Chinese MALWARE Factory 24
URL Shorteners Made My Day! 68
Cover Story Using Kojonet Open Source Low Interaction Honeypot 4
Dear Reader,

Welcome to Issue 003 of the HITB Magazine!

We're really super excited about the release of this issue as it coincides with our first ever HITB security conference in Europe - HITBSecConf2010 - Amsterdam!

The design team has come up with (what we feel) is an even better and more refined layout and our magazine now has its own site! You'll now find all the past and current issues of the magazine for download at http://magazine.hitb.org or http://magazine.hackinthebox.org.

Also in conjunction with our first European event, we have lined up an interview with Dutch master lock picker and founder of The Open Organization of Lock Pickers (TOOOL) Barry Wels.

We hope you enjoy the issue and do stay tuned for Issue 004 which we'll be releasing in October at HITBSecConf2010 - Malaysia. In addition to the electronic release, we're hoping to have a very 'limited edition' print issue exclusively for attendees of HITBSecConf2010 - Malaysia!

Enjoy the summer and see you in October!

Dhillon Andrew Kannabhiran
Editorial Advisor
dhillon@hackinthebox.org

---

TEHTRI-Security is an innovative French company specialising in Security Consultancy and Expertise for Information and Communication Technology.

New Amazing Training
"HUNTING WEB ATTACKERS"
Come and join us during HITB Malaysia 2010

---

Technical consultancy
- Penetration Tests and Security Audits
- Incident Management, Monitoring...

Advanced consultancy
- Assistance for countries, companies...
- Fight against information leaks, Protection...

We do know, understand and master the techniques and the methods of attackers (hackers, business intelligence, computer warfare, etc...) as well as the resources needed to counter the current threats.

---

Contents

INFORMATION SECURITY COVER STORY
Using Kojonet Open Source Low Interaction Honeypot 4

A Brief Overview on Satellite Hacking 16

Malware Analysis
Chinese Malware Factory 24

Windows Security
Reserve Objects in Windows 7 34

Application Security
Javascript Exploits with Forced Timeouts 42

Non-Invasive Invasion
Making the Process Come to You 48

IAT and VMT Hooking Techniques 62

Web Security
URL Shorteners Made My Day! 68

Book Review
ModSecurity Handbook 76

Interview
Barry Wels 78
Using Kojonet Open Source Low Interaction Honeypot to Develop Defensive Strategies and Fingerprint Post Compromise Attacker Behavior

By Justin C. Klein Keane, justin@madirish.net

In attempting to defend against intruders and protect assets using defense in depth principle it is critical to not only understand attacker motivations, but also to be able to identify post-compromise behavior. Utilizing data that identifies attacker trends it may be possible to prevent compromises. Furthermore, information about resource usage and patterns may allow system administrators to identify anomalous activity in order to detect compromises shortly after they occur.
Honeypots can be used to monitor attacker behavior during an active compromise of a system set up for this express purpose. Although we can only guess at attacker motivation, through traffic analysis we are able to infer the types of resources that attackers consider valuable. The preponderance of log evidence of failed SSH attempts by unknown users implies that SSH servers are assets to which attackers are attempting to gain entrance.

By deploying honeypots that simulate resources we know attackers will target, namely SSH servers, we are able to catalog post compromise behavior. Because certain honeypots present inherent risks, utilizing software based, low interaction honeypots we can mitigate risk while still providing a rich target environment within which to collect data about attacker activity.

**One goal of collecting data about brute force attacks is to fingerprint post compromise behavior.**

Given the ability to access many SSH servers using simple usernames and passwords over a well understood protocol, it is unsurprising that brute force, or password guessing, attacks against SSH servers have become common. The SSH protocol is open and well documented; several developers have invested time and effort to implement SSH clients quickly and easily. Many automated attacker tools allow users to easily perform point-and-click password guessing attacks against SSH servers. Much like port scanning, SSH brute force attacks have become a part of the background noise of the Internet. Virtually any administrator running an SSH server need look no further than their SSH server logs to find evidence of password guessing attacks.

**SSH BRUTE FORCE ATTACKS**

Given the preponderance of SSH brute force attacks it is worthwhile to explore the motivations of attackers. Unfortunately, without any data, these motivations remain a mystery. In order to attempt to understand the goals of attackers, or defend against them, it becomes necessary to collect concrete data about SSH brute force attacks.

One goal of collecting data about brute force attacks is to fingerprint post compromise behavior. We assume that the goals of attackers are separate and distinct from those of regular system users. Because malicious users are attempting to utilize system resources in non-traditional ways it may be possible to spot this type of anomalous behavior. It may be impossible to identify malicious users based on usernames and passwords alone, for instance in the case that an attacker has compromised or guessed a legitimate user’s credentials. For this reason fingerprinting behavior immediately following a successful authentication becomes important. Fingerprinting is the process of identifying trends or commonalities amongst attacker behavior (consisting of system commands issued) that might distinguish it from legitimate user behavior. If it is possible to develop a signature of malicious behavior then that signature can be used to identify compromise. This process would not prevent attacks, but would suffice to alert administrators of a compromise soon after it had taken place to minimize damage and contain incidents. Such early identification is critical to containing damage caused by intrusions and forms an additional layer of defense, supporting the defense in depth principle.

**HONEYPOTS**

Honeypots were first popularized by the Honeynet Project’s Lance Spitzner’s and Lance Spitzner’s *Know Your Enemy*. A honeypot is a vulnerable, or deliberately insecurely configured system that is connected to the Internet and carefully monitored. There are many motivations for deploying a honeypot. Some honeypots are deployed to distract attackers from extremely valuable assets and to waste attacker resources on “fake” targets. This strategy is of debatable merit as there is little chance of accurately gauging the success of such a honeypot, especially if compromise of legitimate assets goes undetected. Another use of the honeypot is as a type of early warning system. If the honeypot detects to malicious traffic from an asset within the organization a compromise can be inferred. Where the honeypot returns its most value, however, is when exposed to the Internet in order to observe and analyze attack traffic and attacker behavior independent of an organization’s internal configuration.

There are a number of reasons why honeypots are difficult to deploy in this last mode. In addition to significant time requirements there is also inherent difficulty in setting up a system that is attractive to attackers. Additionally, such a system will likely invite damage by the target attackers and will require a rebuild after use. Furthermore, it is no simple task to configure an effective monitoring system that will not alert an attacker to observation.

In addition to logistical considerations, of significant concern in deploying such a honeypot on the Internet is the possibility for “downstream stream liability”. If such a system were to be compromised by attacker, and then the attacker were to use the system as a pivot point or launching pad to attack other resources there could be serious consequences. If the honeypot were used to attack third party systems then the honeypot maintainer could be culpable in facilitating a compromise. If the honeypot were used to attack internal systems then it could potentially bypass authorization rules that prohibited connections from outside hosts. Using such a pivot point whereby an attacker compromized the honeypot in order to attack other assets that might not be routable from the wider internet could create significant problems.

Furthermore, to be of any value, a honeypot must be analyzed after it is compromised. This process often can be extremely time consuming and may or may not result in valuable intelligence. Even though the advent of virtualization has significantly reduced the overhead of configuring and deploying honeypots, tools designed to significantly streamline post compromise analysis simply do not yet exist. Without adequate time and suitable analysts much of the value of honeypots is lost.

For all of these reasons honeypots should only be deployed with extreme caution and only after consultation with others within your organization to determine acceptable risk.

**Low Interaction Honeypots**

Traditional honeypots consist of full systems that are set up and configured from the hardware layer up to the application layer. Such a configuration provides a rich environment for attackers to interact with and can serve to collect data about a wide variety of vulnerabilities, attack methods, and post compromise behavior. By providing an attacker with a realistic environment you are most likely to collect useful intelligence. Honeypots of this style are known as “high interaction honeypots” because they provide the widest array of response.

High interaction honeypots have significant downsides. Careful consideration must be given to the configuration of egress rules for high interaction honeypots in order to minimize the possibility of downstream liability. Furthermore, encrypted protocols present problems when monitoring traffic to and from a high interaction honeypot. These reasons combined with the high deployment, rebuild, and maintenance overhead make high interaction honeypots unattractive to many organizations.

**Low Interaction Honeypots**

Low interaction honeypots were developed to address many of the deficiencies of traditional, high interaction honeypots. Developer libraries and can provide software that is as secure as traditional honeypots and can serve to collect many of the deficiencies of traditional, high interaction honeypots. Low interaction honeypots consist of software systems that
simulate specific aspects of complex systems. Because they are implemented in software, low interaction honeypots present significant safety improvements over high interaction honeypots. Low interaction honeypots can strictly monitor and limit both inbound and outbound traffic. Low interaction honeypots can restrict functionality and can more safely contain malicious attacker activity.

**METHODOLOGY**

For the purposes of this study, Kojoney\(^*\) written by Jose Antonio Coret, was used as a foundation. Kojoney is an open source low interaction honeypot implemented in Python. Kojoney simulates a SSH server, listening on port 22. Kojoney uses the popular OpenSSH\[^2\] and Python’s Twisted Conch\[^3\] libraries to negotiate SSH handshakes and set up connections.

Kojoney utilizes a list of usernames and passwords that can be used to access the system. This means that not all connection attempts will be successful. Once a connection has been established Kojoney presents attackers with what appears to be an interactive shell. Commands issued by attackers are interpreted by Kojoney and attackers are returned responses based on definitions from within the Kojoney package. The only system functionality available to attackers is ‘wget’ or ‘curl’ for fetching remote files. However, even this functionality available to attackers is not simulated. The most noticeable of these was the inability for an attacker to interact with packages that were downloaded. This meant that attackers could not download toolkits but they could not actually inflate compressed packages or execute binaries. Kojoney responds with a vague error message if it cannot simulate functionality. When attackers encounter this behavior it is common for their session to end. Because Kojoney does not simulate a full system once an attacker attempts complex interaction, it was common for attackers to terminate their sessions after encountering commands that do not produce desired results.

**Deficiencies**

Kojoney deliberately limits functionality. Although the installation utilized for this study was heavily modified there was certain functionality that was not simulated. The most noticeable of these was the inability for an attacker to interact with packages that were downloaded. This meant that attackers could not download toolkits but they could not actually inflate compressed packages or execute binaries. Kojoney responds with a vague error message if it cannot simulate functionality. When attackers encounter this behavior it is common for their session to end. Because Kojoney does not simulate a full system once an attacker attempts complex interaction, it was common for attackers to terminate their sessions after encountering commands that do not produce desired results.

**RESULTS**

For the purposes of this study a modified Kojoney low interaction SSH honeypot was deployed on commodity hardware and connected to the live internet with a dedicated IP address. Kojoney was configured to run on the standard SSH port 22 with a separate interface configured for management. The system was left on and running consistently over a period of roughly six months from October 27, 2009, to May 3, 2010. During this time 105,121 login attempts were observed from 596 distinct IP addresses. Of these distinct IP addresses over 70 participated in brute force attacks separated by more than 24 hour time intervals. The longest span of time between attacks from the same IP address was 135 days wherein a single IP address participated in over 6 distinct attacks.

**Most popular usernames**

Examining the popularity of certain days for attacks also provides some interesting insight. Apparently Sunday and Wednesday are the most popular days to launch SSH brute force attacks. Given the global nature of the internet and timezone differences, however, this data may not provide any real value.

**Countries**

IP addresses are assigned to internet service providers in blocks that are then subdivided to their customers. Using these assignments it is possible to locate the country to which a specific address is assigned. Examining the data for country assignments of IP addresses which participated in attacks provides some stark details.


\[^*\] Kojoney
\[^2\] OpenSSH
\[^3\] Twisted Conch

---

**Figure 1. Hours of Attack**

---

**Figure 2. Distinct IP’s by Month**

---

**Figure 3. Attacks by Weekday**

---

**Figure 4. Attacker IP by Country**

---

*China (118)*

*Romania (111)*

*US (52)*

*Korea (27)*

*Spain (25)*

*Italy (17)*

*Germany (14)*

*Brazil (14)*

*France (11)*

*Netherlands (11)*

*UK (11)*

*Macedonia (7)*

*Canada (7)*

*Russia (7)*

*Taiwan (7)*

*India (6)*
The 20 most popular passwords attempted included several common strings, as well as several based on keyboard layouts, such as ‘1q2w3e’.

Although not represented in the most common passwords, particularly interesting were passwords that seemed to have been generated using permutations of the hostname (See 100 Most Common Passwords).

Average password length

Over 133 distinct passwords utilized in login attempts were greater than 19 characters long. Of the rest, the average length of passwords attempted was 6.78.

Password resets

Although not a native feature of Kojonny, our installation included functionality to capture password reset attempts. In the sample period attackers attempted to reset passwords 42 times. Examining these records reveals interesting data. None of the password resets resulted in a password of more than 8 characters followed by a space or a carriage return. This allows us to examine the core commands (such as directory listing or file content listing) independent of their target.

The most common distinct command was ‘ls’, issued 538 times. This was followed by ‘cd’ with 338 execution attempts, then ‘wget’ with 308 attempts, ‘w’ with 303 attempts, ‘uname’ with 179 attempts, ‘cat’ with 151 attempts, ‘ps’ with 117 attempts and ‘uptime’ with 102 attempts.

Examining the full commands issued by attackers (the full line of input submitted to the honeypot) reveals a slightly different picture. Commands such as ‘ls’ and ‘cd’ became less frequent as they are almost always used with a target, while commands such as ‘w’ which generally do not include any further switches or arguments, percolated to the top of the list in terms of frequency. Looking at the list of commands it is worth noting that certain common commands with specific arguments were seen quite frequently. These include ‘uname’, ‘ls -a’ being an aggregate flag that behaves as though several other flags were utilized. The use of the ‘cat’ command to echo the contents of the virtual file ‘/proc/cpuinfo’ which contains processor identification information, also becomes quite apparent.

Downloads

282 downloads were captured by the honeypot. Interestingly the wget command was used 41 times to download Microsoft Windows XP Service Pack 3. This behavior was perhaps an attempt to test the download functionality of wget and to gauge the speed of the internet connection. Although time did not permit a full analysis of each binary downloaded the most popular download seemed to be PsyBNC, an open source Internet Relay Chat (IRC) bot program. Other popular downloads included other IRC bots, UDP ping flooder (presumably for use in denial of service attacks), port scanners, and SSH brute force tools.

Sessions

Sessions are defined as interactions where the attacker not only attempted to gain access with usernames and passwords, but...
also executed commands on the honeypot. Furthermore, sessions were delimited by time delays of more than an hour between command execution. For instance, if an attacker logged in, executed commands, and waited for more than an hour before executing additional commands then the interaction was counted as two sessions. A total of 248 attacker sessions were identified issuing a total of 3,062 commands. The average session lasted for 4.1 minutes during which the attacker issued 12 commands. The longest session lasted for an hour and 10 minutes. By far the most common command in any session was the ‘w’ command, occurring in 74% of sessions. Wget was used in over 58% of sessions as was uname. The uptime command was issued in 35% of sessions.

150 times the ‘cat’ command was used, the full command issued was ‘cat /proc/cpuinfo’, which is used to display processor information. This type of command is not typical for a normal system user.

Although some common commands observed in the Kojoney session captures could potentially be attributed to normal users, others clearly stand out. The average session is used to report on which users are logged into the system, and the ‘uptime’ command, which reports how long the system has been on, are not regularly used by non-system administrators. Similarly, the ‘uname’ command is generally utilized to determine the kernel version that is running, which could perhaps be used to search for vulnerabilities.

Monitoring command execution on systems seems like a worthwhile exercise given the results of this data. Replacing the ‘w’, ‘uptime’ or even ‘wget’ command with a binary that would log the execution of such a command before executing the intended target could provide some insight into the usage of such utilities. Using a log file monitoring system such as OSSEC, system administrators could easily keep watch over such commands to alert on suspicious behavior.

Given the sophistication of the usernames and passwords utilized by attackers a number of defensive strategies present themselves. It is interesting to note the complexity of usernames and passwords utilized by attackers. Outside of system passwords, common usernames were not necessarily attempted with common passwords. For instance, the data shows no attempts to log in using the username ‘alice’, a relatively common name that would appear at the beginning of a dictionary list of names, with the password ‘password’. From this observation, as well as the fact that the top 20 usernames attempted were system accounts, we can conclude that attackers probably do not focus their efforts on breaking into user level accounts.

Given the breakdown of username choices in brute force attacks it seems that system usernames are by far the most utilized. This is probably because system accounts are standard and the attacker doesn’t have to enumerate or guess them. The fact that root is the most common target is likely attributable to the fact that this account has the most power, but also because it appears on most Unix systems. Choosing strong passwords seems like a safe strategy for protecting the system accounts, but even more effective would be to prohibit interactive login over SSH for such accounts. By disabling SSH root login, nearly half of all brute force attacks observed would have been thwarted.

All attacker behavior was observed on the standard SSH port 22. Running SSH on an alternate port would almost certainly cut down on the number of attacks, although such a solution could confuse legitimate users and result in increased support costs. Brute force detection and prevention countermeasures, such as SSH Black15, OSSEC active response, or the use of OpenSSL’s MaxAuths feature configuration specifications could all be worthwhile. An even more effective solution would be to eliminate the use of username and password authentication altogether. Many SSH servers provide functionality for key authentication. There is additional administrative overhead in implementing key based authentication, and it is not as portable, but it is certainly more secure.

Examining the IP source of attacker behavior shows that there are certain IP blocks, that if not used by legitimate system users, could certainly be blocked to great effect. Locating and blocking specific IP ranges could dramatically cut down on the amount of SSH brute force attacks, but again could create hassle for legitimate users and requires a certain degree of administration.

There do not appear to be strong trends in the times that attackers attempt brute force attacks. Limiting the SSH server access to specific times could cut down on the number of attacks as long as administrators could be confident that legitimate users can still access SSH within time ranges. Great care would need to be taken with such a remediation, however, to prevent a nightmare scenario where a legitimate administrator or user might be unable to respond to a crisis occurring in off hours due to login restrictions.

Some of the greatest utility in deploying a Kojoney based honeypot is in its ability to detect attacks from IP ranges within an organizations network. Based on the fact that some attackers were observed attempting to download SSH brute force tools it is likely that compromised SSH servers are sometimes used as SSH brute force scanners. Detecting an internal attacker could provide extremely valuable evidence in an incident detection or response.

Examining malware or attacker toolkits downloaded to the Kojoney honeypot could also prove valuable. Although a wide variety of packages was not observed, the character of the packages that were downloaded is illustrative of the goals of attackers. Additionally, developing hash fingerprints of attacker tools or components could aid in the detection of these materials on other systems, which could be used to detect compromises. As with high interaction honeypots, forensic analysis of this malware is time intensive and may not provide a very high return on investment.

The actual IP addresses captured by the Kojoney honeypot are probably of the greatest value of all the collected data. Because the honeypot was deployed on an unused and un-advertised IP address it is a justifiable conclusion that all traffic observed by the honeypot was de-legitimate and malicious. By identifying and accounting for these malicious IP addresses it is possible to scan server logs from other machines to detect malicious activity on other assets. Although it is important to note that it is possible some IP addresses may not be aggregating points, or rotating pools, for multiple users and not all traffic originating from the identified IP addresses is necessarily malicious.

15. [SSH Black](http://www.openssl.org/docs/apps/ssh_license.html) for multiple users and not all traffic originating from the identified IP addresses is necessarily malicious.
# INFORMATION SECURITY

## 100 MOST COMMON LOGINS

<table>
<thead>
<tr>
<th>Username</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>45403</td>
</tr>
<tr>
<td>test</td>
<td>4128</td>
</tr>
<tr>
<td>guest</td>
<td>872</td>
</tr>
<tr>
<td>postgres</td>
<td>773</td>
</tr>
<tr>
<td>myid</td>
<td>538</td>
</tr>
<tr>
<td>test</td>
<td>480</td>
</tr>
<tr>
<td>backup</td>
<td>456</td>
</tr>
<tr>
<td>web</td>
<td>436</td>
</tr>
<tr>
<td>admin</td>
<td>384</td>
</tr>
<tr>
<td>info</td>
<td>359</td>
</tr>
<tr>
<td>ftp</td>
<td>343</td>
</tr>
<tr>
<td>sales</td>
<td>336</td>
</tr>
<tr>
<td>office</td>
<td>331</td>
</tr>
<tr>
<td>tomcat</td>
<td>323</td>
</tr>
<tr>
<td>webadmin</td>
<td>313</td>
</tr>
<tr>
<td>postfix</td>
<td>306</td>
</tr>
<tr>
<td>mail</td>
<td>305</td>
</tr>
<tr>
<td>toor</td>
<td>301</td>
</tr>
<tr>
<td>testuser</td>
<td>268</td>
</tr>
</tbody>
</table>

## 100 MOST COMMON PASSWORDS

<table>
<thead>
<tr>
<th>Password</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>123456</td>
</tr>
<tr>
<td>user</td>
<td>123456</td>
</tr>
<tr>
<td>sales</td>
<td>123456</td>
</tr>
<tr>
<td>admin</td>
<td>123456</td>
</tr>
<tr>
<td>welcome</td>
<td>123456</td>
</tr>
<tr>
<td>master</td>
<td>123456</td>
</tr>
<tr>
<td>password</td>
<td>123456</td>
</tr>
<tr>
<td>changede</td>
<td>123456</td>
</tr>
<tr>
<td>oracle</td>
<td>123456</td>
</tr>
<tr>
<td>abc123</td>
<td>123456</td>
</tr>
<tr>
<td>admin1</td>
<td>123456</td>
</tr>
<tr>
<td>postgres</td>
<td>123456</td>
</tr>
<tr>
<td>passwd</td>
<td>123456</td>
</tr>
<tr>
<td>admin</td>
<td>123456</td>
</tr>
<tr>
<td>admin123</td>
<td>123456</td>
</tr>
<tr>
<td>admin123</td>
<td>123456</td>
</tr>
<tr>
<td>admin123</td>
<td>123456</td>
</tr>
<tr>
<td>admin123</td>
<td>123456</td>
</tr>
<tr>
<td>admin123</td>
<td>123456</td>
</tr>
</tbody>
</table>

With the increasingly combative nature of Information Technology Security in the workplace, it is needed for skilled Security Professionals with real-world experience has reached critical levels. Theoretical knowledge obtained from educational institutions and industry certification is insufficient to defend sensitive information from miscreants who utilize the latest methods to infiltrate organizations. Due to the unique characteristics and skill sets of this niche industry, Human Resource personnel are often times unable to quantify a potential employee's battlefield ability.

HITBJobs provides an End-to-End solution to corporate organizations and government departments seeking to form or strengthen their internal IT security teams. We provide HR personnel and decision-makers the ability to select and hire future company employees based on resumes gleaned from a non-biased evaluation process conducted by industry peers and experts.

http://www.hitbjobs.com

**SIGN UP AS AN EMPLOYER AND GET:**

- Access to a global database of IT Security professionals available for immediate hire, contract work or headhunting.
- Placement of available positions for hire into a targeted environment.
- Vetting and Verification of potential Employees' curriculum vitae by similarly skilled peers
- Evaluation and Recommendation of potential Employees, via skill-focused interviews conducted by a two tier panel of IT security professionals and notary figures.
- Security Team development, training and consultancy
A Brief Overview on SATELLITE HACKING

By Anchises Moraes Guimarães de Paula, iDefense
As a large portion of worldwide Internet users increasingly rely on satellite communication technologies to connect to the Web, a number of vulnerabilities within these connections actively expose satellites to potential attacks. The implications of such a successful attack are massive, as satellites are the only means of broadcasting communications in many regions around the globe and an attacker could act from anywhere.

Broadband Internet access via satellite is available almost worldwide. Satellite Internet services are the only possible method of connecting remote areas, the sea or countries where traditional internet cable connections are not accessible. Satellite communications are also widely adopted as backup connection providers by several organizations and countries for those times when the terrestrial communications infrastructure is not available, damaged or overloaded. By the end of 2008, an estimated 842,000 US consumers relied on satellite broadband Internet access.

Communications satellites routinely receive and rebroadcast data, television, image and some telephone transmissions, without proper security measures, leading to frequent fraud and attacks against satellite services. Traditional fraud techniques and attack vectors include satellite TV hacking and the use of illicit decoding technology to hack into television satellite signals. In addition, satellite communications are easily susceptible to eavesdropping if not properly encrypted.

SATELLITE BASICS

Satellites are an essential part of our daily lives. Many global interactions rely on satellite communications or satellite-powered services, such as Global Positioning Systems (GPS), weather forecasts, TV transmissions and mapping service applications based on real satellite images (such as Google Maps). Although anything that is in orbit around Earth is technically a satellite, the term "satellite" typically describes a useful object placed in orbit purposely to perform some specific mission or task. There are several satellite types, defined by their orbits and functions: scientific, Earth and space observation, reconnaissance satellites (Earth observation or communications satellites deployed for military or intelligence applications) and communications, which include TV, voice and data connections. Most satellites are custom built to perform their intended functions.

Organizations and consumers have used satellite communication technology as a means to connect to the Internet via broadband data connections for a long time. Internet via satellite provides consumers with connection speeds comparable or superior to digital subscriber line (DSL) and cable modems. Data communication uses a similar design and protocol to satellite television, known as Digital Video Broadcasting (DVB), a suite of open standards for digital television. DVB standards are maintained by the DVB Project, an international industry consortium. Services using DVB standards are available on every continent with more than 100 million DVB receivers deployed, including at least 100 million satellite receivers. Communications satellites relay data, television, images and telephone transmissions by using the transponder, a radio that receives a conversation at one frequency and then amplifies it and retransmits the signal back to Earth on another frequency that a ground-based antenna may receive. A satellite normally contains 24 to 32 transponders, which are operating on different frequencies.

Modern communications satellites use a variety of orbits including geostationary orbits, Molniya orbits, other elliptical orbits and low Earth orbits (LEO). Communications satellites are usually geosynchronous because ground-based antennas, which operate must direct toward a satellite, can work effectively without the need to track the satellite’s motion. This allows technicians to aim satellite antennas at an orbiting satellite and leave them in a fixed position. Each satellite occupies a particular location in orbit and operates at a particular frequency assigned by the country’s regulatory agency, as the Federal Communications Commission (FCC) in the U.S. The electromagnetic spectrum usage is regulated in every country, so that each country has its regulatory agency which determines the purpose of each portion of radio frequency, according to international agreements.

The satellite provider supports Internet access and Internet applications through the provider teleport location, which connects to the public switched telephone network (PSTN) and the Internet. There are three types of Internet via satellite access: one-way multicast, unidirectional with terrestrial return and bidirectional access. One-way multicast transmits IP multicast-based data, both audio and video; however, most Internet protocols will not work correctly because they require a return channel. A single channel for data download via a satellite link characterizes unidirectional access with terrestrial return, also known as "satmodem" or a "one-way terrestrial return" satellite Internet system, and this type of satellite access uses a data uplink channel with slower speed connection technologies (see Exhibit 1). Unidirectional access systems use traditional dial-up or broadband technology to access the Internet, with outbound data traveling through a telephone modem or a DSL connection, but it sends downloads via a satellite link at a speed near that of broadband Internet access. Two-way satellite Internet service, also known as bidirectional access or "astro-modem," involves both sending and receiving data via satellite to a hub facility, which has a direct connection to the Internet (see Exhibit 2).

The required equipment to access satellite communication includes a satellite dish, a receiver for satellites signals, which is a low-noise block (LNB) converter, a decoder, a satellite modem and special personal-computer software. Usually, a single device or PCI card integrates the decoder and modem. Several software programs and online tools are widely available.

Satellite Internet customers range from individual home users to large business sites with several hundred users. The advantages of this technology include a greater bandwidth than other broadband technologies, nearly worldwide coverage, and additional support to television and radio services. Satellite broadband service is available in areas that are terrestrially based wired technologies (e.g., cable and DSL) or wireless technologies cannot operate. The disadvantages, however, are numerous: weather conditions (rain, storms or solar influences) might affect satellite communications, satellites demand expensive hardware and have a complex setup install-
Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

sign and configuration flaws in public- ly available satellite communication networks and protocols, and they are making impressive progress.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.

In 2004, security researcher Warez-

In 2009, Leonardo Nve, a Spanish senior secu-

Radio programs based on a Linux operating system, he was able to monitor Internet satellite trans-

To get an anonymous Internet connection via the satellite broadband network, Nve used this local Internet access connection as an uplink and the hacked satellite con-

RESEARCH ON HACKING SATELLITES

Typical attacks against satellite networks in- clude satellite television hacking (the use of illegal reprogrammed descrambler cards from legitimate satellite equipment to allow unlimited TV service without a subscription) and hacking into satellite networks to trans-

Researchers have been investigating the inherent security, de-

Radio enthusiasts can buy all the hardware near any truck stop for less than USD $500.
CONCLUSION

Governmental, Military organizations and most of the companies included within the critical infrastructure sector such as transport, oil and energy, are using satellite communications for transmitting sensitive information across their widespread operations. This includes the use of satellite communication at industrial plants operating supervisory control and data acquisition (SCADA) systems. The relevance of satellite communication protection and the consequences of a security incident should enforce these organizations to deploy additional security measures to their internal communication technologies. Companies and organizations must implement secure protocols to provide data protection, such as virtual private network (VPN) and secure sockets layer (SSL), since most traffic transmits unencrypted and is widely available in a large geographic area under the satellite’s coverage.

ABOUT THE AUTHOR

Anchises M. G. de Paula, CISSP, is an International Cyber Intelligence Analyst at Defense, a VeriSign company. He has more than 15 years of strong experience in Computer Security and has previously worked as Security Officer in Brazilian telecom companies before becoming Security Consultant for local infosec resellers and consulting companies. Anchises holds a Computer Science Bachelor degree from Universidade de Sao Paulo (USP) and a master degree in Marketing from ESPM. He has also obtained various professional certifications including CISSP, GIAC (Cutting Edge Hacking Techniques) and ITIL Foundations. As an active member of Brazilian infosec community, he was the President of ISSA Chapter Brazil in 2009 and one of the founding members of Brazilian Hackerspace and Brazilian Cloud Security Alliance chapter.

>>REFERENCES

5. Geostationary orbits (also called geosynchronous or synchronous orbits) are orbits in which a satellite always positions itself over the same spot on Earth. Many geostationary satellites (also known as Geostationary Earth Orbits, or GEOs) orbit above a band along the equator, with an altitude of about 22,233 miles. (Brown, Gary. "How Satellites Work" HowStuffWorks. http://science.howstuffworks.com/satellite01.htm). Accessed on Nov 5, 2009.
6. "The Molniya orbit is highly eccentric." The satellite moves in an extreme ellipse with the Earth close to one edge. Because the planet’s gravity accelerates it, the satellite moves very quickly when it is close to the Earth. As it moves away, its speed slows, so it spends more time at the top of its orbit farthest from the Earth. (Telh Hackers. "Catalog of Earth Satellite Orbits / Three Classes of Orbit" Nov 5, 2009. NASA Earth Observatory. http://earthobservatory.nasa.gov/Features/OrbitsCatalog/). Accessed on Nov 5, 2009.
9. Based on "DVB: Satellite Hacking For Dummys" by Warezzman source: http://www.undercon.org/archives/06b/06b08/06b-DVB-Satellite_Hacking.pdf
16. Note: "Bird" is a term for satellite.
17. Based on "DVB: Satellite Hacking for Dummys" by Warezzman source: http://www.undercon.org/archives/06b/06b08/06b-DVB-Satellite_Hacking.pdf
With the advent of new technologies, new protection parameters are evolving. Are technologies good enough to combat the diversified nature of malware? Well, may or may not be. The world has been noticing a new trend of malware which uses office files to corrupt the system, thereby resulting in complete take over of the victim machine. The most versatile nature of office infection comes from the Chinese malware.
The world is grappling with the worst malware from China in recent times. The exploitation index of vulnerable software is really high. Recent attacks involved MS Office for malware infection by the Chinese attackers. The Google provides a little edge in determining the integrity of the website through safe browsing and by flagging a message prior to website’s visit. The search engine also notifies about the malicious websites. The Chinese CN domain is considered as the most spoiled domain for spreading malware throughout the world. 60% of the online malware comes from China, considering the different facets. If one still goes out on search engine, one can find the facts as provided in Listing 1.

Listing 1: Most exploited vulnerabilities

<table>
<thead>
<tr>
<th>CVE: 2008-3005:</th>
<th>crafted Word 97 file (.doc, .wri, or .rtf extension), which could be exploited by attackers to execute arbitrary code by tricking a user into opening a malicious file</th>
</tr>
</thead>
</table>

On aggressive testing domain show some kind of vulnerability. On aggressive testing a number of MS Office files from the Chinese domains, we came across the facts about the most widespread infection, as presented in Listing 1. The above stated vulnerabilities are not the only exploited issues through Chinese malware. The Excel macro can be dangerous and if it is executed, the victim’s machine can be compromized. The opening and closing of MS Office files for the documents in Microsoft Office, including the documents for Word, Excel, and PowerPoint. These file formats reduce the exploitation to avert the security vulnerabilities.

The information leakage through hidden raw data inside the document. The information leakage through the untamed patterns of file format.

The above presented snippets are the normal cases that occur on a day-to-day routine. More sophisticated Office malware does not get traced by the search engine. This is mentioned to show the anatomy of Office base malware. It derive a lot on the way these malware are served on the internet. Primarily, rogue serves are used in order to trigger infection. 6 out of 10 files downloaded of MS Office from Chinese domain show some kind of vulnerable behavior. On aggressive testing a number of MS Office files from the Chinese domains, we came across the facts about the most widespread infection, as presented in Listing 1.

Truth and Lies about MS Office

Office malware which is used by the Chinese attackers for compromising the systems through infection. The important points which should be taken into consideration for analyzing office malware are as follows.

1. Detecting any type of encryption in the objects through OLE properties.
2. Controlling Macro flow by detecting them and putting control over the execution.
3. Parsing OLE2 format and Magic value check.
4. Scrutinizing the OLE objects.
5. A truth about filter as described by Microsoft is stated below

“Filter components for Indexing Service run in the Local Security context and should be written to manage buffers and to stack correctly. All string coped in have explicit checks to guard against buffer overruns. You should always verify the allocated size of the buffer and test the size of the data against the size of the buffer.”

The information leakage through hidden raw data inside the document. The information leakage through the untamed patterns of file format.

The above stated vulnerabilities are not the only exploited issues through Chinese malware. The Excel macro can be dangerous and if it is executed, the victim’s machine can be compromized. The opening and closing of MS Office files for the documents in Microsoft Office, including the documents for Word, Excel, and PowerPoint. These file formats reduce the exploitation to avert the security vulnerabilities.

The information leakage through hidden raw data inside the document. The information leakage through the untamed patterns of file format.

The above stated vulnerabilities are not the only exploited issues through Chinese malware. The Excel macro can be dangerous and if it is executed, the victim’s machine can be compromized. The opening and closing of MS Office files for the documents in Microsoft Office, including the documents for Word, Excel, and PowerPoint. These file formats reduce the exploitation to avert the security vulnerabilities.
The above stated fact clears the point about the complexity of filters which led to vulnerabilities in the past. The functions used in the filter in OLE32.dll are presented in the Figure 4.

A brief explanation of the filter implementation is provided to understand the requisite flow of information through different functions that handle the objects inside the MS Office file format.

MS Office File Format – Detecting the Infection Point

The very straight fact in determining the success of an exploit is based on the reliability of the constructed pattern of the file. Well, the file has to follow the standard format in order to trigger the relative component functionality. So the question that arises here is what makes the MS Office exploits reliable? Where is the shellcode placed? Which part of the MS Office files are used to store the shellcode for execution? The point here is to understand the format of MS Office files from exploitation point of view in order to prove the sustainability of this concept, we will look into the model of infection used by the Chinese writers to spread malware in order to compromise the victim machines.

In order to understand the exploitation, it is always good to have a deep understanding of the Microsoft Office file format. The complexity is a big issue here because of the chaotic nature of MS Office format. It’s very hard to structure all the information at a single point for analysis. The best deterministic solution is to understand the peripherals of different components being a part of the software and using file format specification side by side to verify the details of the vulnerable component of the software. At this point of time, we are going to cover only the requisite details of the MS Office file format.

MS Office holds a component based structure. Component based design always has parent and child objects. Primarily, the same works for MS Office too. The component starts with a root element which serves as a base component of a MS Office hierarchical system. Overall, it is defined as Object Linking Environment (OLE) storage system. The simple reason is that these elements can be formulated as components that are interlinked to perform the unified functions in the software. The OLE storage system consists of the storage components and the stream component. The storage components further comprise of sub storage and sub stream components. Remember the fact that storage components are standalone components which do not show any dependency but this is not true for stream components. On the other hand, these components are directly linked with the required Dynamic Link Libraries (DLLs) which provide an interface with the system. Objects that are embedded in MS Office files are structured in Object pool with unique storage and stream sub components. For example: embedding of XLS sheet in MS Word file.

The aim of malware writers is to create a sub storage object with malicious code in a manner such that the OLE system storage fails to verify the integrity of the storage component. If the OLE system storage verifies the content of the customized storage component, then the malicious document is ready to perform the actions. Usually, there have been no such appropriate measures of verification that were taken in the previous versions, except some of the newly adopted solutions such as VB Macro disintegration by default. This kind of infection has been used in the vulnerabilities that required malformed object in the word itself. For example: VB Macro. Consider that VB Macro is defined for a separate sheet in a Workbook. So, when a user opens sheets in the workbook, respective VB macro is executed there by resulting in infection.

For reliability purposes, the MS Office file header should remain intact. The Figure 5 presents an infection model of MS Office file format based on the storage components. The scanned layout of the vulnerable exploit during our analysis is presented in Figure 6.

We have modified the code in the malware to execute the calc.exe. On execution of rogue.xls in the controlled environment, the calc.exe is executed as presented in Figure 7.

The exploits are using this sort of infection model. Some of the MS docs may have direct streams under the OLESS root. Another type of exploits use continuous stream to provide as a record entry. Consider an exploit which is using a single work book in XLS and a single stream component in root. A basic scan of a malware driven XLS file is presented in Figure 8.

This scan of evil.xls file projects the stream with a component. The exploit is written as a single stream component which should be having the required details. The shellcode analysis is the most strategic point to detect the type of compromise the exploit is going to perform. Automated tools use signature based detection to trigger the exploit. However, good exploits require manual analysis to determine the exact nature. We are going to look into a generic layout of the evil.xls to detect the shellcode. A basic scan of malicious file gives you an edge to determine the layout of shellcode. It only provides the peripheral information but not the core details. The evil.xls is using a stream component and it does not look complex. Before getting into behavioral analysis, a normal lookup through hex editor seems useful, if exploit is not using a complex layout. When evil.xls is decoded as hex strings, we find the shellcode present in the middle of structure. All the headers were intact. On careful analysis, we segregated the components and detected the pattern which looked like shellcode as presented in Listing 2.

In order to understand the nature of this shellcode, it needs to be transformed into assembly instructions in order to determine what it is actually doing. Before getting into behavioral analysis, a normal lookup through hex editor seems useful, if exploit is not using a complex layout. When evil.xls is decoded as hex strings, we find the shellcode present in the middle of structure. All the headers were intact. On careful analysis, we segregated the components and detected the pattern which looked like shellcode as presented in Listing 2.

The shellcode (stripped) turned out to be as presented in Listing 4.

The evil.xls is using a standard bind shell code on Win XP SP2 which gives remote access on port 5344. Always be ready to find a complex shellcode while analyzing malicious Office documents. The infection describes the differential ways used by an attacker to write malware driven exploits. The cases have been analyzed from the Chinese malware samples.

For Shellcode analysis
1. Hex editing is a good approach.
Scan the default strings for different shellcodes.

2. Metasploit additional tools provide an edge in determining the flow of information.

3. Microsoft's msex.lrar and msec.ror are good extensions to be used for conversion and API hash resolving respectively.

4. Given these techniques and decoder are required. Shellcode should be converted for analytical purposes.

5. Good understanding of Assembly is a prerequisite.

For MS Office Scan

1. Ms Office Malware Scanner

2. Microsoft Office Vis

3. Ms Office vulnerability scanner for initial look up.

4. Ms Office file format specification

Additional

It is necessary to have additional techniques and carrier program such as droppers which are used to spread malware into the victim machines. It includes some standard techniques to control the information flow for target specific exploitation. Some of the techniques and issues have been discussed as follows:

Content Disposition – Forcing File downloads

Most of the Chinese malware uses a typical layout of dropping files on the system. Well, the primary reason is to create a required supporting environment which provides an ease of execution. But continuous analysis of various office malware projects a scenario that the attacks are targeted in a well defined manner. It requires downloading of files and it is a big factor to decide how to dispose the files on the system. The appropriate Content-Disposition HTTP header is required which serves the purpose of exploitation in the real time environment.

A regular analysis has shown the fact that malware writers actually use this header in order to dispose files through Drive by Download. The preference can be inline or attached.

Infection. Thus attackers use different attack modes in order to set a right infection environment.

For example, the infected server produces two malicious files in a different manner as described in Listing 5.

The initial look up of these malicious files produces results as stated in Figure 9.

User Agent – Fingerprinting and Redirection

The user agent strings play a very critical role in determining the success of a malware. This is used by the malware writers to perform a status check on the victim machine through the details present in it. Well, it looks simple and basic but this is used in an extensive manner by the detection programs which define the ability of a browser to download the malicious file in the system. If the user agent does not match as per the requirement by the exploit, the browser is forced to get redirected to another domain.

According to RFC 2616 “The User-Agent request-header field contains information about the user agent originating the request. This is meant for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. User agents SHOULD include this field with requests. The field can contain information about software, hardware, language, and the intended user in a manner as described in Figure 10.”

This is more than enough to detect the victim environment.

VB Macro Stringency

The office files provide active script through VB macro which is a source of potential infection. The previous versions of Ms Office 2002, 2003 have been exploited heavily by these VBA macros accompanied with office files. The Chinese office malware uses these VBA macros in an extensive manner in order to run the
So delivering malicious files with different extensions can result in bypassing antivirus solutions. The Chinese malware writers use tricks to evade antivirus solutions to enter into an internal organizational network bypassing gateway security solutions and even desktop antivirus solution to launch the attack by exploiting the system. There are the standard patterns which have been used by Chinese malware for a long time. The bypassing methods include:

1. Most of the malware exploits 8.3 file naming and extension benchmark. Playing around with file extensions enables the attacker to bypass the anti-virus detection. For example, MS Office older and newer versions use some of the extensions as following:

   - Word: .docx, .docm, .dotx, .dotm, .xml
   - Excel: .xlsx, .xlsm, .xltx, .xltm, .xlsb, .xlam
   - PowerPoint: .pptx, .pprt, .ppsx, .ppsm
   - Access: .accdb (new binary format, not Open XML).

3. Fragmenting OLE2 structure into smaller blocks is another trick of bypassing antivirus solutions that are through antivirus software used in whole organizations. As we know OLE2 file format is a block based file system. Any malicious file which is fragmented into block size of 64 or 128 bytes rather than 512 bytes has higher chances of not being detected by the antivirus solution. OLE2 basically searches the free blocks to unit up all the blocks to make a new file. This technique has been used in the wild for subverting antivirus signature based detection or scanning the inline codes.

4. Encoding is also the far best choice of malware writers for obfuscating the code or inside the code of the infected files. US-ASCII and UTF-7 encoding is also the far best choice enemies used by Chinese malware writers to execute the code on the victim machines.

There can be other variations which beat the antivirus functionality. So, all these techniques collectively trigger highly powerful malware through MS Office files which emulate direct from Chinese Malware Factory.

Conclusion

In this paper we have presented the generalized behavior of Chinese malware that exploits MS Office software at par. We have explained the techniques and methods used by MS Office based Chinese malware to show the impact of exploitation in the real world. We have presented the security specific details of file formats and the types of infections that occur in them. These are the widely used techniques used in Chinese malware. With the change in MS Office file formats, new and advanced exploits of XML based file formats are anticipated in the coming time. The security of the end user lies not only in the automated solutions but also on awareness. But the most exploited vulnerabilities in this world are ignorance and ingenuity, rest is only a software construct. *
Reserve Objects in Windows 7

By Matthew “j00ru” Jurczyk

Microsoft is continuously improving the Windows operating system, as well as implementing brand new features and functionalities, which obviously make things much easier for both users and software developers. On the other hand, as new code is being introduced to the existing kernel- or user-mode modules, new opportunities might be opened for potential attackers, aiming at using the system’s capabilities in favor of subverting its security. Proving the above thesis is one of this paper’s objectives – as the reader will find out, there are always two sides of the coin.
A s indicated in my previous article – Windows Objects in Kernel Vulnerability Exploitation – the Object Manager is a crucial subsystem implemented as a part of the Windows Executive, since it manages access to every kind of system resource utilized by the applications. In this article, I would like to introduce a new type of objects – Reserve Objects – which have been shipped together with the Windows 7 product. As it turns out, the nature of these objects makes it possible to use them as a very handy helper tool, in the context of various, known kernel attacks.

Furthermore, according to the author’s observations, the mechanism described in this paper is also the initial phase of development, and is very likely to evolve in the future Windows versions – in such case, it might become even more useful for ring-0 hackers.

New Windows = new system calls.

Because of the fact that Microsoft developers are gaining feedback and overall experience of how well the current system mechanisms are working, the native system-call set as well as official API differ between distinct Windows versions (please note that while the API interface must provide backwards compatibility, there is no such guarantee regarding native calls). As a very good example, one should take a look at a comparison table1, presenting changes between Windows 7 and Windows Vista SP1, in terms of ntdll.dll exported symbols. As can be seen, numerous new functions have been added, while only a couple of them were removed.

A majority of the new function set is composed of names beginning with Rtl* (Run-time library), implemented as helper routines, commonly utilized by the official API code (such as kernel32.dll). Aside from these, one can also find around fifteen new Nt* symbols, which represent fresh kernel functions that are exposed to ring-3, so that user-defined applications (for more likely, system libraries) can take advantage of what the new system provides. Listing 1 presents a complete set of new ntnt names within our interest.

What shouldn’t be a surprise is the fact that most of the new symbols do not implement a completely new feature – instead, they seem to extend the functionalities that have already been there, using additional parameters, and providing extra capabilities which were not present before. For instance, the NtCreateProfileEx function adds in options that were not available in older NtCreateProfile - the same effect affects symbols like NtOpenKeyEx(), NtQuerySystemInformationEx() and many others.

To get to the point, the functions that we are mostly interested in, are:

- NtAllocateReserveObject - system call responsible for creating an object on the kernel side – performing a memory allocation on the kernel pool, returning an appropriate Handle etc.
- NtQueueApcThreadEx – system call capable of advancing the previously allocated Re- serve Object while inserting an APC (Asynchronous Procedure Call) into the specified thread’s queue,
- NtSetIoCompletion – system call incrementing the pending IO counter for an IO Completion Object. As opposed to the basic NtSetCompletion function, it can utilize the Re- serve Objects, as well.

As can be seen, all of these three above functions have been introduced in Windows 7 and, at the same time, no accurate information regarding these routines is publicly available. In order to get a good understanding on what this new types of object really are, let’s focus on the allocation function, in the first place.

ntNtAllocateReserveObject

In order to give you the best insight of the underlying mechanisms, I would like to begin with a thorough analy- sis of the allocation function; you can find its pseudo-code (presented in a C-like form) in Listing 2.

The system call requires three argu- ments to be passed – one of which is an output parameter, used to return the object handle to the user’s application, while the other two are meant to supply the type and additional information regarding the object to be allocated. Right after entering the function, the hObject pointer is compared against ntNtUserProbeAddress, ensuring that the address does not exceed the user memory regions. Moreover, since the number of supported reserve object types is limited (and equals two at the time of writing this paper), every higher number inside ObjectType bails out the function execution.

After the sanity checks are performed, an internal ntObCreateObject routine is used to create an object of a certain size and type (you can find the func- tion’s definition in Listing 3) – the inter- esting part begins here. As can be seen, both the ObjectType and ObjectSizesToAllocate parameters are volatile – instead, the PspMemoryReserveObj- ectTypes and PspMemoryReserveOb- jects internal arrays are employer, together with the ObjectType param- eter used as an index into these.

As mentioned before, only two types of reserve objects are currently avail- able: UserApcReserve and IoComple- tionReserve objects. Each of them has a separate OBJECT_TYPE descriptor structure, containing some of the ob- ject characteristics, such as the name, allocation type (paged/non-paged), and others. The pointers to these structs are available through the PspMemoryReserveObjectTypes array; the object descriptors for both types are presented in Listing 4. This obser- vation alone implies that one is able to choose the object type to be used.

The second dynamic argument passed to ObCreateObject is the size of a buffer, sufficient to hold the ob- ject’s internal structure. Considering the differences between the size of a machine word on x86 and x86-64, one shouldn’t be surprised that the object sizes stored in the PspMemo- ryReserveObjectSizes array are also distinct. The exact numbers stored in the aforementioned array is present- ed in Table 1.

After the object is successfully allo- cated, the buffer is filled so that no trash bytes could cause any trouble from this point on. Next, in case of IoCompletion allocation, Object- Buffer is filled with some initial values, such as a pointer to itself or a callback function address. Please note that no initialization is performed for an User- Apc object, which remains empty un- til some other function references the object’s pool buffer.

Going further into the function’s body, a call into ntOblsntObjectEx is issued, in order to put the object into the local process’ handle table (i.e. re- trieve a numeric ID number, represent- ing the resource in ring-3). The handle is put into the local hOutputHandle variable, and respectively copied into the hObject pointer, specified by the application (and already verified). If everything goes fine up to this point, the system call handler returns with the ERROR_SUCCESS status.

In short, NtAllocateReserveObject makes it possible for any system user to allocate a buffer on the non- paged kernel pool, and obtain a HANDLE representation of this buff- er in user-mode. As it will turn out later in this paper, the above can give us pretty much control over the kernel memory, when exploit- ing custom vulnerabilities.

ntNtQueueApcThreadEx

The first user-controlled function (i.e. system call) allowing the operating system to operate on the Reserve Objects is re- sponsible for queuing Asynchronous Procedure Calls2 in the context of a specific thread. Once a call to Listing 5 presents a C-like pseudo-code of the function’s real implementation.

First of all, the KTHREAD address ad- dressed to the input hThread parame- ter is retrieved using ObReferenceOb- jectEx. If the call succeeds, and the thread doesn’t have a SYSTEM_ THREAD flag set, the execution can go two ways:

36 HTB MAGAZINE 1 JULY 2010

37 JULY 2010 HTB MAGAZINE

Listing 1. Interesting system calls

Listing 2. NtAllocateReserveObject function pseudo-code

<table>
<thead>
<tr>
<th>ObjectTypes</th>
<th>Sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x2C</td>
</tr>
<tr>
<td>0x01</td>
<td>0x58</td>
</tr>
<tr>
<td>0x02</td>
<td>0x34</td>
</tr>
<tr>
<td>0x03</td>
<td>0x60</td>
</tr>
</tbody>
</table>

Table 1. PspMemoryReserveObjectSizes contents on 32- and 64-bit Windows 7 architecure
If NtApcService is a non-zero value, the object’s memory block address is obtained, and stored in ApcBuffer. Next, an atomic compare-exchange operation is performed, in order to mark the reserve object as “busy” – the first DWORD of the buffer is used for this purpose. ApcBuffer is increased by size(DWORD), pointing to the beginning of the _KAPC structure. Eventually, the Kernel- and RundownRoutine function pointers are set to adequate addresses, so that the reserve object is correctly freed after the APC finishes its execution.

If NtApcReserve equals zero, a straightforward allocation of 0x30 (Windows 7 x64) or 0x58 (Windows 7 x86-64) bytes is performed on the Non-paged Pool, and the resulting pointer is assigned to ApcBuffer. The Kernel-Routine and RundownRoutine pointers are set to IoDeallocateApc and ExFreePool, respectively.

After the if statement, a KernalizeApc call is made, specifying the ApcBuffer pointer as the destination KAPC address, and passing the rest of the previously initialized arguments (KernelRoutine, RundownRoutine, ApcArgument1). Finally, a call to KeInsertQueueApc is issued, which results in having the KAPC structure pointed to by ApcBuffer inserted into the APC queue (as intended in consideration).

On Microsoft Windows versions prior to 7, the user was unable to get the kernel to make use of a specific memory block of a known address. Instead, the latter execution path of the above if statement was always taken. If the application really wanted to queue an APC, the required space was allocated right before queuing the structure – both these conditions had to happen inside one routine (system call). Therefore, no kernel memory address was revealed to the user, thus making it impossible to utilize the KAPC structures (on the kernel pool) in stable attacks against the kernel. Fortunately for us, times have apparently changed :) .

Listing 3. The NtApcService routine pseudo-code

Listing 4. The OBJECT_TYPE structures associated with the Reserve Objects

Listing 5. The NtQueueApcThreadEx routine pseudo-code

An internal IoSetCompletionEx functionality is called, and in case it fails for any reason, the object is restored to its previous state (i.e. with the first DWORD set to zero), and the function bails out. Otherwise, the ERROR_SUCCESS status is returned.

Malicious utilization

Now, as the Reserve Object is clear, we can finally find out some practical examples of how a potential attacker can take advantage of the new object types.

UserApcReserve as a write-what-where target

Because of the fact that Windows kernel make it possible for a user-mode thread to retrieve information regarding all active objects present in the system (including information like the owner’s PID, numeric handle value, the object’s descriptor address and others), one is able to find the address associated to a given object, very easily. More information on how to extract this kind of information from the operating system can be found in the NtQueryObject documentation (together with the SystemHandleInformation parameter).

In general, when a kernel module decides to manually allocate memory using kernel pools, the resulting address (returned by ExAllocatePool or equivalent) never leaves kernel mode, and therefore is never revealed to the user-mode caller. Due to this “limitation”, and because of the fact that it is very unlikely to successfully foresee or guess the allocation address – much memory areas cannot be used as a reasonable write-what-where target. For instance, the NtQueueApcThread system call has always used a dynamic buffer to store the required KAPC structure on every Windows NT-family version previous to Windows 7 – and so, it never appeared to become targeted by a stable code-execution exploit.

Nowadays, once the user can choose between safe NtQueueThreadApc and NtQueueApcThreadApc (in fact, it is impossible to utilize the memory region with known address), things are getting more interesting. The attacker could allocate and initialize the UserApcReserve object, find its precise address and overwrite the KAPC structure contents (using a custom ring-0 vulnerability), and finally flush the APC queue, thus performing a successful Elevation of Privileges attack. A pseudo-code of an exemplary exploit is presented in Listing 7.

Payload inside kernel memory

Across various security vulnerabilities related to the system core, the specific conditions in which code execution is triggered, are always different. As a consequence of numerous back-ground mechanisms keeping the machine alive, a potential attacker can never predict every single part of the system state, at the time of performing the attack. In some cases, there is no guarantee that the payload code is even executed in the same context as the process that issued the vulnerability. This, in turn, could pose a serious problem in terms of creating a reliable exploit, which should launch the shellcode no matter what’s currently happening on the machine.

One possible solution could rely on setting-up the necessary code somewhere inside a known address in process-independent kernel memory; and

If we take a closer look at the KAPC structure definition from the x86-64 architecture OS (presented in Listing 8), we can observe that starting with offset +0x030, there are four user-controlled fields – all of them defined through the NTQueueType structure:
- NormalRoutine – a pointer to the user-specified callback function, called when flushing the APC queue,
- NormalContext – first routine argument, internally used as the KeInitiatedApfc function parameter,
- SystemArgument1, SystemArgument2 – second and third arguments, passed to the KeInitiateQueueApc function.

Being able to control roughly four variables in a KAPC, each of which has the machine word size (32 bits on x86, 64 bits on x86-64), one can insert 16 or 32 bytes of continuous data (depending on the architecture), at a known address! Furthermore, because of the fact that one can create any number of such objects, it is possible to store a chain of 16/32-byte long code chunks, each connected to the successive one using a simple JMP (or any other, shorter) instruction. The overall idea is presented in Image 1.

DEP in Windows x64 kernel
One important issue regarding the idea presented in this section is the uncertainty whether it is possible to execute the code placed inside a pool allocation safely, i.e. avoid problems with DEP-like protections, that are continuously extended and improved by Microsoft. As MDSON states, however, the hardware-enforced Data Execution Prevention aims to protect only one (32-bit) platforms or three (64-bit) crucial parts of the non-executable kernel memory, leaving the rest on its own.

DEP is also applied to drivers in kernel mode. DEP for memory regions in kernel mode cannot be selectively enabled or disabled. On 32-bit versions of Windows, DEP is applied to the stack by default. This differs from kernel-mode DEP on 64-bit versions of Windows, where the stack,paged pool, and session pool have DEP applied.

As can be seen, both the stack and all types of kernel pools except the non-paged pool are protected against code execution. Let’s take a look at the OBJECT_TYPE structure contents associated to UserApcReserve and IoCompletionReserve objects (Listing 9). Fortunately for us, both objects are allocated on non-paged pool, which means that one can execute the code within a custom KAPC without any real trouble.

Heap spraying-like techniques
If one realizes that the reserve objects are actually small pieces of memory controlled by the user, in terms of position and content, a wide variety of possible ways of utilizing arises. For instance, according to the author’s research, it is likely that a user-mode process is able to partially control the kernel pools memory layout, by properly manipulating the Reserve Objects present in the system, i.e. by allocating and freeing appropriate chunks of memory. Due to the fact that any process is able to queue new KAPCs using NtAllocateReserveObject + NTQueueType, one can use its user-controlled values – all of them being used and so on – as for now, this subject is left open to be researched by any willing individual. Overall, what should be remarked is that there are still countless ways of evading the generic protections ceaselessly introduced by the operating system vendors. The game is not over, yet.

Conclusion
In this paper, the author wanted to present a new, interesting mechanism, introduced in the latest Windows versions; show some possible ways of turning this functionality against the system and make it work in the attacker’s favor; and finally present how fresh, legitimate features created by the OS dev’s should be analyzed in the context of exploitation usability. As old ideas and methods already have their countermeasures implemented in the system core, new ones have to be developed – the best source for these, in my opinion, is the mechanisms such as the one described in this paper.

It is believed that many interesting, sophisticated attacks against the kernel can be carried out using functionalities like Reserve Objects, therefore the author wants to highly encourage every individual interested in ring-0 hacking, to investigate the subject on his own and possibly contribute to the Know-How kernel exploitation field in some way. Good luck!

References
2. Gyslain Coldwind, Changes in Microsoft Windows 7 vs Microsoft Vista SP1, msbl, http://gyslain.coldwind.pl/?s=14
Due to the dynamic features of Javascript, obfuscation of the exploit code is quite easy. As Javascript is an interpreted language, websites have to deliver the source code to the user. Therefore, obfuscation of Javascript is commonly applied to protect the source code against simple copy and paste, saving the intellectual property of the developer. Algorithms used for obfuscating exploit code have vastly improved in the last years. Commercial tools are available, and even obfuscators using steganography (hiding payload in whitespace formatting) have been developed.

Problems detecting Javascript Malware

This leads to the problem that known signatures do not work due to the dynamically obfuscated code, while the obfuscation itself is no prove for the code being malicious. Thus, an anti virus engine has to analyze and emulate Javascript until it sees the real functionality of a script, in order to detect malicious code. As mentioned before, Javascript is a language with countless ways to hide code - it supports some sorts of metaprogramming, meaning code can modify itself and create new code. Decrypting a string and executing the result with the eval() function is a well known method. Since the code has to be able to execute itself, every Javascript obfuscator integrates the key and decrypts itself with a massively obfuscated algorithm.

Different goals and constraints of Javascript packers

From an attackers’ point of view, there is one advantage over the website developer that has not been taken into account in most Javascript packers: the time factor. The obfuscated code in a legitimate website has to execute almost as fast as if it were not packed. Nevertheless, from an attackers’ point of view we do indeed have some time - it does not matter if the exploit executes in milliseconds or 2 seconds - the average victim won’t notice it and would not even be able to find the task manager to kill the process in that time.

However, the anti virus scanner has to handle the javascript in the same way as the website developer - the execution may not take significantly more time than without scanning it, so at best it has tenths of a second.

Taking advantage of the time factor

To take advantage of this, the packer needs to create code that cannot be analyzed within a certain timespan. As the technique should not rely on complexity, it has to be implemented in a way that makes it impossible to analyze the code within a short time, regardless of how well the Javascript emulation of the anti virus engine works.

Again, the solution is to encrypt the payload. In contrast to the existing packers, this new one does not in-

With the rise of web-based threats, Javascript has become an increasingly used language for client-side attacks. Most vulnerabilities in browsers require script code to be executed in the victims browser. In most cases, these scripts prepare the exploitation and trigger a vulnerability.
Implementing the concept

To implement the cryptographic functionality, this packer uses a free MDS library that cannot be detected as malicious, as it is used on legitimate sites.

The packer uses the MDS hash of a key to xor-encrypt the payload, whereby the key itself is splitted into three parts. The browser is given the first part with the delivered script. The second part is the query string and the third part is the guess-key. The packer returns the target URL where the payload gets executed. As a result, the anti virus engine times out, but the payload gets executed in the browser.

Today’s highly optimized JavaScript engines in modern browsers, by executing the brute force algorithm quite fast, give us even more of an advantage.

The third part of the key has to be guessed. To make this possible, the browser gets the MDS hash of that part, whereby the key itself is splitted into three parts. The browser is given the first part with the delivered script. The second part is the query string and the third part is the guess-key. The packer returns the target URL where the payload gets executed. As a result, the anti virus engine times out, but the payload gets executed in the browser.

The third part of the key has to be guessed. To make this possible, the browser gets the MDS hash of that part, whereby the key itself is splitted into three parts. The browser is given the first part with the delivered script. The second part is the query string and the third part is the guess-key. The packer returns the target URL where the payload gets executed. As a result, the anti virus engine times out, but the payload gets executed in the browser.

The third part of the key has to be guessed. To make this possible, the browser gets the MDS hash of that part, whereby the key itself is splitted into three parts. The browser is given the first part with the delivered script. The second part is the query string and the third part is the guess-key. The packer returns the target URL where the payload gets executed. As a result, the anti virus engine times out, but the payload gets executed in the browser.

The implementation of the last part is important: the key has to be randomly chosen so that it takes about 2-3 seconds to crack. If a weak key is chosen, one of the first guesses will be the right one. To circumvent this issue, 5 keys of a smaller size (1/5) need to be generated. 3 keys are needed to decrypt the payload and cannot be analyzed without access to the query string.

After calculating the unknown part of the key (based on the known MDS hash), the victims’ browser is able to reassemble the original key. This is thereafter used to decrypt the payload, which then gets executed, using the eval() function.

Another difficulty lies in the execution time of JavaScript in different browsers. Scripts that will run in one second in the latest browser versions will take vast amounts of time in older ones (e.g., Internet Explorer 6). As most exploits target specific browser versions, the performance of the executing JavaScript engine is known. Therefore, the packer can be given the expected speed of the executing JavaScript engine (as a consequence, an AV scanner is in advantage if an old browser is attacked).

Integration into the Metasploit framework and further use cases

Listing 1 shows parts of the original Aurora exploit from the Metasploit framework. All variable names are manually set to random strings, making the code hard to read and maintain. The newly developed packer leaves the original code almost untouched. The packer uses the MD5 hash of a key to xor-encrypt the payload, whereby the key itself is splitted into three parts. The browser is given the first part with the delivered script. The second part is the query string and the third part is the guess-key. The packer returns the target URL where the payload gets executed. As a result, the anti virus engine times out, but the payload gets executed in the browser.

The implementation of the last part is important: the key has to be randomly chosen so that it takes about 2-3 seconds to crack. If a weak key is chosen, one of the first guesses will be the right one. To circumvent this issue, 5 keys of a smaller size (1/5) need to be generated. 3 keys are needed to decrypt the payload and cannot be analyzed without access to the query string.

Another difficulty lies in the execution time of JavaScript in different browsers. Scripts that will run in one second in the latest browser versions will take vast amounts of time in older ones (e.g., Internet Explorer 6). As most exploits target specific browser versions, the performance of the executing JavaScript engine is known. Therefore, the packer can be given the expected speed of the executing JavaScript engine (as a consequence, an AV scanner is in advantage if an old browser is attacked).

Integration into the Metasploit framework and further use cases

Listing 1 shows parts of the original Aurora exploit from the Metasploit framework. All variable names are manually set to random strings, making the code hard to read and maintain. The newly developed packer leaves the original code almost untouched. The packer uses the MD5 hash of a key to xor-encrypt the payload, whereby the key itself is splitted into three parts. The browser is given the first part with the delivered script. The second part is the query string and the third part is the guess-key. The packer returns the target URL where the payload gets executed. As a result, the anti virus engine times out, but the payload gets executed in the browser.

The implementation of the last part is important: the key has to be randomly chosen so that it takes about 2-3 seconds to crack. If a weak key is chosen, one of the first guesses will be the right one. To circumvent this issue, 5 keys of a smaller size (1/5) need to be generated. 3 keys are needed to decrypt the payload and cannot be analyzed without access to the query string.

Another difficulty lies in the execution time of JavaScript in different browsers. Scripts that will run in one second in the latest browser versions will take vast amounts of time in older ones (e.g., Internet Explorer 6). As most exploits target specific browser versions, the performance of the executing JavaScript engine is known. Therefore, the packer can be given the expected speed of the executing JavaScript engine (as a consequence, an AV scanner is in advantage if an old browser is attacked).
will encode the code with the eval() function once it is decrypted. Listing 2 shows the exploit code using the new packer (not using the query string feature for simplicity).

A shortened example of the created Javascript code is shown in Listing 3.

Figure 2 shows the detection of the first version of the aurora exploit from VirusTotal results for the packed version. Figure 3 contains the results for the packed version. Though VirusTotal does not exactly reflect an anti-virus product running on an attacked client, this does show that the packer is successful in helping circumvent anti-virus engines.

In contrast to other packers, the purpose of this one lies solely in penetration testing scenarios – except for the needed techniques, no additional steps have been taken to complicate manual analysis.

The presented solution also works for Javascript embedded in PDF files. Although obfuscated code in PDF files is not as common as in web pages, it seems that many AV scanners trigger on Javascript only if they see the signature of a vulnerable function that is needed and AV scanners cannot solely rely on known signatures – only will they see the signature of a vulnerable function that is going to be exploited.

Countermeasures

Though the analysis and detection of the original Javascript code is not possible due to time constraints, the

Recent research deals with the problems of client side attacks that are often too complex to analyze in a short time – the project Razorback™ (formerly known as Near Real-Time Detection) from the Sourcefire Vulnerability Research Team Labs might be a way to handle the problems arising with the techniques described in this paper, especially from the perspective of a network IDS (See http://labs.snort.org/razorback/).

Another solution are behavioral-based detection and whitelisting: even if the Javascript code cannot be analyzed, the malicious activities of the final payload could still be detected and prevented.

Conclusion

Although the described technique might pose another difficulty to AV products, it is likely to be used in targeted attacks. These are often insufficiently considered aspect - the exploits that are widely spread will be found by AV vendors and signatures will be created. In contrast, code used in a targeted attack will most likely never be seen by an AV vendor.

No tested AV product has detected the generated Javascript code samples as being malicious. Though this is valid for most new packers, I think it will be true for this one for quite some time. It shows again that new techniques like behavioral based detection are needed and AV scanners cannot solely rely on known signatures - those will not be found within reasonable time when the code is packed with the described techniques.

Please visit the author’s website at http://revelless-coding.blogspot.com/p/project.html for the latest updates on the project.
Non-Invasive Invasion: Making The Process Come To You

By Shawn Wilcoxen

E xternal hacks and tools are the fastest to be blocked simply due to hooks placed on system calls that are frequently needed to interface with the target game.

This article covers a bypassing method that allows external hacks and tools to access any target process by using DLL injection to bring the target process to the tool/hack, avoiding any calls to hooked system functions that would trigger anti-cheat action if called directly.

In this article, there are 2 separate entities of code: One for the DLL to be injected into the game, and one for the tool/hack that will interface with the DLL in order to get information about the target process secretly. The terms “DLL” and “client” will be used to refer to these applications respectively from here out.

Creating THE DLL
A DLL is the foundation for the entire process. We begin by creating a basic skeleton DLL and injecting it into a process. The code for our DLL at this point is nothing special. See Listing 1.

The call to Beep() is simply to let us know that the DLL has been loaded into the target process. Use any DLL injector, pick a random process, and inject your skeleton DLL. If you hear a beeping sound, your DLL is working and has been successfully injected.

Note: On Windows 7, the Beep() function uses the default soundcard, unlike other versions of Windows which relay the sound to the motherboard speaker.

Note: To debug the DLL using Microsoft Visual Studio®, open the project properties (Alt-F7) and select the Debugging property page. Set the Command to “winmine.exe” (with or without quotes) on Windows XP or “Minesweeper.exe” on Windows Vista or Windows 7. This should be done on the Debug build (the Release build is optional). With the Debug build active, press F5 to launch Minesweeper, then use any software to inject your DLL (MHS, CheatEngine, etc.) into the newly opened Minesweeper. If you have set a breakpoint inside DllMain(), you will see it being hit as soon as you inject the DLL manually. You can single-step and debug normally from here.

Injecting THE DLL
Once we have tested that the DLL is ready for injection, we need to test our methods for injecting it into all processes silently. There are several ways to inject a DLL into a target process, and ultimately any of them will work for our purposes as long as the injection process is not detected and hampered. Anti-cheat software typically detect brute-force injection methods using CreateRemoteThread() and SetWindowsHookEx(), but if these methods work on the target process(es) of your choice, feel free to use them. The method explored in this article is the AppInit_DLLs registry key which is used frequently by non-intrusive applications.

The easiest way to test our method is to manually add the path to our DLL to the HKKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs registry key. To debug the DLL using Microsoft® Visual Studio®, open the project properties (Alt-F7) and select the Debugging property page. Set the Command to “winmine.exe” (with or without quotes) on Windows XP or “Minesweeper.exe” on Windows Vista or Windows 7. This should be done on the Debug build (the Release build is optional). With the Debug build active, press F5 to launch Minesweeper, then use any software to inject your DLL (MHS, CheatEngine, etc.) into the newly opened Minesweeper. If you have set a breakpoint inside DllMain(), you will see it being hit as soon as you inject the DLL manually. You can single-step and debug normally from here.

Note: In Windows XP, this task is simple. In Windows Vista, security measures will probably prevent you from using this method. Windows 7 can work, but only after you jump through some hoops and modify 2 other registry values in the same location (LoadAppInit_DLLs and RequireSignedAppInit_DLLs).

For these systems, it is better to use one of the alternative methods for DLL injection.

After setting AppInit_DLLs to “F:\temp\MyDll.dll”, without the quotation marks. The value is delimited by

Listing 1. Our DLL shell simply beeps to let us know it has been injected.

BOOL APIENTRY DllMain( HMODULE _hModule, DWORD  _dwReason, LPVOID _lpvReserved ) {
    switch ( _dwReason ) {
    case DLL_PROCESS_ATTACH : {
        ::Beep( 1000, 10 );
        break;
    }
    return TRUE;
}

Note: To manually add the path to our DLL to the HKKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs registry key, use regedit and set the value to “F:\temp\MyDll.dll”. Without the quotation marks. The value is delimited by

Listing 1. Our DLL shell simply beeps to let us know it has been injected.

BOOL APIENTRY DllMain( HMODULE _hModule, DWORD  _dwReason, LPVOID _lpvReserved ) {
    switch ( _dwReason ) {
    case DLL_PROCESS_ATTACH : {
        ::Beep( 1000, 10 );
        break;
    }
    return TRUE;
}

Note: To debug the DLL using Microsoft® Visual Studio®, open the project properties (Alt-F7) and select the Debugging property page. Set the Command to “winmine.exe” (with or without quotes) on Windows XP or “Minesweeper.exe” on Windows Vista or Windows 7. This should be done on the Debug build (the Release build is optional). With the Debug build active, press F5 to launch Minesweeper, then use any software to inject your DLL (MHS, CheatEngine, etc.) into the newly opened Minesweeper. If you have set a breakpoint inside DllMain(), you will see it being hit as soon as you inject the DLL manually. You can single-step and debug normally from here.

Note: In Windows XP, this task is simple. In Windows Vista, security measures will probably prevent you from using this method. Windows 7 can work, but only after you jump through some hoops and modify 2 other registry values in the same location (LoadAppInit_DLLs and RequireSignedAppInit_DLLs).

For these systems, it is better to use one of the alternative methods for DLL injection.

After setting AppInit_DLLs to “F:\temp\MyDll.dll”, without the quotation marks. The value is delimited by
Listing 2. Our CProc class allows for easy upgrading of the methods used to interact with remote processes.

class CProc {
public:
  static HANDLE OpenProcess(DWORD _dwDesiredAccess, BOOL _bInheritHandle, DWORD _dwProcessId);
  static METHOD findProc(DWORD _dwProcessId);
  static METHOD findProc(DWORD _dwProcessId, HANDLE &hProc);
private:
  METHOD readProcessMemory(HANDLE hProc, LPVOID lpvAddress, SIZE_T &stSize);
  METHOD writeProcessMemory(HANDLE hProc, LPVOID lpvAddress, SIZE_T stSize);
  METHOD breakProcess(HANDLE hProc);
};

Listing 2. Types of messages we can handle.

typedef struct HITB_COMMUNICATION_BUFFER {
  enum HITB_MESSAGE { HITB_IDLE, HITB_INITIATECONTACT, HITB_COMMUNICATION, HITB_DISCONNECT, HITB_INITIAL_REPLY, HITB_LAST } mType;
  DWORD dwId;
  union HITB_COM_DATA {
    HITB_COMMUNICATION_BUFFER cbBuffer;
    union {
      LPCVOID _lpvBaseAddress, LPVOID _lpvBuffer, SIZE_T _stSize, SIZE_T * _psize;
      PVOID _lpvAddress, SIZE_T _stSize;
    }
  } cbBuffer;
}.;

Listing 3. Gathering each of the message formats together in a union.

Listing 5. Gatheres each of the message formats together in a union.

Listing 6. The code for initiating contact from the DLL.

Listing 7. The helper function.

Immediately after applying these changes to the registry, loading an application such as Windows Calculator results in a short beep, confirming that the system is working. In order to proceed, remove the entry from the registry and reboot.

COMMUNICATION THEORY

The DLL needs to broadcast its presence to every other process in the system. If one (or more) of the processes responds, the DLL needs to make a ‘connection’ to that process, allowing more streamlined communication between them.

There are many ways to set up a private communication network. By ‘private’, we mean a communication network that should not trigger alarms inside the software of interest. For example, if your communication network uses SendMessage() with HWND_BROADCAST and (WM_USER + 0x100) parameters, an anti-cheat could be updated to pick up this message and assume your communication network is active, shutting down the game.

There are many ways to mask the communication network, however. One method that takes work to detect is via LAN communication. Another possibility is to simply not send messages to the target window. The name of your DLL should be random, so only the DLL itself and your client software actually know its name. If your client software unloads the DLL from itself, the DLL only needs to send its secret message to processes that do not have that DLL loaded. This is the method chosen for this article.

The client software may not initiate contact in any way, since that may disturb any protections surrounding the game. But at the same time the DLL does not know beforehand if a given process is the client, so a special address for data sharing cannot be

preallocated. The method discussed uses SendMessage() (only to applications that do NOT have the DLL loaded) to initiate the first contact, and then uses ReadProcessMemory() and WriteProcessMemory() thereafter to communicate.

GETTING READY

There are a few key issues to cover before we can implement the communication layer. Firstly, it is vital that you create a class for working with the target process. Wrap system functions inside this class so that they can be overridden and changed later. For example, instead of calling ReadProcessMemory() directly, call the wrapper function on an instance of your class, which will in turn call ReadProcessMemory(). Later, when you want to add a kernel driver to change how you read process memory, you can simply override the function on your class and create an instance of that class instead.

All code that uses the wrappers on your class will be automatically updated. A truncated example of such a class is shown in Listing 2.
Listing 8. The main logic for the DLL, which primarily sits and searches for client applications.

```c
Listing 9. DllInProc scans a process for a module whose name matches the name of this DLL.

class CDllMagic {

  static DWORD WINAPI DllMain(HMODULE hModule, DWORD dwReason, LPVOID lpvParam);

  // the thread that monitors all processes searching for the client process.
  DWORD WINAPI DllQuery(HMODULE pModule, LPVOID lpvParam);

  // When the thread first begins, some required DLL’s may not have
  // been loaded yet.
  HANDLE hSnap = ::CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0UL);

  if ( hSnap == INVALID_HANDLE_VALUE ) {
    // set default to the correct value and zero’s everything else.
    ::FreeResource( m_pEntries );
    return false;
  }

  return true;
}
```

Listing 10. Our new DLL entry point.

```c
Listing 11. The client message handler used to catch the initial message sent by the DLL.

```c
Listing 12. The shell of our connection class from the client's point of view.

```c
```

Which is basically all of them. Once a reply is detected, we will send the buffer off to be managed by a class that will handle all communications between the DLL and the replied client application.

**Communication Buffers**

Our communication system works by letting each application (the DLL and the client) write information to a designated area of RAM inside the receiver which the receiver is assumed to be constantly monitoring. Each message has a specific format known to both the DLL and the client software. We model this in code via structures, unions, and enumerations.

Firstly, the actual message types must be enumerated, as shown in Listing 3.

Secondly, the format of each message must be defined as shown in Listing 4.

Finally a union allows a single structure to contain data in any of the formats in Listing 4. See Listing 5.

Note that this structure will be used in both the DLL and the client.

**First Contact**

Initial contact is attempted whenever the DLL spies an application without the DLL inside it. Since the DLL is planned to be injected into every process at start-up (but is not restricted so), we assume any processes without the DLL have purposely removed the DLL from themselves and are likely to be the client software with whom we want to make a connection. Additionally, this prevents sending suspicious and detectable messages to the game itself, which is assumed to be protected by an anti-cheat.

All contact works the same generally speaking. The client software will have a region of memory that is monitored by the DLL, and when changes are detected, a response is given back using the same buffer. But the initial contact requires sending a Windows message to set all of this up.

To complicate things, the DLL does not know which window in the client is the window that is designed to respond to first contact, so it must send the message to every window on every thread of the client. The code is straightforward, but long. The comments in Listing 6 explain the code.

The call to _EnumThreadWindows() requires the below helper function. This is the actual function that posts the message to the client software hoping for a reply, and is shown in Listing 7.

Here, m_lpcbBuffers is a member of our class defined as std::vector<LPHITB_COMMUNICATION_BUFFER> m_lpcbBuffers. We keep records of each initial communication here and use this record to check for replies.

With this function, once we have a process ID we suspect may be a client, all we have to do is call CDllMagic::InitiateCommunication() and the process of communication will begin. Now all we have to do is find processes suspected of being a client.

**Finding The Client**

Finding potential clients is conceptually simple. Searching must happen constantly, so the routine will be a second thread, looping infinitely until told to stop. It must not eat CPU resources, so its priority must be low and it must sleep a while between iterations.

Our search loop also has the dirty duty of finding processes that never responded and are no longer open and removing our record of that communication, freeing resources for later. This code has been omitted for brevity, however. The comments document the code. Notice that this is a static function since it will be used in a later call to ::CreateThread(). See Listing 8.

```c
```

```c
```

```c
```
DLLInProc simply scans the given process for modules matching the name of the current DLL. The code for this is shown in Listing 9.

With this code in place, we can run the DLL from the main entry point as shown in Listing 10.

Client Response
With the DLL ready to broadcast messages, let’s take a look at the client end, whose first task is to receive these messages. The message is sent to every window, so catching it is simple, as Listing 11 demonstrates.

We catch the message here in the main window procedure for our client application. _lParam holds the address in the contexts of our application where the communication buffer has been placed.

To test that your system is working, put a useless line of code above the break (such as “int jghjghj = 0;” and breakpoint the useless line of code. Run your client in debug mode in Microsoft “Visual Studio” and inject your DLL into any other process (as you recall, the DLL may be injected via any means). Shortly after injection the breakpoint should be hit, indicating that the communication system is up and running.

Like in the DLL, we want a class to handle communications with DLLs. This class is really very simple, as it mainly just needs a pointer to the communication buffer and an interface for working with that buffer. We will use the interface for every request we send to the DLL. For example, when we want to read the process memory of a DLL-infected process we will go through the communication class and it will handle all possible situations that can arise during the communication process, including the successful completion of the read operation and the failure of the operation. The start of the class is shown in Listing 12.

```
Listing 13. Verifying a connection and creating an object to manage it.
```

```
Listing 14. Creating a connection to the DLL from the client.
```

```
Listing 15. Hooking up the connection to the message handler in the client.
```

```
Listing 16. In the DLL we check for a reply from any potential clients.
```

```
Listing 17. The bold area shows our addition to the searching routine.
```

There will be many connections to DLLs in the final run, so we keep an array of these. A simple std::vector<CClien tConnection * > m_pmmConnections will do fine. Managing the array of client connections is left up to the reader; in our case we are simply using the above vector and a critical section. Once the communication class is made, it will be clear how to use it, and any number of methods will work fine for managing these objects.

The constructor of the object applies the communication response, which at this point just means setting the buffer type to HTB_COMMUNICATION_BUFFER::HTB_INITIALREPLY. The buffer may not be a valid memory location, so reading/writing to it may crash the client application. Rather than abort in the constructor, we make a static function that does this check and actually returns a pointer to a created object if the address is valid. See Listing 13.

With this static function helper, adding a communication object becomes easy. Listing 14 shows an example function using our own management system.

All that remains is to hook this up to the window message. See Listing 15.

Sealing The Deal
With the client now responding to initial contact from the DLL, it is up to the DLL to catch that reply and create a dedicated thread for communication between the DLL and the client. We modify the search thread to check for replies from the client. First, the function that actually checks for the reply (Listing 16).

The client works locally in its own address space, so we begin by copying the client’s reply buffer locally to the DLL. Once the buffer is local, the only check that needs to be made is on the buffer type.

```
Listing 18. The start of our DLL class to handle connections to clients.
```

54 H T B M A G A Z I N E J U L Y 2 0 1 0

J U L Y 2 0 1 0 H T B M A G A Z I N E 5 5
Once this helper function is in place, it becomes easy to check for replies in the main DLL thread, as demonstrated by Listing 17.

At the beginning of the article we mentioned creating a class to handle single connections from the DLL to the client software. The job of CreateLink() is to make such a class and run it on its own thread. The class, running on its own thread, loops indefinitely until the connection is broken, either because the DLL application closed or because the client closed. Each iteration of the loop makes one check on the remote communication buffer and if a request has been made by the client application it is filled.

The shell of the class is shown in Listing 18.

This handles the basic functionality of the class: Attaching to and detaching from a client process and constantly checking the client process for closing. Notice that when the connection is made, the class sets the types in the target process (the client application) to idle. This must be done or the DLL will try to connect to the client repeatedly through the same buffer, since the message in the client application would otherwise remain as a response to initial contact.

Next we add the logic for handling requests from the client application to which we are connected. One call to this function will perform a single request check and, if a request is found, will satisfy the request. Listing 19 shows this function.

Next add the function for handling requests from the client application to which we are connected. One call to this function will perform a single request check and, if a request is found, will satisfy the request. Listing 19 shows this function.

We begin by handling only the idle message, which is the only message possible at this point. This function is meant to be called repeatedly on its own thread. Next, we add the thread function itself, which is a park and function. This is one of the simplest functions and needs little explanation (Listing 20).

Finally, the job of the CreateLink() function is to create one of these objects and start it on its own thread. We create a nested structure for storing the class object and the handle to its running thread. See Listing 21.

Next we have 2 management routines for dealing with connections, one of which creates connections (CreateLink()), and one of which closes connections, as shown in Listing 22. Notice the addition to Stop().

WHAT IS HAPPENING

CreateLink() is already called when a response to initial contact is detected from the main DLL loop that searches for both open processes and replies to initial contact. Replies to initial contact are detected when the remote process (the client application) is remotely detected. In order to avoid re-establishing connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client. In order to avoid re-creating connections to the same client, the buffer in the client process is remotely changed by the DLL, setting the mType member to BITB_COMMUNICATION_BUFFER::{BITB_INITIAL_REPLY} to its own buffer. When this change is detected from the DLL, a new CTargetProcess object is created to handle all of the remaining communications with that client.

Finally, the DLL sends its new CTargetProcess object into a loop on its own thread that checks for and handles changes to the buffer in the remote client application. Each time it checks, it must copy the buffer locally. It modifies the client application indirectly, since the client application is not allowed to modify the DLL process in any way. The point of this communication network is to avoid methods of modifying the DLL process that might trigger anti-cheat protections.
The DLL and the client application are now in communication. All that remains is to decide what types of requests and can be made. We will only show 2 requests in this article: Reading and writing of the DLL process's RAM.

In order to add a new request of any kind, the HITB_COMMUNICATION_BUFFER structure must be updated. We add a new request type to the enumeration and add a new structure for the data specific to that request type. In Listing 23, we add both the ReadProcessMemory() and WriteProcessMemory() requests.

In order to processes these messages we update the Tick() function on the CTargetProcess class (Listing 24).

When the client is requesting a read of memory, the actual operation that needs to be done is to copy memory from the DLL process to the client process. From the perspective of the DLL process, this resolves to a WriteProcessMemory(). The implementation of a request from the client to write memory to the DLL. After each request is answered, the return data must be sent back to the client, overwriting the previous buffer. We only modify data related to the type of request we are fulfilling.

Every request causes the buffer in the remote client application to be reset back to the idle state. The code in Listing 25 is used in the client application to initiate a request.

Notice the addition of the HITB_COMMUNICATION_BUFFER::HITB_CLOSING buffer type. This tells us the request cannot be filled out due to the target process closing. Also note that it may be possible for our local buffer to become HITB_COMMUNICATION_BUFFER::HITB_CLOSING at any time, and the solution is solid.

CONCLUSION

With this code in place, the client can simply call the ReadProcessMemory() function on its own communication object to read the memory of any process on the PC at any time, while remaining truly silent—hidden from all current anti-cheat methods.

This method is several times slower than direct access to a process, but can crack even the toughest of protections, and runs entirely in ring-3 using very basic coding principles. Improvements can be made as well. The password sent between the DLL and the client should be randomized on a per-boot basis, and hard-coded into the DLL. That is, the client application can actually modify the DLL itself, changing the password inside the DLL before it is injected for the next go. This also changes the DLL communication buffer.

Finally, in order to enter the HITB_COMMUNICATION_BUFFER::HITB_CLOSING state and to clear up the only resource leak, we add a destructor to the CTargetProcess class in the DLL. See Listing 26.

When the client is closed from the DLL, we will no longer be able to reply to any requests from it, so the last message we send to it is HITB_COMMUNICATION_BUFFER::HITB_CLOSING. The destructor for this class happens only after both its monitoring thread and main-logic thread have completely stopped, so there is no risk of over-writing the buffer status in the middle of a pending request. The client application is coded to be aware that its buffer could be changed to HITB_COMMUNICATION_BUFFER::HITB_CLOSING at any time, and the solution is solid.

CONCLUSION

With this code in place, the client can simply call the ReadProcessMemory() function on its own communication object to read the memory of any process on the PC at any time, while remaining truly silent—hidden from all current anti-cheat methods.

This method is several times slower than direct access to a process, but can crack even the toughest of protections, and runs entirely in ring-3 using very basic coding principles. Improvements can be made as well. The password sent between the DLL and the client should be randomized on a per-boot basis, and hard-coded into the DLL. That is, the client application can actually modify the DLL itself, changing the password inside the DLL before it is injected for the next go. This also changes the DLL communication buffer.

Finally, in order to enter the HITB_COMMUNICATION_BUFFER::HITB_CLOSING state and to clear up the only resource leak, we add a destructor to the CTargetProcess class in the DLL. See Listing 26.

When the client is closed from the DLL, we will no longer be able to reply to any requests from it, so the last message we send to it is HITB_COMMUNICATION_BUFFER::HITB_CLOSING. The destructor for this class happens only after both its monitoring thread and main-logic thread have completely stopped, so there is no risk of over-writing the buffer status in the middle of a pending request. The client application is coded to be aware that its buffer could be changed to HITB_COMMUNICATION_BUFFER::HITB_CLOSING at any time, and the solution is solid.

CONCLUSION

With this code in place, the client can simply call the ReadProcessMemory() function on its own communication object to read the memory of any process on the PC at any time, while remaining truly silent—hidden from all current anti-cheat methods.

This method is several times slower than direct access to a process, but can crack even the toughest of protections, and runs entirely in ring-3 using very basic coding principles. Improvements can be made as well. The password sent between the DLL and the client should be randomized on a per-boot basis, and hard-coded into the DLL. That is, the client application can actually modify the DLL itself, changing the password inside the DLL before it is injected for the next go. This also changes the DLL communication buffer.
MD5/checksum. The DLL size can be randomized at every boot as well by appending random bytes to the end of the file. This does not corrupt the DLL. Note that there are no string literals in the DLL. Strings are an easy way for the anti-cheat to detect your DLL.

The DLL is ready for upgrade to kernel-mode as well. By overriding the methods in the CProcess class, ring-0 exchange of information from the DLL to the client becomes easy, removing the most likely method of detection.

Nearly all working cheats for protected games work by injecting a custom DLL into the game itself. This method extends upon this idea to bring the target process out into the open where it can be controlled remotely by existing software.

Listing 25. Initiating a request from the client to the DLL to read or write memory remotely.

```c
while ( m_lpcbBuffer->mType != HITB_COMMUNICATION_BUFFER::HITB_IDLE ) {
    if ( m_lpcbBuffer->mType == HITB_COMMUNICATION_BUFFER::HITB_CLOSING ) {
        ::LeaveCriticalSection( &m_csCrit );
        return FALSE;
    }
}
// Fill out our local buffer, changing the buffer type last.
   m_lpcbBuffer->u.wdWpm.lpvBaseAddress = _lpvBaseAddress;
   m_lpcbBuffer->u.wdWpm.lpvBuffer = _lpvBuffer;
   m_lpcbBuffer->u.wdWpm.stSize = _stSize;
   m_lpcbBuffer->u.wdWpm.~type = HITB_COMMUNICATION_BUFFER::HITB_WPM;

   ::InterlockedCompareExchangeAcquire( reinterpret_cast<LONG *>(&m_lpcbBuffer->mType),
                                    HITB_COMMUNICATION_BUFFER::HITB_WPM,
                                    HITB_COMMUNICATION_BUFFER::HITB_IDLE );

while ( m_lpcbBuffer->mType == HITB_COMMUNICATION_BUFFER::HITB_WPM ) {
}
// Request satisfied.
// Check the buffer type.
   if ( m_lpcbBuffer->mType == HITB_COMMUNICATION_BUFFER::HITB_IDLE ) {
       if ( _pstNumberOfBytesWritten ) {
           (*_pstNumberOfBytesWritten) = m_lpcbBuffer->u.wdWpm.stSize;
       }
       bRet = m_lpcbBuffer->u.wdWpm.bStatus;
    }
else {
   // All other buffer types are errors.
   if ( _pstNumberOfBytesWritten ) {
       (*_pstNumberOfBytesWritten) = 0UL;
   }
}
::LeaveCriticalSection( &m_csCrit );
return bRet;
}
```


```c
while ( m_ppRemoteFunction ) {
    if ( *m_ppProcess->readProcessMemory( m_hTarget, m_lpcbBuffer, &cbRemoteBuffer, sizeof( cbRemoteBuffer ) ) ) {
        cbRemoteBuffer.mType = HITB_COMMUNICATION_BUFFER::HITB_CLOSING;
        m_ppProcess->writeProcessMemory( m_hTarget, m_lpcbBuffer, &cbRemoteBuffer, sizeof( cbRemoteBuffer ) );
    }
}
```

MDS/checksum. The DLL size can be randomized at every boot as well by appending random bytes to the end of the file. This does not corrupt the DLL. Note that there are no string literals in the DLL. Strings are an easy way for the anti-cheat to detect your DLL.
In this article, I would like to focus on two methods of hooking – Virtual Method Table in DirectX. Both methods are similar and differ only in the first hook in the hook chain. The first method will start the hook chain with a classical and well known Import Address Table hooking (or IAT hooking) and the second one will use the DLL spoofing technique - replacing the original library (in this case - DInput.dll) with a fake one.

As an example, we will hook the GetDeviceState method from the IDirectInputDevice object which returns the mouse click information. This method is commonly hooked in game bots mainly for auto aiming purposes.

Let us start with discussing how a normal unhooked call chains to the GetDeviceState method in DInput.dll looks like. The first function in the call chain is DirectInputCreateA. When an application calls this function, it passes four parameters which are - application handle, version of DirectInput which the program relies on, output pointer for the IDirectInput interface structure (it is written to only if the call succeeds) and a pointer to an IUnknown object (most of the times it is NULL).

Next the method CreateDevice from IDirectInput is called. According to MSDN, this method takes three parameters but a macro-declaration in the dinput.h header appends a fourth; ppvOut-pointer - a pointer to the interface. The full declaration is shown on Listing 1 (Delphi syntax).

Listing 1. CreateDevice declaration

```delphi
function CreateDevice( ppvOut:pointer; const rguid:TGUID; var lplpDirectInputDevice:IDirectInputDeviceA; pUnkOuter:IUnknown):Hresult;stdcall;
```

If everything goes well, an object IDirectInputDevice will be created. It contains several methods including GetDeviceState which we would like to hook.

That is the normal call chain. To start with the IAT and VMT hooks, we need to know how the Import Address Table and Virtual Method Table structures. Let us start with the Import Address Table (IAT)

Most of the Win32 applications use functions from various DLL library files. To make it work properly, an application needs to know the address (in memory) of each imported function from each imported DLL library. For that reason, the Import Address Table is used (IAT). Every DLL library which is used by the application is listed in the array of IMAGE_IMPORT_DESCRIPTOR structures, the address (RVA) can be found in the IMAGE_DIRECTORY_ENTRY_IMPORT (defined as 1) entry in the DataDirectory array in the Opt-
When the operating system loads the application to the memory, it parses the content of the array of IMAGE_IMPORT_DESCRIPTOR and loads into memory the DLL libraries listed there (unless the DLL already exist in the memory). The loader then searches the address of every imported function in the Export Address Table of the given library and writes them the first thunk of the library in the IAT under the proper function address slot.

Virtual Method Table (VMT)
The VMT is mostly described as “Virtual Function Table”, “Dispatch Table” or “VTable”. Living up to its name, it is the mechanism behind dynamic dispatch of virtual methods.

For those who are interested in more specific description of VMT, I recommend reading part 1 and 2. Enough theory for now - let's do the practical work now.

The Hooks
The first method uses a classical approach - we create a DLL library when loaded (e.g. using the DLL injection or similar technique which was already described in HITB Ezine Issue 001 - The Art of DLL Injection by Christian Wojner) will overwrite the address of the original function in the IAT of the application (see Picture 1A) along with the address of our replacement hook function starting the chain of hooks.

When the application calls the hooked function, it will hook a method that initializes the device after creating IDirectInput (it is the second hook in the chain). After the hooked mechanism is called to initialize the device, it will hook another method - GetDeviceState, this time in the IDirectInputDevice object (last link of the chain. See Picture 1B).

In the first step, we must add two modules to our DLL - win32_pe, DirectX as shown in Listing 2. Next, write a function that will perform the first hooking. This function will acquire IAT address from the PE header and search out the address of the function IDirectInputCreateA. This address will be overwritten with the address of our replacement function (discussed in the next paragraph).

When the first hooking function is ready, it is time to prepare the IDirectInputCreateA replacement (call it mDirectInputCreateA as shown in Listing 3). We need it to be exported by our DLL library (it will come in handy later) therefore it is necessary to add it to the export table. Since the original function is stdcall type, we need to declare the replacement as stdcall.

We call the original function to get the Virtual Method Table. Next, we add to that pointer the value 12 (3*4) since CreateDevice is the third method declared from IDirectInputObject (see dinput.h). Save the address of the original function and overwrite it with the address of our replacement function mCreateDevice. The mCreateDevice (see Listing 4) is to be made as the same rules with the previous replacement function.

To start with, it is worth to see the declaration of this method in the “dinput.h” file. This function receives four parameters (not three as mentioned before). In the above case, we have to call the original method in order to receive the address of the next VMT table. The structure of this table is the same as the previous one therefore to get the address of that particular method - simply add 36 (9*4) to the VMT address. Save the address to a variable and replace it with the address of our next replacement function - mGetDeviceState, which is shown on Listing 3.

Similarly to CreateDevice, there is one parameter missing in the declaration of the function (see the macro in the “dinput.h” header file).

Finally, we have to complete our library with the declaration of all functions and add them to the table of exports, as shown on Listing 6. After compilation, we will receive a fully functioning DLL library which can be tested as an exemplary application available on12. When the application calls the GetDeviceState method, our hooks will capture the mouse click information which could be changed on the fly.

DII Spoofing
As far as the second method is concerned, it is easier in most cases because we do not have to hook IAT to the application. It is also helpful in bypassing certain issues concerning loading DLL in the proper time and lack of access to IAT.
The application is convinced that it uses the original library. In this case, we only have to export a function called DirectInput-CreateA. The rest of the DLL will be the same as in the previous method. We can copy functions; mCreateDevice, mGetDeviceState, hookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).

We will start the implementation of the rouge DLL with loading the original library and saving the address of the original function, as shown on Listing 7.

We have to create the declaration for all functions which are used by our test application. In this case, we only have to export a function called DirectInput-CreateA. The rest of the DLL will be the same as in the previous method. We can copy functions; mCreateDevice, mGetDeviceState, hookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).

We copy functions; mCreateDevice, mGetDeviceState, hhookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).

Since the DirectInput.dll is not a system library, the sequence of search is different - it should start with application directory. This will cause our fake library placed in the application directory to be searched quicker than the original one (see Picture 3).

We will implement the of the rouge DLL with loading the original library and saving the address of the original function, as shown on Listing 7.

We have to create the declaration for all functions which are used by our test application. In this case, we only have to export a function called DirectInput-CreateA. The rest of the DLL will be the same as in the previous method. We can copy functions; mCreateDevice, mGetDeviceState, hookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).

We will implement the of the rouge DLL with loading the original library and saving the address of the original function, as shown on Listing 7.

We have to create the declaration for all functions which are used by our test application. In this case, we only have to export a function called DirectInput-CreateA. The rest of the DLL will be the same as in the previous method. We can copy functions; mCreateDevice, mGetDeviceState, hookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).

We have to create the declaration for all functions which are used by our test application. In this case, we only have to export a function called DirectInput-CreateA. The rest of the DLL will be the same as in the previous method. We can copy functions; mCreateDevice, mGetDeviceState, hookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).

If an application imports a system DLL, it is searched in the following locations (sequence is important):

- System directory (C:/Windows/System32)
- Application directory
- Current process directory (if different than application directory)
- Windows directory (C:/Windows)
- Directory form environmental variable PATH

Since the DirectInput.dll is not a system library, the sequence of search is different - it should start with application directory. This will cause our fake library placed in the application directory to be searched quicker than the original one (see Picture 3).

We will implement the of the rouge DLL with loading the original library and saving the address of the original function, as shown on Listing 7.

We have to create the declaration for all functions which are used by our test application. In this case, we only have to export a function called DirectInput-CreateA. The rest of the DLL will be the same as in the previous method. We can copy functions; mCreateDevice, mGetDeviceState, hookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).

If an application imports a system DLL, it is searched in the following locations (sequence is important):

- System directory (C:/Windows/System32)
- Application directory
- Current process directory (if different than application directory)
- Windows directory (C:/Windows)
- Directory form environmental variable PATH

Since the DirectInput.dll is not a system library, the sequence of search is different - it should start with application directory. This will cause our fake library placed in the application directory to be searched quicker than the original one (see Picture 3).

We will implement the of the rouge DLL with loading the original library and saving the address of the original function, as shown on Listing 7.

We have to create the declaration for all functions which are used by our test application. In this case, we only have to export a function called DirectInput-CreateA. The rest of the DLL will be the same as in the previous method. We can copy functions; mCreateDevice, mGetDeviceState, hookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).

If an application imports a system DLL, it is searched in the following locations (sequence is important):

- System directory (C:/Windows/System32)
- Application directory
- Current process directory (if different than application directory)
- Windows directory (C:/Windows)
- Directory form environmental variable PATH

Since the DirectInput.dll is not a system library, the sequence of search is different - it should start with application directory. This will cause our fake library placed in the application directory to be searched quicker than the original one (see Picture 3).

We will implement the of the rouge DLL with loading the original library and saving the address of the original function, as shown on Listing 7.

We have to create the declaration for all functions which are used by our test application. In this case, we only have to export a function called DirectInput-CreateA. The rest of the DLL will be the same as in the previous method. We can copy functions; mCreateDevice, mGetDeviceState, hookcode and mDirectInputCreateA and have to make some modifications to the last one; DirectInputCreateA is the exported function; thus we have to remove prefix “m” in the replacement function (currently from the application directory to be searched quicker than the original one (see Picture 3)).
URL Shorteners Made My Day!

By Saumil Shah, Net-Square

Imagine yourself walking around in a shady part of town, looking for a place to eat. A guy comes up with a fake friendly smile, takes you to a run down building, opens a door and tells you that it is a shortcut to the best restaurant in town. You step in enthusiastically with glee and wonder. The digital equivalent of this scenario is clicking on something that says bit.ly/6ktven.
URL shorteners are here to stay. They have gone from being cool to being a downright necessity, thanks to services like Twitter. Posting shortened URLs is now the norm across all social networking sites. Over the past couple of years, society has come to trust these short creepy looking strings. Yet, no one seems to be bothered.

URL shorteners have many intrinsic design flaws. Part of the blame goes to the HTTP standard, which is in need of a serious overhaul. The rest of the blame lies with the design of many URL shortening services. URL shorteners were born out of necessity, as many other inventions and devices. However, they have been rolled out hastily. Hundreds of URL shortening services have mushroomed after seeing the success of an initial few. Some URL shorteners are a bit strict as to what they will allow to be shortened. But a vast majority simply don’t care. This article is a result of my musings with URL shorteners and pushing the envelope on how bad can things get.

First, let us see how URL shorteners work. All URL shorteners are based on HTTP redirects. HTTP’s response code 301 and 302 stand for “Resource Permanently Redirected” and “Resource Temporarily Redirected” respectively. When a browser receives an HTTP 301 or 302 response, it looks for the “Location” response header. Figure 1 shows how a typical HTTP 301/302 response looks like.

At the outset, this does not seem so terrifying. Bear in mind that HTTP redirects were thought up during a time when it required your own web server to issue 301 and 302 responses. If you want to trick someone, you had to run your own rogue web server. In the late 90s, that meant buying a hosted server which allowed you to configure the HTTP server any way you wanted. This meant having root level access on an Apache box. Today, you can get 301 and 302 redirects for free.

Let us explore some URL shorter abuse scenarios, beginning from the least sophisticated tricks to uber cool hacks.

**Sending your browser on a wild goose chase**

URL shorteners make it very easy to send browsers into redirection loops. The scenario is simple. Let URL A redirect to URL B which in turn redirects to URL A. Many URL shorteners allow the user to give unique names and keywords to shortened URLs. tinyurl.com and doiop.com are two URL shortening services that allow custom aliases to be assigned to shortened URLs. Interestingly, there are URL lengthening services such as hugeurl.com, which expand short URLs into insanely long URLs! I am sure the creator of hugeurl.com has made it purely for humour, but hugeurl.com serves an invaluable purpose for hiding our evil tracks!

We begin with hugeurl.com. Let us generate a huge URL for “http://doiop.com/ricknrolla”.

Now, we create a short URL for this huge URL on doiop.com, and assign it the alias “http://doiop.com/ricknrolla”. Figure 3 shows doiop.com shrinking the huge URL to “http://doiop.com/ricknrolla”. Now, all it takes is someone to land on http://doiop.com/ricknrolla. The browser enters a URL merry-go-round, and eventually gives up. Figures 4 and 5 show what happens to the browser.
Redirect Loop

Firefox has detected that the server is redirecting the request for this address in a way that will never complete.

The browser has stopped trying to retrieve the requested item. The site is redirecting the request in a way that will never complete.

- Have you disabled or blocked cookies required by this site?
- **NOTE**: If accepting the site’s cookies does not resolve the problem, it is likely a server configuration issue and not your computer.

![Try Again](image)

## Redirect Loop

Firefox has detected that the server is redirecting the request for this address in a way that will never complete.

The browser has stopped trying to retrieve the requested item. The site is redirecting the request in a way that will never complete.

- Have you disabled or blocked cookies required by this site?
- **NOTE**: If accepting the site’s cookies does not resolve the problem, it is likely a server configuration issue and not your computer.

![Try Again](image)
The first step was to convert my VLC exploit into a pure alphanumeric payload using Metasploit’s msfencode. Use msfencode’s BufferRegister=REG option to generate a pure alphanumeric shellcode if you know that register REG points to the payload. The other challenge lay in finding DLL jump addresses that were alphanumeric. After hours of playing with DLL addresses and egghunter shellcode, I arrived at the following alphanumeric payload to exploit VLC’s smb:// URI handling overflow, as shown in Figure 8.

Why do we need an alphanumeric payload? An alphanumeric smb:// URI can be easily shortened using a URL shortener! Simply copy and paste this string into a URL shortener of your choice. Figure 9 shows tinyurl.com shortening this URI.

To test this technique, start VLC and choose to open a network resource identified by an HTTP/HTTPS/FTP/MMS URL as show in Figure 10. Provide the shortened URL in the URL field. VLC will receive a redirect from the URL shortener and then proceed to open the smb:// URI as shown in Figure 11.

Sure enough the exploit succeeds, launching calc.exe which has now come to pass as the “Hello World” of all shellcode! Here we see that the entire exploit is hosted on the URL shortener. The attacker needs only a URL shortener to launch this exploit on victim’s browsers.

The final cherry on the icing comes from turning this VLC bug into a remote browser exploit. Use an OBJECT or an EMBED tag to automatically launch VLC as a browser plugin, supply the shortened URL as a target resource and watch the browser get owned! VLC installs a Firefox plugin when installed with default options. An example using the EMBED tag in Firefox is shown in Figure 13.

The final cherry on the icing comes from turning this VLC bug into a remote browser exploit. Use an OBJECT or an EMBED tag to automatically launch VLC as a browser plugin, supply the shortened URL as a target resource and watch the browser get owned! VLC installs a Firefox plugin when installed with default options. An example using the EMBED tag in Firefox is shown in Figure 13.

**Conclusion**

Have I made my point that URL shorteners are not healthy for the Internet?

**References**

CanSecWest 2010 Lightning Talk: http://slideshare.net/saumilshah/url-shorteners-made-my-day.

**About the Author**

Saumil Shah is a security researcher. He has been speaking and training at many conferences worldwide for over a decade. His recent interests are in combining old school web hacking techniques with browser exploits. He has written a few books, tools and papers. He has been running a specialized security services company, Net-Square, for the past 10 years. He likes to travel and take photographs. He doesn’t tweet and doesn’t facebook. And he hates being harassed. •

Figure 11. VLC following the redirect and proceeding to process the smb:// URI

Figure 12. VLC owned! calc.exe successfully launched

Figure 13. Using an EMBED tag to launch and exploit VLC

<embed type="application/x-vlc-plugin" width="320" height="200" target="http://tinyurl.com/ycctrzf" id="vlc" />

Figure 13. Using an EMBED tag to launch and exploit VLC
Book Review

ModSecurity Handbook
Ivan Ristic

Review by Gynvael Coldwind

I am sure a majority of the HITB Magazine readers are familiar with ModSecurity – we come across it during network security planning, maintaining and penetration tests. To make sure we are on the same page, ModSecurity is an open source Web Application Firewall, in form of an Apache HTTP Server module and it can work as an embedded WAF (inside the main web server itself). It can also work as well as a reverse proxy, shielding some other web server.

Before I get to the ModSecurity Handbook itself, let me briefly introduce the author - Ivan Ristic. Ivan is a programmer, a web security specialist, a writer and what is most important is he is one of the ModSecurity creators - so he knows his stuff. Thankfully, his internal knowledge of the module can be seen all through this book – we are provided with information of some ModSecurity internal mechanics, traps (both in CPU expensiveness and in maintaining difficulties) that awaits rule writers and the changes between versions. Some features described in the book are taken directly from the developers’ branch of the project.

Let us start from the beginning. This book is divided into two major parts – the User Guide providing bits of ModSecurity history, brief installation description, more detailed configuration section and a rule writing tutorial. You can also find detailed sections covering practical rule writing; performance and content injection; utilizing LUA scripting language in rules, as well as in-depth handling of XML based traffic or tips on writing ModSecurity extensions.

The second part of the book is basically a reference manual describing every command, variable, transformation function, action and operator which can be used while creating rules for ModSecurity. The output log formats are characterized in that part of the book which will come in handy if you are planning to write a log parser for a detection system.

The second part contains what a good reference manual should contain, a description of each item, information about the syntax, usually an example of usage, minimum version required (as I have mentioned before some features are yet to be available in the main release) and remarks about the behavior or possible usage of the command/operator & etc. Everything is clear being verbose enough to cover most important details and brief enough so one does not have to read ten pages to understand how to make use of a simple operator. This is definitely a must-have for rule designers.

As for the main section of the book - the User Guide; I must admit that before I got this book I only knew ModSecurity from the attacker’s perspective and I have never written rules for it. From my experience, this book can get you started as a novice while explaining some of the inner mechanics and get you to an advance level provided you read the User Guide section and write some rules on your own. The focus is placed on writing CPU-efficient rules; hence the knowledge gained is applicable even for demanding websites. Everything is well explained - written for humans (I really enjoyed reading the text between the examples, as opposed to some books) and the order of tasks is perfectly written. I especially like is that sometimes the author skips to a topic covered in another chapter, just to show how some rule or syntax looks like. It may seem a little chaotic but it is not as it really simplifies the learning process.

Let us focus on how the book looks like. The cover greets us with a cool looking ninja with crossed hands and a handle of a sword visible above his right shoulder (he is probably a left-handed ninja). In my opinion, the cover looks aesthetic and stylish.

I have come across few complains on the Internet as to the quality of the English in this book. I disagree with the fact that the English is poor as in my opinion; the English is fluent with no grammar or vocabulary mistakes.

The layout of the book is clear with the lines are spread enough to ease reading, the text and code fonts are easily distinguishable and with additional clearly marked “Notes” appearing here and there makes a point to reach to the readers.

The book is available in both printed edition (it is around 19 x 23.5 cm) with soft cover and in the electronic PDF format, designed for both printing and screen reading. Although the book is also available on Amazon, I have not seen a Kindle edition just yet.

It is important to note that the development of the book was not stopped after its release – the author is still working on it and the readers who bought the book can get an updated version on the Feisty Duck publisher’s website (If I recall correctly a free-update lasts for one year). If you have any remarks or requests regarding the book you can e-mail the author and the fixes might appear in future update.

Overall, I think ModSecurity Handbook is a well designed, nicely written and interesting book. I am glad to have a copy on my shelf. If you are interested in learning what a WAF is, how ModSecurity works, how to write efficient and advanced rules or just to polish your knowledge in these fields - then this book is a must-have for you.
In the world of lockpicking, Barry Wels stands out as a reflection of the hard work and dedication that turns enthusiasts into experts. Barry’s journey began when he was a teenager, and his fascination with locks continued into adulthood. His interest in physical security and his passion for lockpicking have led him to become a renowned lockpicker and the founder of TOOOL, an organization dedicated to the education of people on how locks work and why they are designed the way they are.

What inspired Barry to share his knowledge in the area of lockpicking? Barry’s goal has always been to impart knowledge to people and show them the weaknesses of these locks and how they could be defeated. His wife, who was a member of TOOOL, inspired him to share his knowledge by bringing him to a TOOOL gathering, where he saw the passion and dedication of the group. This experience sparked his interest in sharing his knowledge with others.

During his interview, Barry shares his perspective on the differences between computer security and physical security. In computer security, most of the flaws can be fixed through a patch or by updating the software with the latest version. However, in physical security, like mechanical locks, it is more difficult and costly to fix these flaws. The protection mechanism, while it may seem simple in theory, can be complex in practice, which is why Barry believes that learning things the hard way is sometimes good for us.

Barry also reflects on the importance of the TOOOL community in his development as a lockpicker. He shares that the group has taught him a lot, and he is grateful for the opportunities it has given him. The TOOOL community is not just a group of lockpickers; it is a community of like-minded individuals who share a common interest in physical security. They work together to educate others on the importance of understanding how locks work and how they can be defeated.

In conclusion, Barry Wels’ interview offers a glimpse into the world of lockpicking and the passion it can inspire. His journey from a curious teenager to a respected lockpicker and founder of TOOOL is a testament to the power of dedication and hard work. As he continues to teach others about locks and how they can be defeated, Barry’s contributions to the TOOOL community will undoubtedly continue to have a positive impact on the field of physical security.
INTERVIEW

If you want to pick a lock, you have to follow the three O rule of TOOOL, you have to practice Over and Over and Over again.

Do you guys work closely with law enforcement agencies or if they have ever asked you guys to help them to solve cases?

There are number of times that we were called to assist with forensic investigation and become an expert witness in the court. One of the most common questions we normally get in the court is something like this, "Is it possible to open the lock without doing any damage to it?".

How many members do you have for Amsterdam chapter and how many times a week do you guys meet?

We have about 100 members here in Amsterdam alone and we meet once every 2 weeks.

Other than the weekly gatherings, do you guys organize any other events?

Yes, of course. We are the organizer of LockCon, the lockpicking equivalent of HITB Conference. People come here to present new materials related to lockpicking. In fact, some of the materials are only available at our conference. Other than that, this is where our Lockpicking Championship is being held and you will be able to witness how people open safes and locks at record speed.

For those people who are interested in learning lockpicking and are not able to attend any of the gatherings, do you guys provide the materials online?

Yes, Many of the videos including demonstrations and animations created by our members are available online for free. Kindly visit Waag Society website (http://connect.waag.org/toool) for some of the videos.

Can anyone run a TOOOL chapter in their respective country?

Not really. At TOOOL, quality is more important than quantity. For that reason, we are very careful in approving our members and chapters. The process normally requires us meeting the applicant in person for an interview. This is very important as we want to avoid any weirdo from making stupid statements that will tarnish our image.

One final question, what is probably the most important thing in becoming a lockpicker?

Patience. In fact, we have a motto in our organization that goes like this, "If you want to pick a lock, you have to follow the three O rule of TOOOL, you have to practice Over and Over and Over again."

Thank you Barry.

You’re welcome.