TDDC03 Projects 2006

Examining a public exploit

Magnus Florén magfl705   Anna Vapen annva871

Supervisor: Claudiu Duma

Examining a Public Exploit

Magnus Florén Anna Vapen
Linköpings universitetet, Sweden
Email: {magfl705, annva871}@student.liu.se

Abstract

A public exploit is code that has been made for attacking computer systems and that is made public on the Internet. This attack code can be used for committing computer crimes by people that don't have enough knowledge to write their own code. On the other hand, public exploits can be a good source of knowledge for those who are trying to build secure systems and protect against intrusions. In this paper testing and analysis of public exploits is done to show how this works. Our aid to accomplish this is a guide by Don Parker [1] that we used in parts of the analysis.

1. Introduction

Our task is to examine a public exploit following the guidelines defined by Don Parker [1] and document experiences and facts. This includes attacking a Windows 2000 Pro machine from a Linux machine by using the RPC DCOM exploit. By using an IDS (Intrusion Detection System) and several other tools we are able to see how the exploit works by analysing its behavior. To help us doing this we are using a step by step guide written by Don Parker [1] [2]. In the guide the author suggests that we use the RPC DCOM exploit that uses a well known bug in Windows. The exact version of the exploit that Parker used is no longer available online. Instead we found newer versions of the RPC DCOM exploit and tried to modify them to fit our system. We also tried to apply Don Parker's guide [1] with another public exploit called the ShixxNote exploit. We did a Windows to Windows attack with the RPC DCOM exploit and a Windows to Linux attack with the ShixxNote exploit. The ShixxNote attack is not as well spread as the RPC DCOM attack since the ShixxNote attack uses a flaw in the not so common digital post-it-notes software ShixxNote. The ShixxNote exploit was rewritten so that it could run from Gentoo Linux attacking Windows 2000 Pro, see figure 2.

By studying exploits and their behavior it is possible to learn how to better protect a computer. In our project we wanted to show how easy these public exploits are to use, what harm they can do and what we can learn from them.

We also wanted to show what tools a system administrator can use to detect intrusion attempts.

The goals of this paper are: how do the RPC DCOM exploit and the ShixxNote exploit work and how do they exploit the vulnerabilities in Windows 2000 Pro and ShixxNote? How does a public exploit work and how can it be analysed? We will also evaluate how the test bench suggested by Don Parker works.

2. Background

In this section we explain the terms used in the report.

2.1 What is a vulnerability?

"When someone breaks into a computer system, that person takes advantage of lapses in procedures, technology, or management, allowing unauthorized access or actions. The specific failure of control is called a vulnerability" [10]. Vulnerabilities often results from bugs caused by careless programmers or design flaws in the system [11]. An example of a vulnerability is unchecked buffers, these can be used for buffer overflows (described below).

2.2 What is a buffer overflow?

A buffer is a data area of memory shared by different hardware devices or program processes to temporarily hold data. The data can be output or input from devices outside the computer or processes within a computer.

These buffers can be exploited if there is no check that ensures that the data written to the buffer is within the size of the buffer. If no check exists, data can overflow the buffer resulting in overwriting memory adjacent to the buffer. This can happen accidentally or with the purpose to do that. A buffer overflow is often used to overwrite the buffer until reaching the return address of a stack frame. The return address points to where the CPU shall execute code from when returning from a function call. When reaching the return address the attacker puts his/her own return address where attack code is residing and that will eventually be executed when the new return address is used. In figure 1 an example is given when a stack frame is put on the stack [12]. A stack is a temporary data structure. Usually it temporary holds functions and subroutines in stack frames. The stack frames contains the return address, local variables and parameters. By overflowing the local buffer with A until reaching the return address the attacker puts 0xffffffff as the return address. This results in a jump to an area where the attacker has placed malicious code, like in our cases, somewhere in the overflowed buffer.

Local buffer	AAAAAAAAAAA AAAAAAAAAAA
Return address	0xffffffff
Parameters	Parameters

Figure 1. Buffer overflow

2.3 What is an exploit?

An exploit (or exploitation program) is a kind of malicious logic that is used to exploit vulnerabilities in a computer system. There are many exploits that are made public by their creators to show how to exploit common bugs in software and also show why it is important to fix the problems. By analysing exploits in the way that we have done in this project it is possible to learn about how security flaws can be exploited. By knowing that, it is also possible to learn how to write better software and how to protect computer systems from being exploited.

Exploits can be harmful in many ways and some are worse than others. Where some exploits can make a computer crash others can give the attacker full administrator rights on the compromised system. That means the attacker can run arbitrary code, remove all files and so on [5]. The exploits that we analysed use buffer overflows to gain full rights on the Windows machine and that makes them very harmful.

2.4 RPC DCOM and ShixxNote exploit

The RPC DCOM exploit was made public in 2003 and used vulnerabilities in unpatched versions of the Microsoft Windows operating system. There are still many machines running this system without patches and firewalls, that is why it is interesting to see what harm a public exploit can do and how it really works. The original RPC DCOM exploit only works on Windows 2000 Pro without any service packs. However, the versions we found also work on newer versions of Windows 2000 and XP. The RPC DCOM exploit exploits a service called the RPC. It stands for Remote Procedure Call and is a protocol for inter-process communication, which allows a system to execute code on a remote system [8]. When this exploit was made public other security professionals improved the code and made it so easy that everyone capable of writing *dcom.exe “WindowsVersion” “targetIP”* in a command shell could get administrator rights on any computer running unpatched Windows 2000 or XP. The amount of people exploiting the flaw was so big that it was like a worm being spread.

The ShixxNote exploit is equally harmful, but only to machines running the ShixxNote software. We choose to analyze the ShixxNote exploit too because it works in a similar way and we wanted to try Don Parkers test bench with another exploit than the RCP DCOM exploit.

3. Method of work

Our method of work followed Don Parker’s guide [1], but there were some exceptions. We used three computers, see figure 2, as Parker suggested. The computers where the exploit code executed ran with Linux, in our case Gentoo that fits our hardware, a Sun Ultra1, better than SuSE used in Parkers guide [1], and Windows XP. The attacked computer ran with Windows 2000 Pro.

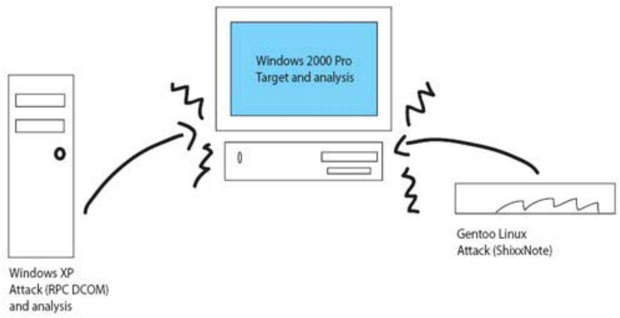


Figure 2. The lab environment.

The method of work which Don Parker’s guide suggested is these following steps:

1. Preparation

The preparation part included installation and learning to work with the following software:

- Windows 2000 Pro
- Snort
- Snortsnarf
- Windump
- Winpcap
- ActivePerl v5.6.1
- JulianDay.pm,TimeZone.pm, ParseDate.pm for Snortsnarf to work.

2. Execute the exploits
3. Analyse the exploits

Here we used the software that we installed in the preparation phase to analyse both exploits.

In section 4 Experiments, we have documented how our work actually was carried out.

4. Experiments

In this section we present our experiments with two attacks using the RPC DCOM exploit and the ShixxNote exploit. We also present the results of the analysis of the attacks.

4.1 The preparation phase

In the guide it was suggested to use Windows 2000 Pro without any service packs, but that version was hard to find. Instead we used Windows 2000 Pro with service pack 4. We realized that there are many exploits that works with SP4 so we installed it on a virtual machine to try if the RPC DCOM exploit would work at once. It didn’t so we tried out a few other versions of the same exploit, but they didn’t work either, more on that in the next phase. Later on we used a dedicated PC only for Windows 2000 Pro without any service packs on which we installed the other required software such as Windump, Winpcap, ActivePerl, Snort and Snortsnarf.

We ran into some problems getting Snortsnarf to work, the reason that it didn’t work was because we installed a newer version of both Snortsnarf and ActivePerl than Don Parker used in the guide [1]. When we installed the older versions it eventually worked.

4.2 The execution phase

When we entered this second phase we had already made some tries to get the RPC DCOM exploit to work, but they were unsuccessful. Before execution the exploits were compiled on the Linux machine, and then executed from that machine. The version of the RPC DCOM exploit that was linked from the guide [1] was no longer available online, but instead we found newer versions that was said to work with all service packs. Although that exploit didn't run on our system a porting of the ShixxNote exploit that works in a way similar to the RPC DCOM exploit gave us remote access to the Windows machine after a buffer overflow.

We tried to change the return addresses in the RPC DCOM exploit a couple of times, but every execution failed. All these tries were made from the Linux machine.

Later on we got another new version of the RPC DCOM exploit, which we compiled on a Windows XP laptop. After the compilation we tried to attack the Windows 2000 machine from the laptop and this time it was successful.

4.3 Analysis of logs

The tools for the analysis turned out to work much better on Windows XP than on Windows 2000 so we used a laptop with XP for the analysis, we also installed Windows XP and the necessary analysing tools on a dedicated PC. Windump was used to record the exploit in action and save the recorded data into a binary file. This file was filtered (or “washed”, to use Don Parker´s term) through Snort to make Snort recognize what the exploit in action looked like.

First we listed the interfaces that the Windows machine listened on:
windump.exe -W

We found that interface no 2 was the right one so we used that as the interface for Windump and also wrote the result to a file:
windump.exe -w filename -i 2

After that we could read the file through Snort (the so called washing):
snort.exe -r filename -i 2 -c rules.conf -A full

Here we also used a ruleset file available from the Snort homepage and enabled full logging. However it didn't give any alerts. Instead we used a different version of snort.conf available from [4]. From that site we also downloaded a bigger set of snort rules (bleeding-all.rules) and now found three alerts from the RPC DCOM exploit and two from the ShixxNote exploit. We now had the two alert files ready to run through Snortsnarf. When we examined the alerts closer we saw that the alerts was related to suspicious UDP packets and had nothing to do with the exploits.

This was not the result that Don Parker [1] got. He got five alerts where some of the alerts were false positives.

We did not use the same rules that Don Parker used because he stated that you should use the standard rules that comes with Snort, the problem is that the few rules that came with the installation did not give any alerts. Therefore we have tried with different rules set, but none resulted in the same alerts that Parker got.

Another reason why our result was different could be that we didn’t use exactly the same exploit. We used a newer version of the RPC DCOM exploit together with the ShixxNote exploit. A lesson learned with Snort is that it’s not easy to know what rules to use. We installed Snortsnarf, a tool that Don Parker used to show the alerts in a more readable way and it worked well. We knew that the alerts we had wasn’t useful, but we wanted to try out Snortsnarf as in Parkers guide.

In Parkers [1] guide he recommended that we shouldn’t use Ethereal [9] to do the analysing, but we got curious about this program, so we decided to try it out and see what result we would get.

Ethereal is a network protocol analyser. We installed Ethereal on both XP machines. This program has a graphical user interface and is very easy to use to analyse network traffic. The reason Parker didn’t recommend the program was that he thought people would learn more using the other software.

4.4 Analysis of the RPC DCOM log

What you can see in the logs is a typical behavior of a special exploit and this can be useful for a system administrator that wants to see suspicious behavior. In the case with the RPC DCOM exploit we can see that it reaches port 135 where the RPC service is running. We can also see lots of malformed TCP packets, which is typical for the exploit. In the end of the exploit running we also saw that port 4444 was open and the attacker can communicate through that port.

The screenshot in appendix A shows the packets that Windump captured, while the RPC DCOM exploit was running, as they are shown in Ethereal. The interesting parts are the black packets. They are malformed TCP packets sent not from the attacker, but from the target which can seem odd, but they are answers to packets that the attacker (using the IP 10.0.0.6) uses to overflow the target (IP 10.0.0.4). The violet packets are bind calls where the attacker tries to bind to the RPC service. As a system administrator it is easy to detect this exploit with Ethereal (or Snort, if a good ruleset is used). The traces are malformed SYN-ACK calls from the target and bind calls from an attacking host to port 135 on the target followed by traffic on an unknown port where you usually don't run anything. Ethereal can even show the commands that are sent to this port.

4.5 Analysis of the ShixxNote log

The behavior of the ShixxNote exploit is similar with the malformed TCP packets, but the ports are different. ShixxNote is running on port 2000 and opens a port on 101. In the ShixxNote case you can see the same pattern with malformed TCP packets. In Appendix B the Ethereal log is shown. The red reset call is where the attacker failed and tried again. The difference between this log and the other one is that the RPC port isn't used here, so no RPC bind calls are needed. Otherwise this looks like an ordinary remote buffer overflow, easy to recognize with Ethereal.

5. Results

How well did the step by step guide work, and how useful was it? One of the goals with this project was to evaluate Don Parker's guide [1] to see what use you can have of an online description of how to run and analyze a public exploit. There are many guides about hacking and cracking, but not as many about analysing attacks. Since this guide could be good for people working with security we wanted to know how useful it really is. The guide is from 2003 and of course things have changed since then. First of all, the exploit itself was no longer located at the homepage linked to from the guide. The site that had published this exploit and similar is now only open for people paying for their services (virus alerts, an exploit archive and similar things that can be useful for those who work with security). Although, famous exploits will always be out there on forums, mailing list archives and sites for all kinds of people that are interested about security. The guide did not say anything about how to get the exploit up and running, or how to get the tools working. By trial and error we found out which version of ActivePerl to install to get Snortsnarf working, where to place the time-libraries for Snortsnarf and how to list the interfaces to see which one to listen to. Except of that the guide was useful when it came to recommend tools such as Windump and Snort. It was easy to listen to the network traffic with these tools and see how the attack code implanted a buffer overflow from one port and then opened a shell on another.

The guide also showed how to detect false positives from the output from Snortsnarf. Since our network is not connected to the Internet the risk of false positives is not big. A problem with the guide was that the first part did contain information and tips about how to set up the network and do the analysis in the best way, but this information was found in the last chapter of the second part of the guide. It would have been much more helpful to put that information in the beginning.

6. Conclusions

We were going to see how the RPC DCOM exploit worked and how it exploited the RPC flaw in Windows 2000 Pro. We studied the code before compiling it and because of that we already knew what to expect from the exploit. The analysis with Ethereal showed that what we had learned from studying the code was true, that this exploit uses a buffer overflow in the RPC service on port 135 to open a command shell on another port. The logs showed clearly what kind of traffic the exploit generated and on which ports. Malformed SYN-ACK:s from the attacking machine followed by bind calls to the RPC port and after that another port opening is clear traces of this exploit in action. What a system administrator can do to protect against this kind of exploits is to patch Windows. Another good idea is to use a firewall and block unknown ports like 4444.

We were also going to evaluate Don Parker's test bench [1]. The idea of using a test bench made of an IDS (like Snortsnarf or Ethereal) and several other helpful tools to see how an exploit works is good. It gives the tester knowledge about how an exploit affects a computer and by knowing that it is easier to protect your system. By knowing how an exploit works it is also possible to recognize such an attack. We tested the test bench with two similar exploits and it seems like remote buffer overflows are quite easy to detect according to their special behaviour.

The test bench we used was similar to the one in the guide, but instead of using Snort and Snortsnarf we used Ethereal. The problem with this type of guides is that they soon become dated, and this one is three years old. In the end we didn't follow Parker's guide all the way, but the guide was very useful to us. We learned how to work with Windump, Snort and Snortsnarf. Besides of that we also learned how to do a log analysis with

Ethereal and we showed that our own version of Don Parker's guide works also with the ShixxNote exploit.

7. References

[1] Don Parker, *Examining a Public Exploit, Part 1*, http://www.securityfocus.com/infocus/1795

[2] Don Parker, *Examining a Public Exploit, Part 2*, http://www.securityfocus.com/infocus/1801

[3] *Snort – the defacto standard for intrusion detection/prevention*, http://www.snort.org

[4] *The Bleeding Edge of Snort – Open Snort signatures*, http://www.bleedingsnort.com

[5] Stuart McClure, Joel Scambray, George Kurtz, *Hacking i Fokus (Hacking Exposed, third edition)*, ISBN 91-636-0707-7

[6] Flashsky and Benjurry, *The Analysis of LSD's Buffer Overrun in Windows RPC Interface*, http://www.xfocus.org/documents/200307/2.html

[7] *Milw0rm*, http://www.milw0rm.com

[8] Matti Aharoni, *Windows DCOM RPC Exploit*, http://securitypronews.com/securitypronews-24-20030814WindowsDCOMRPCExploit.html

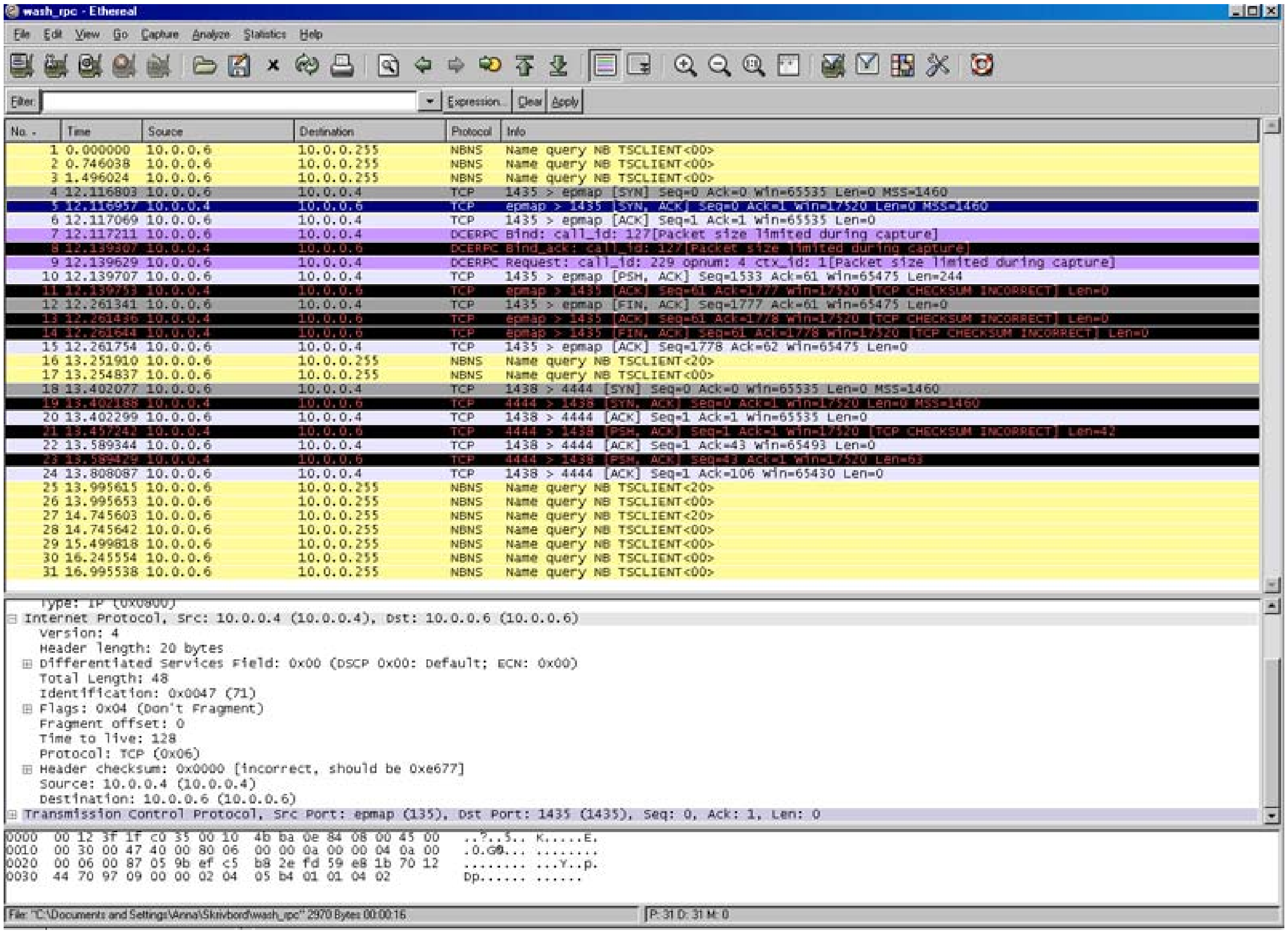
[9] Ethereal: A Network Protocol Analyzer http://www.ethereal.com

[10] Matt Bishop, Introduction to COMPUTER SECURITY, page 389, ISBN 0-321-24744-2

[11] Vulnerability http://en.wikipedia.org/wiki/Vulnerability_(computer_science)

[12] Wilander, Kamkar, *A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention*

8. Appendix A– Ethereal log of the RPC DCOM exploit


wash_rpc - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

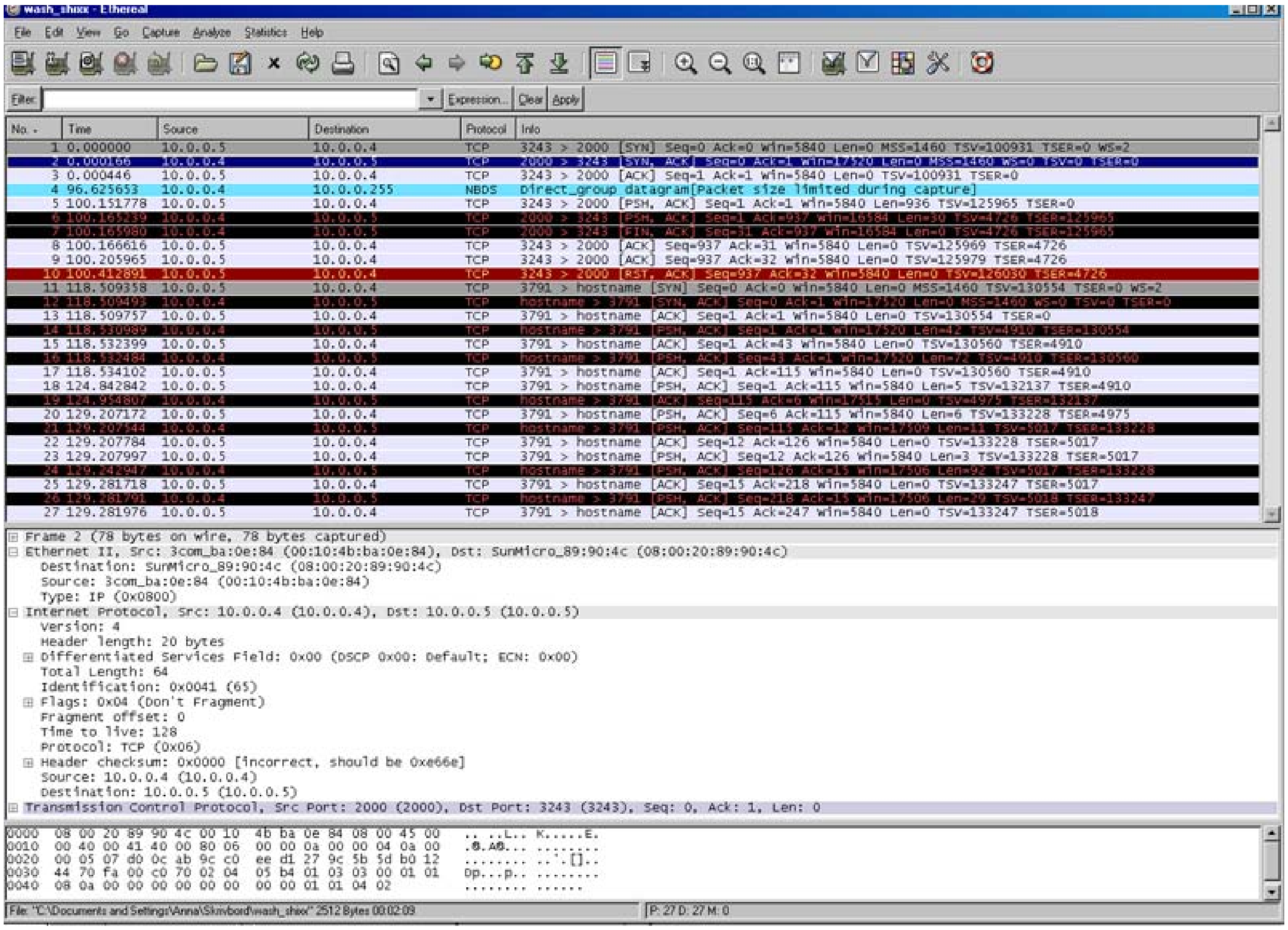
```
No. -  Time        Source      Destination   Protocol Info
     1 0.000000    10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
     2 0.746038    10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
     3 1.496024    10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
     4 12.116803   10.0.0.6    10.0.0.4      TCP      1435 > epmap [SYN] Seq=0 Ack=0 Win=65535 Len=0 MSS=1460
     5 12.116957   10.0.0.4    10.0.0.6      TCP      epmap > 1435 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460
     6 12.117069   10.0.0.6    10.0.0.4      TCP      1435 > epmap [ACK] Seq=1 Ack=1 Win=65535 Len=0
     7 12.117211   10.0.0.6    10.0.0.4      DCERPC   Bind: call_id: 127[Packet size limited during capture]
     8 12.139307   10.0.0.4    10.0.0.6      DCERPC   Bind_ack: call_id: 127[Packet size limited during capture]
     9 12.139629   10.0.0.6    10.0.0.4      DCERPC   Request: call_id: 229 opnum: 4 ctx_id: 1[Packet size limited during capture]
    10 12.139707   10.0.0.6    10.0.0.4      TCP      1435 > epmap [PSH, ACK] Seq=1533 Ack=61 Win=65475 Len=244
    11 12.139753   10.0.0.4    10.0.0.6      TCP      epmap > 1435 [ACK] Seq=61 Ack=1777 Win=17520 [TCP CHECKSUM INCORRECT] Len=0
    12 12.261341   10.0.0.6    10.0.0.4      TCP      1435 > epmap [FIN, ACK] Seq=1777 Ack=61 Win=65475 Len=0
    13 12.261436   10.0.0.4    10.0.0.6      TCP      epmap > 1435 [ACK] Seq=61 Ack=1778 Win=17520 [TCP CHECKSUM INCORRECT] Len=0
    14 12.261644   10.0.0.4    10.0.0.6      TCP      epmap > 1435 [FIN, ACK] Seq=61 Ack=1778 Win=17520 [TCP CHECKSUM INCORRECT] Len=0
    15 12.261754   10.0.0.6    10.0.0.4      TCP      1435 > epmap [ACK] Seq=1778 Ack=62 Win=65475 Len=0
    16 13.251910   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<20>
    17 13.254837   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
    18 13.402077   10.0.0.6    10.0.0.4      TCP      1438 > 4444 [SYN] Seq=0 Ack=0 Win=65535 Len=0 MSS=1460
    19 13.402188   10.0.0.4    10.0.0.6      TCP      4444 > 1438 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460
    20 13.402299   10.0.0.6    10.0.0.4      TCP      1438 > 4444 [ACK] Seq=1 Ack=1 Win=65535 Len=0
    21 13.457242   10.0.0.4    10.0.0.6      TCP      4444 > 1438 [PSH, ACK] Seq=1 Ack=1 Win=17520 [TCP CHECKSUM INCORRECT] Len=42
    22 13.589344   10.0.0.6    10.0.0.4      TCP      1438 > 4444 [ACK] Seq=1 Ack=43 Win=65493 Len=0
    23 13.589429   10.0.0.4    10.0.0.6      TCP      4444 > 1438 [PSH, ACK] Seq=43 Ack=1 Win=17520 Len=63
    24 13.808087   10.0.0.6    10.0.0.4      TCP      1438 > 4444 [ACK] Seq=1 Ack=106 Win=65430 Len=0
    25 13.995615   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<20>
    26 13.995653   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
    27 14.745603   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<20>
    28 14.745642   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
    29 15.499818   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
    30 16.245554   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
    31 16.995538   10.0.0.6    10.0.0.255    NBNS     Name query NB TSCLIENT<00>
```

```
    Type: IP (0x0800)
⊟ Internet Protocol, Src: 10.0.0.4 (10.0.0.4), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 48
    Identification: 0x0047 (71)
  ⊞ Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
  ⊞ Header checksum: 0x0000 [incorrect, should be 0xe677]
    Source: 10.0.0.4 (10.0.0.4)
    Destination: 10.0.0.6 (10.0.0.6)
⊞ Transmission Control Protocol, Src Port: epmap (135), Dst Port: 1435 (1435), Seq: 0, Ack: 1, Len: 0
```

```
0000  00 12 3f 1f c0 35 00 10  4b ba 0e 84 08 00 45 00   ..?..5.. K.....E.
0010  00 30 00 47 40 00 80 06  00 00 0a 00 00 04 0a 00   .0.G@... ........
0020  00 06 00 87 05 9b ef c5  b8 2e fd 59 e8 1b 70 12   ........ ...Y..p.
0030  44 70 97 09 00 00 02 04  05 b4 01 01 04 02         Dp...... ......
```

```
File: "C:\Documents and Settings\Anna\Skrivbord\wash_rpc" 2970 Bytes 00:00:16    P: 31 D: 31 M: 0
```

9. Appendix B – Ethereal log of the ShixxNote exploit


wash_shixx - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .  Time        Source      Destination  Protocol  Info
    1 0.000000    10.0.0.5    10.0.0.4     TCP   3243 > 2000 [SYN] Seq=0 Ack=0 Win=5840 Len=0 MSS=1460 TSV=100931 TSER=0 WS=2
    2 0.000166    10.0.0.4    10.0.0.5     TCP   2000 > 3243 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
    3 0.000446    10.0.0.5    10.0.0.4     TCP   3243 > 2000 [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSV=100931 TSER=0
    4 96.625653   10.0.0.4    10.0.0.255   NBDS  Direct_group datagram[Packet size limited during capture]
    5 100.151778  10.0.0.5    10.0.0.4     TCP   3243 > 2000 [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=936 TSV=125965 TSER=0
    6 100.165239  10.0.0.4    10.0.0.5     TCP   2000 > 3243 [PSH, ACK] Seq=1 Ack=937 Win=16584 Len=30 TSV=4726 TSER=125965
    7 100.165980  10.0.0.4    10.0.0.5     TCP   2000 > 3243 [FIN, ACK] Seq=31 Ack=937 Win=16584 Len=0 TSV=4726 TSER=125965
    8 100.166616  10.0.0.5    10.0.0.4     TCP   3243 > 2000 [ACK] Seq=937 Ack=31 Win=5840 Len=0 TSV=125969 TSER=4726
    9 100.205965  10.0.0.5    10.0.0.4     TCP   3243 > 2000 [ACK] Seq=937 Ack=32 Win=5840 Len=0 TSV=125979 TSER=4726
   10 100.412891  10.0.0.5    10.0.0.4     TCP   3243 > 2000 [RST, ACK] Seq=937 Ack=32 Win=5840 Len=0 TSV=126030 TSER=4726
   11 118.509358  10.0.0.5    10.0.0.4     TCP   3791 > hostname [SYN] Seq=0 Ack=0 Win=5840 Len=0 MSS=1460 TSV=130554 TSER=0 WS=2
   12 118.509493  10.0.0.4    10.0.0.5     TCP   hostname > 3791 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
   13 118.509757  10.0.0.5    10.0.0.4     TCP   3791 > hostname [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSV=130554 TSER=0
   14 118.530989  10.0.0.4    10.0.0.5     TCP   hostname > 3791 [PSH, ACK] Seq=1 Ack=1 Win=17520 Len=42 TSV=4910 TSER=130554
   15 118.532399  10.0.0.5    10.0.0.4     TCP   3791 > hostname [ACK] Seq=1 Ack=43 Win=5840 Len=0 TSV=130560 TSER=4910
   16 118.532484  10.0.0.4    10.0.0.5     TCP   hostname > 3791 [PSH, ACK] Seq=43 Ack=1 Win=17520 Len=72 TSV=4910 TSER=130560
   17 118.534102  10.0.0.5    10.0.0.4     TCP   3791 > hostname [ACK] Seq=1 Ack=115 Win=5840 Len=0 TSV=130560 TSER=4910
   18 124.842842  10.0.0.5    10.0.0.4     TCP   3791 > hostname [PSH, ACK] Seq=1 Ack=115 Win=5840 Len=5 TSV=132137 TSER=4910
   19 124.954807  10.0.0.4    10.0.0.5     TCP   hostname > 3791 [ACK] Seq=115 Ack=6 Win=17515 Len=0 TSV=4975 TSER=132137
   20 129.207172  10.0.0.5    10.0.0.4     TCP   3791 > hostname [PSH, ACK] Seq=6 Ack=115 Win=5840 Len=6 TSV=133228 TSER=4975
   21 129.207544  10.0.0.4    10.0.0.5     TCP   hostname > 3791 [PSH, ACK] Seq=115 Ack=12 Win=17509 Len=11 TSV=5017 TSER=133228
   22 129.207784  10.0.0.5    10.0.0.4     TCP   3791 > hostname [ACK] Seq=12 Ack=126 Win=5840 Len=0 TSV=133228 TSER=5017
   23 129.207997  10.0.0.5    10.0.0.4     TCP   3791 > hostname [PSH, ACK] Seq=12 Ack=126 Win=5840 Len=3 TSV=133228 TSER=5017
   24 129.242947  10.0.0.4    10.0.0.5     TCP   hostname > 3791 [PSH, ACK] Seq=126 Ack=15 Win=17506 Len=92 TSV=5017 TSER=133228
   25 129.281718  10.0.0.5    10.0.0.4     TCP   3791 > hostname [ACK] Seq=15 Ack=218 Win=5840 Len=0 TSV=133247 TSER=5017
   26 129.281791  10.0.0.4    10.0.0.5     TCP   hostname > 3791 [PSH, ACK] Seq=218 Ack=15 Win=17506 Len=29 TSV=5018 TSER=133247
   27 129.281976  10.0.0.5    10.0.0.4     TCP   3791 > hostname [ACK] Seq=15 Ack=247 Win=5840 Len=0 TSV=133247 TSER=5018
```

```
⊞ Frame 2 (78 bytes on wire, 78 bytes captured)
⊟ Ethernet II, Src: 3com_ba:0e:84 (00:10:4b:ba:0e:84), Dst: SunMicro_89:90:4c (08:00:20:89:90:4c)
    Destination: SunMicro_89:90:4c (08:00:20:89:90:4c)
    Source: 3com_ba:0e:84 (00:10:4b:ba:0e:84)
    Type: IP (0x0800)
⊟ Internet Protocol, Src: 10.0.0.4 (10.0.0.4), Dst: 10.0.0.5 (10.0.0.5)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 64
    Identification: 0x0041 (65)
  ⊞ Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
  ⊞ Header checksum: 0x0000 [incorrect, should be 0xe66e]
    Source: 10.0.0.4 (10.0.0.4)
    Destination: 10.0.0.5 (10.0.0.5)
⊞ Transmission Control Protocol, Src Port: 2000 (2000), Dst Port: 3243 (3243), Seq: 0, Ack: 1, Len: 0
```

```
0000  08 00 20 89 90 4c 00 10  4b ba 0e 84 08 00 45 00   .. ..L.. K.....E.
0010  00 40 00 41 40 00 80 06  00 00 0a 00 00 04 0a 00   .@.A@... ........
0020  00 05 07 d0 0c ab 9c c0  ee d1 27 9c 5b 5d b0 12   ........ ..'.[]..
0030  44 70 fa 00 c0 70 02 04  05 b4 01 03 03 00 01 01   Dp...p.. ........
0040  08 0a 00 00 00 00 00 00  00 00 01 01 04 02         ........ ......
```

```
File: "C:\Documents and Settings\Anna\Skrivbord\wash_shixx" 2512 Bytes 00:02:09    P: 27 D: 27 M: 0
```