to perform various nefarious acts, such as performing Distributed Denial of Service (DDoS) attacks, creating spam gateways, carrying out advertising fraud, and engaging in other money-making schemes. Though many articles and papers have been written on the technical capabilities of bots—how they propagate, communicate, and operate—very little is widely known of their development processes, such as their release cycles, contributor coordination, bug fixing, testing, and coding practices, and how those processes support the money-making operations that enable the entire venture.



So what makes open-source development different? Here are a few keys:

- ▶ Project resources are all donated
- ▶ Features are specified and decided by the same people writing the code
- ▶ Architecture or system-level designs are rarely created before a project is started
- ▶ Multiple contributors donate to the code base. Some open-source projects have hundreds of code contributors with many working in the same area of the code
- ▶ Contributors choose the features or bugs they want to fix. No work is assigned by a manager
- ▶ No direct roles are assigned to contributors. No one is necessarily dedicated to quality assurance or a certain area of the code base
- ▶ No project plan, milestones, or deliverables are set. Releases are ad hoc and normally initiated by new features and bug fixes

Open-Source Development

Open-source development is a variation on the standard professional software development model. Because the source code in open-source projects is, by definition, open, there are some notable differences, including the number and type of project contributors, how they interact, and what tools and methodologies they use. Yet the output of open-source projects is as reliable, if not more so, than what is produced by traditional, closed-source, commercial software organizations.

Furthermore, because most open-source development occurs over the Internet with contributors distributed across

Though different from standard commercial software development, open source has proven itself to be a robust and sustainable development model by building some of the most popular software in the world, including the Apache Web server and the Firefox Web browser. According to *www.netcraft.com*, Apache runs more than 60 percent¹ of the world's Web servers, while Firefox is steadily increasing its share of the browser market and is now the second most used browser in the world.²

Bot Development Life Cycle

There are four main bot families: Agobot (a.k.a. Gaobot), Sdbot, GT-

Bot authors' adoption of open-source development methodologies has no doubt led to the production of more reliable and robust bots.

the world, they often rely on asynchronous communication channels—such as email and informal documentation—to maintain the status of the code base. Most open-source development projects use rudimentary change management, version control, and bug-tracking systems. These systems are made publicly accessible so that users can check on a project's current status without bothering contributors. This basic development infrastructure enables contributors to release updates quickly—without worrying about customer support.

Bot, and Spybot (Spybot, though it is really an Sdbot variant, has enough variants to warrant its own family). Each family contains both functional variants and bug-fixing re-releases. Most variants include small changes to increase bot stealth, but sometimes variants contain new functionality, such as a new exploit vector. For example, Spybot 1.3 can now propagate via NetBIOS, which was not possible in previous versions.

1. http://news.netcraft.com/archives/2006/05/09/may_2006_web_server_survey.html

2. http://en.wikipedia.org/wiki/Usage_share_of_web_browsers

The bot development life cycle is quite different from that of previous malware such as viruses and Trojans, which mostly used a write-and-quickly-release methodology. Bot development, on the other hand, appears to follow a very similar process to open-source development. Multiple contributors help build the product; developers pick the feature they want to work on; module reuse is essential; version control is enforced; testing is done by developers; and releases are driven by bug fixes. Bot authors' adoption of open-source development methodologies has no doubt led to the production of more reliable and robust bots. In the following sections I will compare and contrast in more detail the botnet development process with the standard open-source development methodology for each key critical area.

Multiple Contributors

A virus or Trojan is usually written by a single author who has complete control of the features and timing of release updates. Bots, however, are different; most bots are written by multiple authors.

Looking at RBot, an Sdbot descendant, we can see that five developers worked on the code base. We can infer from the many comments made by various authors that they used the source code as a medium to communicate with one another, potentially displacing the use of IRC, email, or even telephone. In Figure 1 below, the comment after this integer initialization is an example of their dialogue.

Other comments show that the authors frequently used source code to ask questions and even argue with each other about the proper way to implement a solution.

After analyzing bot source code for Spybot and Agobot, it appears that the first version of a bot is usually written by a single author with testing help from friends. After the first version is released, other contributors join the effort and help develop new features, do more testing, and fix bugs. For example, both Spybot and Agobot contain a file that lists the contributors and what fixes or features they helped to develop. Figure 2 contains an excerpt from Agobot 0.2.1-pre2 contrib.txt file:

```
int current_version=0; //Nils wtf is this?
```

Figure 1: Example of developer conversations in the RBot source code

```
Contributions to Agobot3:
Num - Name -What
1. - Ago -Writing Agobot3 base, being the
author/maintainer
2. - Fight -Hosting my testing bots
3. - killer77 -Donating money to make Agobot3 as good as
it is today
4. - dj-fu -Helped me finding bugs
5. - Chrono -Hosting me a site and helping find bugs
6. - harr0 -Hosting me a site
7. - ryan1918 -Hosting me a site or forum too (not yet)
8. - PhaTTY -Implementing new features into Agobot3
```

Figure 2: Excerpt from pre2.contrib.txt file detailing Agobot's contributors

Clearly, many contributors directly participated in the development of this release: dj-fu worked to fix bugs within the bot, and other contributors helped the project by hosting a site instead of writing code.

This model is similar to that of open-source projects, which usually start with one programmer producing an application that others feel would be better with a few more features. More programmers join the project and help test and develop features. Eventually, others join in who are not programmers but help the project by donating money, resources, and other expertise.

Feature Modifications

The ability to work on what you want, when you want to—without worrying about a release date—is one of the main motivations for open-source developers. They are not required to work on features that improve stability or robustness if they do not want to.

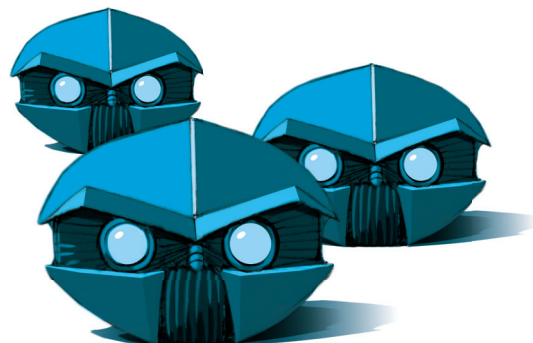
This pick-and-choose methodology also fuels the innovative development seen in bots and, consequently, determines the features that are added or modified. Compared with other malware, bots have evolved to incorporate far more advanced and stealthy methods of command and control, in addition to using multiple exploit vectors, advanced propagation techniques, and advanced programming constructs (polymorphism, object-oriented programming, etc.). However, not all bot development comes solely from developers' desires; much of it comes from the changing environment, or arms race, in which all malware and anti-malware authors are engaged.

Money is a second reason that certain features make it into a release and others do not. A feature may generate revenue either by fraud, such as click fraud, or because the author was paid to develop it. This method is similar to open-source sponsorship, in which a commercial entity pays for developers to work on a project. Naturally, the commercial entity will have its developers work on features that will not only help the general public but also itself. A great example is IBM's involvement with Linux. IBM has contributed large amounts of code to the Linux kernel, including a journal-based file system. IBM has benefited by gaining a more stable version of Linux for the PC and mainframe servers it sells. Furthermore, by contributing, IBM can offer consulting services based upon the Linux expertise its developers gained.³ Bot authors employ the same plan, which is called the patronage strategy. The arrangement allows bot developers to create a market for themselves in which they can be paid to customize the software.

Module Reuse

The Mythical Man Month, by Fredrick P. Brooks, is widely considered one of the best texts on software development. In it, Brooks says that within software

3. <http://www.opensource.org/docs/products.html> <http://management.itmanagersjournal.com/management/04/05/10/2052216.shtml?tid=85>



development there is no silver bullet; there is no one method that produces reliable, robust, and efficient code the first time. However, Brooks states that there is a *brass* bullet—module reuse—which can greatly help software development projects. By creating modules and reusing them in different projects, developers decrease testing and integration time as well as the number of bugs.

The premise of open source implies modular development and reuse of code. Reusing modular code, however, was not common in malware until the introduction of bots. Agobot, Version 3.0.2.1, for example, uses multiple open-source modules such as openssl, pthreads, and adns. Using each of these modules decreased the development time by implementing a specific feature (e.g., encryption) while simultaneously gaining cross-platform support. Furthermore, because openssl and pthreads are used in other projects, the likelihood of running into bugs that would need to be fixed when integrating these modules drops considerably. Even when there are bugs, other users of the modules will likely have documented the problems on various community forums and message boards.

A more recent example of module reuse was the release of the first Windows® Kernel Mode IRC Bot in April 2006.⁴ This bot, though only a proof of concept, would not have been developed as quickly without the pre-existing kernel-level network sockets code released on *www.rootkit.com*. This public code allowed the author to easily and quickly recreate the functions for interoperating with the IRC protocol from a Windows kernel driver without specialized knowledge of the Windows kernel.

Beyond integrating pre-existing code, the author of the Kernel Mode IRC Bot wrote the code in a way that permitted user testing. This simple functional enhancement of making the kernel portion of a bot an easy-to-debug module decreased the time that other bot authors would need to adapt their existing code base to the kernel-mode bot.

Version Control

Version control systems are very important to any development project, open or proprietary source, but they are particularly vital to open-source projects because the contributors are usually geographically distributed. Version control tools let developers revert to previous builds if the latest one is rendered unusable by new, poorly written contributions. Furthermore, version control systems help contributors track what other contributors have written into the code and how a feature or bug was fixed. Lastly, version control systems enable new users, such as potential new contributors, to learn about the project, its internals, and its evolution without working directly with other developers.

As bot development projects grew in size and scope, the contributors decided to adopt formal version control systems to track their source code. For example, the first bots had one source file, but recent generations of bots, such as Agobot, contain hundreds of source files. In the case of Agobot, certain folders and default files associated with CVS (Concurrent Versions System) can be found in the source code while Phatbot, an Agobot descendant, appears to be tracked with Subversion.

4. http://tibbar.blog.co.uk/2006/04/06/kernel_mode_IRCbot-708256

Testing

Most security professionals believe that the majority of malware authors do not test their software to ensure that it functions as intended prior to a release. Perhaps this belief was true when most malware was produced by “script kiddies” who generally lacked the programming skills and expertise necessary to develop high-quality software. But this belief is no longer valid because more and more malware authors are profiting from their creations. As financial incentives come to dominate, malware authors will be compelled to test their products for efficacy before releasing them.

Open-source testing differs from commercial development in that code writers are also the testers. Lack of a formal quality-assurance (QA) group results in many bugs being found by users of the software instead of a QA team prior to release. Although many open-source software packages compete head-to-head with commercial products, such as the Apache Web server with Microsoft® Internet Information Services (IIS), most open-source software is not known for its quality. Developers quickly release bug fixes and finished features so that they can move on to develop the next interesting feature.

Bots follow this same quick test-and-release process. For example, when

a new feature was added to Agobot that resulted in a bug, the developer responsible for the feature almost immediately diagnosed the bug, fixed it, tested the fix, and generated a new release (see Figure 3).

Release Intervals and Bug Fixing

Bots have a simple release cycle. New versions appear when a major bug is fixed or the author adds a “cool” feature, such as a new exploit vector. As with open-source software, bot releases are *interrupt driven*. They happen only when needed or when a developer feels like working on the software. Open-source software rarely has product-development documents, such as timelines and system designs. Similarly, bot development also lacks timelines, release schedules, and design documents.

Viruses and Trojans can have many variants, but most variants are released with only simple changes—typically to avoid detection by anti-virus software and other security mechanisms. The Mytob virus illustrates this point. A new variant with only a single change to avoid detection was released almost every day in July 2005.⁵ There were neither new features nor additional bug fixes in these variants. Even though

5. <http://www.virusbtn.com/virusbulletin/archive/2005/09/vb200509-new-malware-distribution-methods>

```
0.2.1-pre4-fix1:
-----

-for users:-
1. Fix for executing commands without login - Ago
   - Sorry I didn't notice this, I added an internal
     message path for handling topic commands without
     login, but due to debugging code left in the code
     every message was handled that way :)
```

Figure 3: Bug fixing comment from Agobot release note

they were released rapidly, the Mytob variants were not versioned as bots are. Bot authors, because of the very active bug fixing, need an efficient way to tell their customers—bot authors and bot herders—that a new version is available. Almost all public software that requires bug tracking uses versioning. The majority of bots that have public source code provide the same type of versioning as production-grade software: listing a version and patch level in the change log or source code, or by executing a version command, such as !ver, within the bot software.

When a new release of a bot becomes available and bot herders want to upgrade, they simply use their command-and-control framework to update their drone army in the field. Introducing bug fixing within bots forced bot authors to build-in upgrade functionality. This auto-upgrade feature is evident in both Spybot and Agobot, as well as in many well-known open-source applications, such as Mozilla's Firefox browser.

Open-source projects are distinct from proprietary development projects in that the users are often the developers themselves. This overlap between user and developer creates a powerful incentive to fix bugs, particularly ones that are visible and irritating to users. Bots share this characteristic with open source in that the bot developers use the bot software to build their own herds of drones. Thus, they are motivated to fix any bugs quickly that impede their herd building.

Most other malware, such as viruses and Trojans, are generally released with bugs or architectural defects that cause them to fail. For example, the IP address-generation routine in the SQL Slammer worm prevented it from infecting a large portion of the Internet because the routine was entirely random and gave no preference to local subnets of reachable, populated addresses.⁶

Bot development efforts even track changes and authors with formal change logs, typically found inside the source code. For example, both Agobot and Sdbot contain change logs within the source code packages that list what bugs were fixed and by whom. In other bots, the changes are sometimes credited to the person who reported and fixed them. See Figure 4 for an Sdbot change log example.

Change logs are a standard feature in commercial product development. In many open-source applications, the change logs are simple text files. Knowing when a bug was fixed helps other developers, such as botnet authors building variants, decide if and when they should merge their code bases. More importantly, change logs tell bot herders when to recompile and update their bots and botnets. This process is very similar to customers of legitimate commercial software checking for a new software update.

6. http://vil.nai.com/vil/content/v_99992.htm

```
changes since last release
-----
fixed 3 letter nick bug in spy
fixed c_privmsg and c_action
fixed clone acting like spy bug
fixed random nick generator (now includes the letter 'z')
fixed login/logout issues with private messages
```

Figure 4: Excerpt from the readme file for Sdbot Version 0.5b

Conclusion

Botnets are a new type of threat that should be managed differently than past threats, such as viruses and Trojans. Aside from being controlled by human bot herders, bot malware is being developed with methodologies similar to those used in open-source software development. The use of a professional development methodology represents a critical change in malware evolution.

Bots will continue to push the malware engineering envelope. The methods described here—multiple contributors, active bug fixing, versioned releases, and module reuse—are what make open-source products reliable, robust, and successful. Because these same

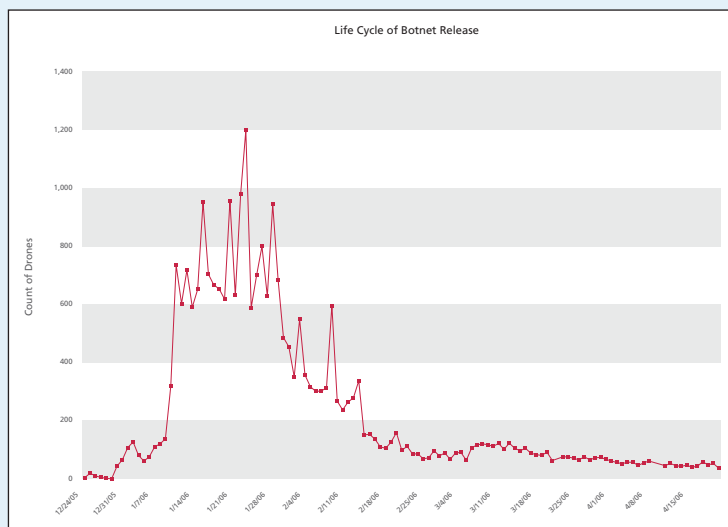
methods are used in the development efforts of the largest bot families, we can predict that the same quality, robustness, and specific features will make bots technically superior to all other types of malware the industry has seen thus far. As bots improve in reliability and robustness, bot herders will be able to demonstrate to their customers a solid return on investment (ROI). Offering customers a guaranteed ROI will cause the bot and overall malware market to grow explosively within the next few years. [Sage](#)

About the author

Michael Davis is a Research Scientist at McAfee Avert Labs. He is an active developer and installer of intrusion detection systems, contributes to the Snort Intrusion Detection System, and is a founding member of the HoneyNet Project.

Evolution of a botnet

McAfee® Avert® Labs discovered a control Web site that contained data about a particular botnet release. The graph below shows the number of unique compromised computers on each day from December 25, 2005, to April 21, 2006.



Source: McAfee Avert Labs

The first bot in this network appeared on Christmas Day, 2005. For about two weeks, there were very few infections scattered all over the world. On January 9, 2006, the botnet rapidly expanded, with most infections occurring in the U.S., Italy, Russia, and Brazil. By mid-January, there were approximately 1,000 active compromised systems, with the U.S. hosting the most (about 35 percent) and Italy in second place (about 9 percent).

In the last week of January, the botnet began to disintegrate. Most infections occurred between January 10 and February 10, so the effective lifetime of this botnet release was about one month.

Note that the curve has a long tail, which implies that some computers were never cleaned. As might be expected, the countries that had the most infections (U.S. and Italy) also had the most computers that were not cleaned.

This botnet release helped distribute adware and could push advertisements to users' machines, which might be one of the reasons why its lifetime was only one month. A bot that displays ads is much easier to spot than a bot that furtively participates in a DDoS attack or pay-per-click scheme.

This botnet release demonstrates a typical pattern in malware deployment—a fairly rapid initial propagation followed by a period of malicious activity and then, eventually, subsidence. Presumably, when returns from a deployment stop satisfying bot herders, they generate yet another bot release to circumvent security measures that blocked the last release, enabling them to sustain their overall numbers.

Is Open Source Really so Open?

By Jimmy Kuo

Advocates of open source evangelize its merits with the fervor of the converted. What could be wrong with a movement that promises collegial cooperation, collective improvements, and freedom from proprietary restrictions? Don't many hands make quick work in software development?

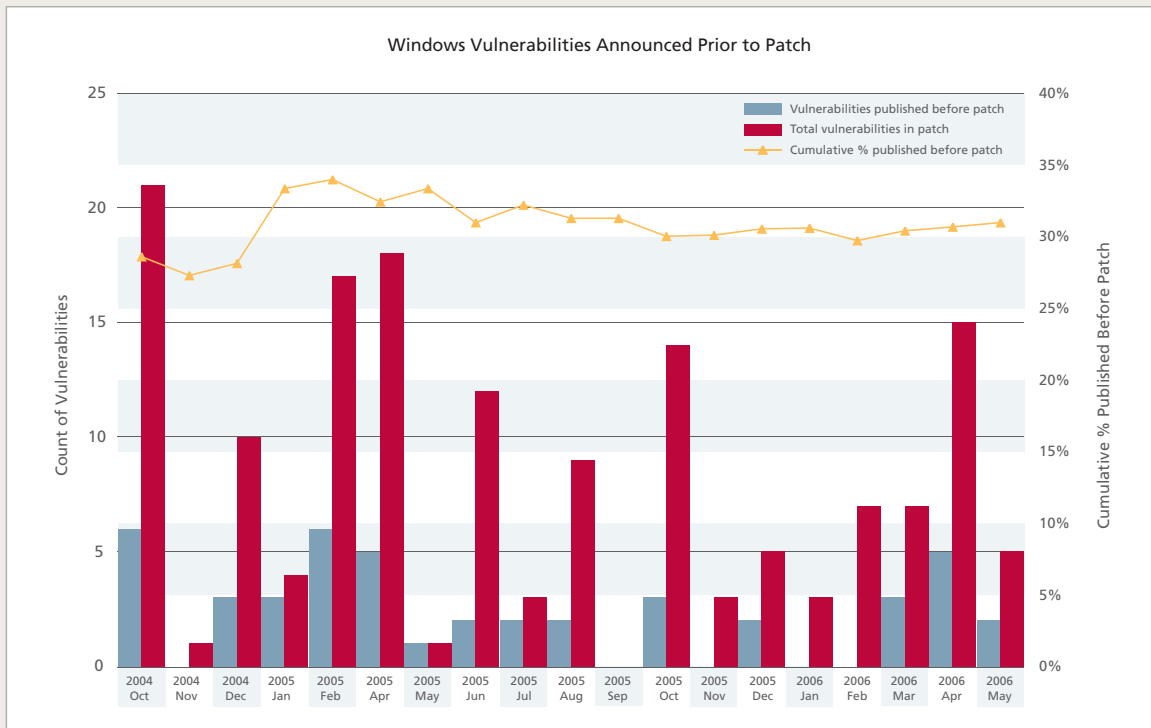
They do indeed, but there's a big drawback to a system that's always open, all the time. There are times when the community needs to show some restraint, some public responsibility, before sharing with the world. When it comes to disclosing security vulnerabilities, for example, it is not hard to see the advantage of allowing the developer time to patch holes before making a public announcement. Doing so places the greater good ahead of an individual seeking fame for being the first to spot a threat. Granted, for some people, it's asking a lot to hold back—and pass up a chance for self-promotion—while letting others quietly save the day.

Ultimately, open source's effectiveness depends on people. Some support the concept, others support the alternative. We cannot say that either is right or wrong; both have pros and cons. However, it is a fallacy to assume that all members of the open-source community insist on immediate publication of all vulnerability research results.

Share and Share Alike

The open-source position—for better or worse—is that research should be shared throughout the community. How people use and build upon the results shows the extent to which they demonstrate their responsibility to that community.

The Organization for Internet Safety (OIS; www.oisafety.org) was founded on the idea that “standardized, widely accepted processes will allow security vulnerabilities to be handled in a way that reduces the dangers they pose and will help security vendors and researchers to more effectively protect Internet users and critical infrastructures.” In September 2004, OIS released Version 2.0 of its Guidelines for Security Vulnerability Reporting and Response Process, providing researchers and organizations with a method to follow if they believe it is better not to publicly divulge vulnerabilities until a patch is available. This gives developers time to produce a fix and requires members to keep their discoveries secret until the fix is available. Those who do not support responsible disclosure guidelines, such as those from OIS, will sometimes post their discoveries immediately and share their research.



Source: McAfee Avert Labs

Microsoft vulnerabilities publicized prior to patch release

Let's examine the popularity of these two positions.

Which Side of the Fence Are You On?

Microsoft® has a well-documented history of reported Windows® vulnerabilities for which it consistently provides patches on a mostly consistent basis. Here's where we can determine how many believe in responsible disclosure and how many believe in immediate publication. I studied the count of Windows vulnerabilities and how many were publicized prior to the release of the appropriate patch. The line graph shows the cumulative percentage of security

researchers who believe their research—whether positive or negative—should be released immediately to all.

The bar chart shows the number of vulnerabilities for which Microsoft released patches each month (red bars) and the number of those made public prior to the patch release by the discovering researchers (blue bars).

Here we have results of actual behavior, not an opinion poll, and we see that only 30 percent of researchers practice unrestrained open-source behavior, which means approximately 70 percent act more cautiously.

Though many researchers voice support for open source, more than two-thirds of them are willing to keep a secret and let developers prepare their fixes when it comes to Windows vulnerability disclosures. Clearly, belief in open source does not equate to engaging in irresponsible disclosure.



About the author

Jimmy Kuo is a Senior Fellow of McAfee® Avert® Labs. He speaks internationally at security conferences and universities. Jimmy is the technical representative for McAfee at government and industry consortia.

Vulnerability Bounties

Do they really protect customers?

By Stuart McClure

There are few controversies as volatile in the security world as vulnerability discovery. From discovering and reporting the vulnerability to paying researchers to find specific flaws in popular software—the gamut of discovery, acknowledgement, fixing, and crediting runs amok with ego, publicity, and negative financial impact.

The stakeholders in this debate include the customer (the one who is supposed to benefit most from all of this hubbub and who wants only to be protected), the vendor (who just wants to sell more products), and the researcher (who often has multiple motives: protecting people, fueling his or her own ego, and, now, making money). As you would imagine, each of these parties has needs that are not always aligned.

The customer wants no security holes at all; but if vulnerabilities do exist, then customers want to know about them as quickly as possible so that they can either apply a patch or mitigate the problem.

Most vendors are motivated to protect their customers (though it wasn't always that way), but their side motivation is to spend their scarce resources building and selling products, not patching them up. They are also anxious to avoid the negative publicity that inevitably surrounds vulnerability reports.

Finally, the researcher wants to protect the customer by finding vulnerabilities, but researchers are driven primarily by ego. Most want the vendor to acknowledge the vulnerability, so that they can claim credit for their accomplishment (the bragging rights). And now we have an added incentive for these researchers: cold, hard cash.

Two Views of Security

Vulnerability discovery is complicated by two competing views of best practice: what we call security by obscurity and full disclosure. In the first case, we keep information close to the vest until the vendor can offer a fix. In the second, we assume that the bad guys already know this stuff, so we need the good guys (administrators) to know about these weaknesses as soon as possible—before the bad guys attack.

The anti-virus and, largely, the government and military worlds have fallen on the security by obscurity side of the fence. These fields have a long-standing maxim to trust no one when it comes to virus code and samples. The logic has always been that the fewer people who have this stuff (viruses), the less likely the problem will grow. So vendors have long held their research close to their chests in an attempt to restrict accessibility and, in theory, reduce the potential exposure of that threat in the wild. As a result, the groups of vendors who do share their collections and samples are often closed to new members. In many cases, the anti-virus researchers' paranoia has been justified because malefactors have tried numerous times to infiltrate this closed world. The upside of this model is obvious: the fewer people who have guns, the fewer bullets that will be used. The downside is that this knowledge is kept to only a select few and therefore cannot be disseminated to help the masses.

The full-disclosure movement comes from the pioneers of whitehat hacking who exposed the techniques and tools used by the bad guys in an effort to educate the masses before they became victims. The argument for immediate

full disclosure is that it gets the information out to the people who need it most and motivates vendors to react quickly and provide patches for the problem. The downside is that it can put the masses it seeks to protect in a compromising position if the bad guys have a quicker turnaround than the vendors.

Commercializing Discoveries

Once solely the domain of benevolent researchers, vulnerability discovery and disclosure has now taken on a financial motive. The security companies that have jumped on the vulnerability discovery pay-to-play bandwagon include VeriSign's "Vulnerability Contributor Program" (<http://www.idefense.com/methodology/vulnerability/vcp.php>), which was developed by VeriSign subsidiary iDefense; 3Com's TippingPoint "Zero Day Initiative" (<http://www.zerodayinitiative.com/>); and Mozilla's "Mozilla Security Bug Bounty Program" (<http://www.mozilla.org/security/bug-bounty.html>).

Although the intended outcome of Mozilla's program appears to be straightforward—to find bugs in their own software before bad people do—both iDefense's and TippingPoint's programs may produce ethically conflicted consequences. If companies provide a cash reward for bugs found in their own software, that's a good thing. After all, if a researcher has invested his or her time finding a bug, it's fitting for the benefiting vendor to pay for the work. But when security companies pay for finding bugs in other vendors' software, the results may not be so beneficent. By using the research of others to publicize vulnerabilities, for example, these companies may sell more subscriptions to their threat intelligence services and gain publicity from it—in other words, they will make money.

From the customer's perspective, the disadvantages of such a vulnerability discovery program are many. The more vulnerabilities that are found, the more you must fix to protect yourself; and the more you must fix, the fewer you inevitably will. Further, the more people involved with a particular finding, the more likely that information about the vulnerability will leak out. And a leak means that someone can build a worm that will affect customers before they are patched or prepared. The last point strongly undermines the expressed goal of the program: to protect people.

Want more bad news? We've read that some of the researchers who report these vulnerabilities to commercial programs, such as iDefense, are often associated with real

hacking events. For example, look at a Distributed Denial of Service (DDoS) attack that occurred last year: "Security Web Sites Taken Down by Unhappy Hackers" (<http://www.techworld.com/security/news/index.cfm?NewsID=3465>).

In this case, a Web site was hit by a group of hackers who reportedly were unhappy when Web site members criticized the group and a particular researcher, ATmaCA. What followed, according to the article, was a series of extortion emails from the associated hackers to the Web site owners. In the debacle that followed, ATmaCA was ultimately tied, either directly or indirectly, to the attack.


Fast-forward to May 11, 2006, when the "Zero Day Initiative" released an advisory with credit to ATmaCA (<http://www.zerodayinitiative.com/advisories/ZDI-06-015.html>). Was this ATmaCA the same person who sparked and perhaps launched the DDoS attacks on the Web sites in 2005? If these companies are paying known hackers—and ultimately funding real hacking efforts against legitimate companies, perhaps these programs warrant further reflection.

Nonetheless, there are advantages to proactive vulnerability discovery, regardless of the sponsor. We know that there are countless undiscovered vulnerabilities in all products made. Whom would you rather find these vulnerabilities: malicious profiteer hackers or good guys who work with the vendor to get them patched?

Between Extremes

Clearly, full disclosure, and its related incentives, can cause problems. But security by obscurity may not always be the best practice. Perhaps somewhere between these extremes is the right position, but finding that position is difficult.

If an organization offers payment to motivate individuals to report their findings and uses that information to improve its own products, then who can blame them? Or if a vendor discovers vulnerabilities as part of its everyday fight against threats and wants to incent its team members to report their findings, then such a program benefits everyone. But if payment programs simply fill the coffers of malicious hackers who look hard for more and more vulnerabilities, then vendors, customers, and legitimate researchers are all hurt. In the first case, vulnerability disclosure means everyone wins; but in the second case, we all lose.

What's your view on vulnerability disclosure? Let us know at Sage-feedback@McAfee.com. 



About the author

Stuart McClure is the Senior Vice President of Global Research and Threats at McAfee® Avert® Labs. He is the lead author of several books, including the popular Hacking Exposed series. He has been a security professional since 1989.



Will the Worm Eat the Apple?

By François Paget

Until recently, the Apple Macintosh OS X operating system stood as something of an anomaly. With a unique blend of proprietary and open-source technologies, Apple sought to create a modern operating system. By leveraging the advantages of both development approaches, Apple aimed to build a more secure and robust platform than either of its two main rivals, wholly proprietary Windows® and wholly open-source Linux. Did Apple get the recipe right? Was Apple's claim to better security justified? For the first few years, the answer seemed to be yes. Few vulnerabilities were discovered in OS X, and Apple addressed them quickly and with little fanfare.

As 2006 began, the situation changed dramatically for Apple and OS X. In early February, someone with the pseudonym r3d3pshun posted a file several times on an underground Internet forum. The clearly suspicious file sometimes appeared under the guise of a photo of Britney Spears' baby. When this effort failed to provide any notoriety for its author, r3d3pshun posted the same file, but this time labeled it as screenshots of Leopard Version 10.5, the next release of the Mac OS X operating system. This attempt worked; three days

later, the code was named OSX/Leap. Designed to propagate through the AIM/iChat instant messaging system, it became the first known virus to attack the Mac OS X platform. This episode was the first in what was to be an eventful February.

A short time after OSX/Leap, OSX/Inqtana.A and its variants appeared. Arriving a few days too late, the authors did not get the publicity that they likely hoped for. This virus family exploits an old vulnerability in the Mac OS X 10.3.9 service that handles directory traversal in Bluetooth file and object exchanges, allowing remote attackers to read arbitrary files.

During this unprecedented week, two more vulnerabilities appeared, one in Apple's Safari Web browser and the other in Apple's Mail application. Both vulnerabilities were linked to the ability of these applications to run certain scripts without asking for permission. For Safari, the vulnerability affects ZIP archives; for Mail, it concerns the AppleDouble MIME format. When exploited, these vulnerabilities enable any application to run, theoretically enabling a remote user to control a machine.

When these two extremely critical weaknesses were announced, Apple quickly set to work. On March 3, the company distributed an initial series of patches. Ten days later, a second series of patches was distributed. In less than two weeks, Apple had fixed more than two dozen weaknesses.

Vulnerabilities: The Real Threat

Vulnerabilities are not the same as viruses. Vulnerabilities alone are benign, but they are weaknesses that could open the door to malicious actions. A recent tally by McAfee® Avert® Labs demonstrated that vulnerabilities in the Apple environment today are being discovered at a markedly increased rate over that in prior years. All reporting organizations—NIST, FrSIRT, or Secunia—are seeing a sharp increase. Two of these organizations report an increase exceeding 200 percent between 2003 and 2005.


However, the number of new viruses in OS X has remained almost zero since January 2004, and the bugs and limitations encountered in the various versions of OSX/Leap and OSX/Inqtana.A ensure that they will not propagate. Though OS X users are not immediately at risk from known malware, the danger has not yet passed. The source code for these viruses is available on the Internet and is likely being studied and optimized for future release. A blog site offering commented code of OSX/Leap saw more than 8,500 downloads since it became available. It is almost certain that one day the source code will serve as the basis for another, perhaps more effective attack.

Some specialists now claim the Mac OS X platform might be more fragile than Windows XP. In his ZDNet blog, George Ou recently compared Secunia's assessments. For the three highest threat levels (extreme, high, and moderate), he classified the vulnerabilities associated with the two operating systems for the months from February 2004 to February 2006. For the 25-month period, Secunia counted 238 vulnerabilities in Mac OS X, compared with just 95 vulnerabilities in Windows XP.

No End in Sight

Between January 1 and May 11 of 2006, Apple corrected more than 80 vulnerabilities. In April 2006, two days after Apple released its Security Update 2006-002, independent researcher Tom

Ferris announced seven previously non-public flaws. Ferris also released proof-of-concept code on the Internet. He said he had sent his research to Apple Support at the start of 2006 and was assured that they would "be fixed in the next security release." But as of April, only one weakness had been patched, and the fix was done with a silent update in the transition to Version 10.4.6. The other six—judged "highly critical" by Secunia—affect all Mac OS X versions, including the most recent. On May 11, 2006, Ferris thanked Apple for the new Security Update 2006-003, which fixed almost all of the issues he had reported previously. Four days later, however, he moderated his tone and message. Apple's "fixes" apparently did not address the root cause of the flaws. Ferris easily re-exposed them with slight modifications of his original exploit files. On his blog, Ferris also offers hints of several other image-file-related vulnerabilities that he is reporting to Apple.

Despite these developments, OS X still faces significantly fewer known malware threats than Windows. Many would argue that this is more a result of smaller market share than of any architectural security choices or use of open and proprietary source code. Regardless, Apple faces a formidable foe in the open-source threat community, and that community's propensity for combining tools and code in blended threats may one day result in OS X exploit code being added to an existing piece of malware to enhance its virulence. 

About the author

François Paget is a Senior Research Scientist at McAfee Avert Labs. He has been involved in virus research since 1990. François is a regular conference speaker at French and international security events, author of numerous articles and a book, and General Secretary of the French Information Security Club (CLUSIF).

1. Apple's Macintosh OS X is based upon Free BSD and incorporates a number of well-known open-source technologies, such as Samba, Apache, and MySQL.
2. http://vil.nai.com/vil/content/v_138578.htm
3. http://vil.nai.com/vil/content/v_138608.htm
4. For more information about the vulnerability (CVE-2005-1333), see <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1333>
5. <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0848>
6. Vulnerability ratings were given by Secunia, <http://secunia.com/advisories/19064/>
7. Security pack 2006-001: <http://docs.info.apple.com/article.html?artnum=303382> and Security pack 2006-002: <http://docs.info.apple.com/article.html?artnum=303453>
8. National Vulnerability Database: <http://nvd.nist.gov/statistics.cfm> French Security Incident Response Team: <http://www.frSIRT.com/search.php> Secunia: <http://secunia.com/vendor/>
9. Vulnerability statistics for Mac and Windows: <http://blogs.zdnet.com/Ou/?p=165>
10. <http://docs.info.apple.com/article.html?artnum=61798>
11. Mac OS X Multiple Potential Vulnerabilities: <http://secunia.com/advisories/19686/>
12. <http://www.security-protocols.com/>

McAfee, Inc. 3965 Freedom Circle, Santa Clara, CA 95054, 888.847.8766, www.mcafee.com

McAfee and/or additional marks herein are registered trademarks or trademarks of McAfee, Inc. and/or its affiliates in the US and/or other countries. McAfee Red in connection with security is distinctive of McAfee brand products. All other registered and unregistered trademarks herein are the sole property of their respective owners. © 2006 McAfee, Inc. All rights reserved. 11-avert-sage-rpt-0706