Optimize NFR

Part 1

———MSSQL Hello Buffer Overflow

benjurry benjurry@xfocus.org

http://www.xfocus.org

http://www.xfocus.net

2003-8-10

Table of Contents

1. Foreword 3
    1.1. About NFR 3
    1.2. Objective of this article 3
    1.3. Vulnerabilities of SQL Sever 4
2. Alert Information 4
3. Detected by NFR 5
    3.1. Signature of MSSQL Hello Buffer Overflow 5
    3.2. Data recorded by NFR 7
4. Analysis of TDS protocol 9
5. Description of Hello Buffer Overflow 16
6. Analysis of Hello Buffer Overflow 18
7. Summary 21

1. Foreword

1.1. About NFR

NFR(Network Flight Recorder) is a well-know IDS ， Which was founded by Marcus J. Ranum, who built the first commercial network firewall product, the DEC SEAL .As a product to analyze and record the network's data, NFR supply the power scripts N-CODE to extend the ability of detecting the intrusion. NFR works well in many evaluations and NFR Security is rated best in class by objective third parties, such as industry analysts, security experts, and independent laboratories .But lack of effective attack signatures is always his heel, although L0pht has written hundreds of signatures for NFR.

1.2. Objective of this article

It is some time when I am using NFR, in my humble, personal opinion; there were many problems about NFR's NIDS. Besides AI(Administration Interface) is abnormal occasional for compatibility with Chinese Version Windows and the descriptions are old although the NIDS is upgraded, the Important trouble is the key of IDS –attack signatures, which are upgraded lingering , further more ,some of them are wrong .

In order to optimize NFR and detect the attack accurately, the series of articles will analyze these problems and correct this error by analyzing the vulnerabilities and exploits. At the same time, these articles will talk about how to produce a good attack signature. Limited by my poor knowledge, there must be some errors, please tell me if you have any advice .My email is: benjurry@xfocus.org

1.3. Vulnerabilities of SQL Sever

SQL Server is database product of Microsoft, which antagonizes Oracle. Now it used widely, but the vulnerabilities of SQL server had been discovered more and more. On January 24, 2003, a worm named Slammer was targeting computers running SQL Server 2000 and MSDE 2000 systems, caused a dramatic increase in network traffic worldwide. The vulnerabilities of SQL server were regarded by security companies. NFR also published the signature of SQL Server, which included a Hello Buffer Overflow vulnerability discovered by Dave Aitel of Immunity on 7th, August ,2002.In the during of deploying NFR, I find many alerts about this attack and analyze it.

2. Alert Information

Following is the alert information of MS SQL Hello Buffer Overflow by NFR NIDS:

```
-----------------------------------------------------------------------------------------
Severity:               Attack
Time:                     13:54:21 15-Jul-2003
Source File:            packages/mssql/sql2k.nfr
Line:                    226
Host:                     benjurry-xfocus
Alert ID:              mssql_sql2k:buffered_hello
Source ID:               mssql_sql2k:source_me
Source:                   mssql_sql2k:source_me
Source Description: Sqlserver 2k overflow detector
Source PID:              36531
Alert Message:          Saw 8 Mssql HELLO overflows from 192.168
```

	0.110 in 900 seconds
:	3
Source IP:	192.168.0.110
Destination IP:	--

Thought the alert, we can see the severity of attack, time, and host of NFR sensor, source IP, destination IP and other information. Because there are hundreds of alerts like this in a day, I think it is a false alarm.

3. Detected by NFR

3.1. Signature of MSSQL Hello Buffer Overflow

Let's open the MSSQL.fp file, which is the signature of MSSQL (We can view the N-CODE in AI ->package also), and we will see as following:

```
<Snip>
…..

#variables defined here…

…
<Snip>
sqlserv_schema = library_schema:new(1, ["time","ip","int","ip","int", "str"],
    scope());
sqlserv_rec = recorder("bin/list %c", "sqlserv_schema");

HELLO_SIG = "\x12\x01\x00\x34\x00\x00\x00\x00\x00\x00\x15";
MIN_LEN = strlen(HELLO_SIG);
```

```
…….
<snip>
filter hello tcp (client, dport: 1433) {
    declare $Blob inside tcp.connsym;
    if ($Blob == NULL) {
        $Blob = tcp.blob;
    } else {
        $Blob = cat($Blob, tcp.blob);
    }

    if (strlen($Blob) < MIN_LEN)
        return;

    if (prefix($Blob, HELLO_SIG)) {
        if (COUNTHELLO[tcp.connsrc]) {
            COUNTHELLO[tcp.connsrc] = COUNTHELLO[tcp.connsrc] + 1;
        } else {
            COUNTHELLO[tcp.connsrc] = 1;
        }
        if (do_alert(hello_overflow_alert, tcp.connsrc)) {
            alert(source_me, hello_overflow_alert, tcp.connsrc,
                tcp.connsport, tcp.conndst, tcp.conndport,
                "--AlertDetails",
                "ALERT_ID", "40-8",
                "ALERT_CONFIDENCE", 60,
                "ALERT_SEVERITY", "medium",
                "ALERT_IMPACT", "unknown",
                "ALERT_EVENT_TYPE", "attack",
                    "ALERT_ASSESSMENT", "unknown",
                    "IP_ADDR_SRC", tcp.connsrc,
                    "PORT_SRC", tcp.connsport,
                    "IP_ADDR_DST", tcp.conndst,
                    "PORT_DST", tcp.conndport,
                    "IP_PROTO_NUM", 6);
          }
          record packet.sec, tcp.conndst, tcp.conndport, tcp.connsrc,
               tcp.connsport, $Blob to sqlserv_rec;
          misc_attacks:rec(packet.sec, scope(),
                                   "Mssql HELLO overflow!", tcp.connsrc,
tcp.conndst);
     }
}
```

From the N-CODE, we can know the procedure of detecting the attack :

First, NFR define the characteristic of the attack,

HELLO_SIG = "\x12\x01\x00\x34\x00\x00\x00\x00\x00\x00\x15";

And the length of the attack characteristic

MIN_LEN = strlen(HELLO_SIG);

Second, take the data from payload, compare the length of data to the length of the attack characteristic, if less the return.

Or compare the data to characteristic, if data include the characteristic, NFR think it is attack, and then NFR will stop the attack or produce alarm.

3.2. Data recorded by NFR

Now let's view the data recorded by NFR , in the menu of AI, we choose package->Query->MSSQL->MSSQL Server 200,click Table to get the data. Select a record and copy ,we can view it as following:

```
----------------------------------------------------------------------------------------------------
Time:                    15-Jul-2003 13:54:21
NFR:                      benjurry-xfocus
Destination Address:192.168.0.135
Destination Port:    1433
Source Address:        192.168.0.110
Source Port:             1391
Payload:

\x12\x01\x004\x00\x00\x00\x00\x00\x00\x15\x00\x06\x01\x00\x1b\x00\x01\x02
\x00\x1c\x00\x0c\x03\x00(\x00\x04\xff\x08\x00\x00\xc2\x00\x00\x00MSSQLSe
rver\x00x\x03\x00\x00
-----------------------------------------------------------------------------------------------
```

This Recorder displays the Time, NFR sensor host, Destination IP and port, source IP and port. We concern about the payload, because these data will be compared to the signature. There are 2 problems here:

1. NFR translate the data which can be translated to ASCII ;
2. NFR cannot deal with Unicode.

In order to analyze, we translate the ASCII to hex except “MSSQLServer”:

```
\x12\x01\x00\x34\x00\x00\x00\x00\x00\x00\x15\x00\x06\x01\x00\x1b\x00\x01\
x02\x00\x1c\x00\x0c\x03\x00\x28\x00\x04\xff\x08\x00\x00\xc2\x00\x00\x00MS
SQLServer\x00\x78\x03\x00\x00
```

When NFR get the packet and find it is a TDS package, it will compare the payload to MSSQL signature. It is obviously that the payload includes the signature, and then an alarm comes into being. Is it a real attack? Let us continue to analyze it.

4. Analysis of TDS protocol

According to the Project of Protocol Analysis in Xfocus, which will be open soon, we know MS SQL 2000 uses TDS 8.0 in its package. The packet format is list as following:

```
        ————————————————————————————————————————————————————————
        | TDS package header (8bytes)  |    TDS payload         |
        ——————————————————————————————————————————————————————
```

```
TDS package header:
—————————————————————————————————————————————————————————————————————
| TOKEN | STATUS | LENGTH | SIGNED NUM | PACKET NUM | WINDOW SIZE |
—————————————————————————————————————————————————————————————————————
```

The field TOKEN is one byte, which shows packet type. In this article, it is 0x12, which shows the request of ConnectionPreLogin to get some values, such as the version of SQL Server. When SQL Server receives these kinds of package, it will pass the package to function in SSlibnet.dll. In my SQL Server with no SP, the function is list as following:

```
.text:42CF6DDD ; Attributes: bp-based frame
.text:42CF6DDD
.text:42CF6DDD                 public ConnectionPreLogin
.text:42CF6DDD ConnectionPreLogin proc near
.text:42CF6DDD
.text:42CF6DDD var_4           = dword ptr -4
.text:42CF6DDD arg_0           = dword ptr  8
.text:42CF6DDD arg_4           = dword ptr  0Ch
.text:42CF6DDD arg_8           = dword ptr  10h
.text:42CF6DDD arg_C           = dword ptr  14h
.text:42CF6DDD arg_10          = dword ptr  18h
```

```
.text:42CF6DDD
.text:42CF6DDD                 push    ebp
.text:42CF6DDE                 mov     ebp, esp
.text:42CF6DE0                 push    ecx
.text:42CF6DE1                 mov     eax, [ebp+arg_0]
.text:42CF6DE4                 mov     ecx, [eax+94h]
.text:42CF6DEA                 mov     [ebp+var_4], ecx
.text:42CF6DED                 cmp     [ebp+var_4], 1
.text:42CF6DF1                 jz      short loc_42CF6E01
.text:42CF6DF3                 cmp     [ebp+var_4], 1
.text:42CF6DF7                 jle     short loc_42CF6E3D
.text:42CF6DF9                 cmp     [ebp+var_4], 3
……
```

The STATUS field has one byte, 0x01 means it is last packet in the TDS session.

The LENGTH field has two bytes, which is the length of TDS package include the length of TDS header.

The SIGNED NUM field has two bytes, which is reserved now.

The PACKET NUM filed has one byte, which shows the sequence number in the current TDS operation.

The WINDOW SIZE filed has one byte, which is reserved now.

When the TOKEN filed is 0x12, the format is:

```
-----------------------------------------------------------------
| TDS package Header (8bytes) |Field Indicator Header| Information |
-----------------------------------------------------------------
```

Field indicator Header is a table whose length is not fixed, each item indicate the information of offset or length. General there are four filed in MS SQL 2000, the structure of Field Indicator Header is list as following:

```
{
   BYTE CNETLIBVERNO;
   WORD CNETLIBVEROFFSET;
   WORD CNETLIBVERLEN;
   BYTE CENYFLAGNO;
   WORD CENYFLAGOFFSET;
   WORD CENYFLAGLEN;
   BYTE SINSTNAMENO;
   WORD SINSTNAMEOFFSET;
   WORD SINSTNAMELEN;
   BYTE CTHREADIDNO;
   WORD CTHREADIDOFFSET;
   WORD CTHREADIDLEN;
   BYTE FILEDEND;
}
```

The structure of information is list as following:

```
{
   BYTE CNETLIBVER[CNETLIBVERLEN]
   BYTE CENYFLAG[CENYFLAGLEN];
   BYTE SINSTNAME[SINSTNAMELEN]
   DWORD CTHREADID[CTHREADIDLEN];
}
```

Field:

CNETLIBVERNO

Offset:0

Length: 1

Meanings: The number about Version of NETLIB in client

Description: None

Remark: The value is 0x00 fixed

CNETLIBVEROFFSET

Offset: 1

Length: 2

Meanings: The Offset of NETLIB Version in client

Description: Network Byte Order

Remark:

CNETLIBVERLEN

Offset: 3

Length: 2

Meanings: The length of information about the version of NETLIB in client

Description: Network Byte Order

Remark: The value is 0x06 fixed

CENYFLAGNO

Offset: 5

Length: 1

Meanings: The number about flag of client encryption.

Description:

Remark: The value is 0x01 fixed

CENYFLAGOFFSET

Offset: 6

Length: 2

Meanings: The offset of flag of client encryption.

Description: Network Byte Order

Remark:

CENYFLAGLEN

Offset: 8

Length: 2

Meanings: The length of flag of client encryption.

Description: Network Byte Order

Remark: The value is 0x01 fixed

SINSTNAMENO

Offset:0XA

Length:1

Meanings:The number of server's instance name.

Description:

Remark: The value is 2 fixed.

SINSTNAMEOFFSET

Offset:0XB

Length:2

Meanings:The offset of of server's instance name.

Description:Network Byte Order

Remark:

SINSTNAMELEN

Offset:0XD

Length:2

Meanings: The length of server's instance name.

Description: Network Byte Order

Remark:

CTHREADIDNO

Offset:0XF

Length:1

Meanings:The number of the client's process.

Description:

Remark: The value is 3 Fixed

CTHREADIDOFFSET

Offset:0X10

Length: 2

Meanings: The offset of the client's process.

Description:Network Byte Order

Remark:

CTHREADIDLEN

Offset:0X12

Length:2

Meanings: The length of the client's process.

Description: Network Byte Order

Remark: The value is 4 fixed.

FILEDEND

Offset:0X14

Length:1

Meanings: This show that the Field Indicator Header is over,and the next is the information

Description:The sign is 0XFF

Remark:

CNETLIBVER

Offset:0X15

Length:6

Meanings: The version of NETLIB

Description: The version of DBNETLIB.DLL

Remark: The format is Network Byte Order，For example,if the version is
 80.528.00，then the fild is
 08 00 02 10 00 00

CENYFLAG

Offset:0X1B

Length:1

Meanings: The flag of Client encryption.

Description:0 encrypt ，1 don't encrypt

Remark:

SINSTNAME

Offset:0X1C

Length:SINSTNAMELEN

Meanings: The instance name asked by client.

Description:

Remark:default is MSSQLserver

CTHREADID

Offset:0X1C+SINSTNAMELEN

Length:4

Meanings:The ID of client's process

Description: host Byte Order

Remark:

From the format list as foregoing, if the default instance is 一 MSSQLserver, the tds package is like:

```
\x12\x01\x00\x34\x00\x00\x00\x00
\x00\x00\x15\x00\x06\x01\x00\x1b
\x00\x01\x02\x00\x1c\x00\x0c\x03
\x00\x28\x00\x04\xff\x08\x00\x00
\xc2\x00\x00\x00MSSQ
LServer\x00
\x78\x03\x00\x00
```

But the signature is :

HELLO_SIG = "\x12\x01\x00\x34\x00\x00\x00\x00\x00\x00\x15";

It is obviously that the signature is a part of normal package of TDS 0x12, that is why there are so many alarms. And how did NFR publish the signature? Let do it from the description of the vulnerability.

5. Description of Hello Buffer Overflow

Following is Description of the vulnerability from NFR N-CODE

FALSE POSITIVES

False positives are unlikely due to the nature of the attack

REFERENCES

Bugtraq Post

http://cert.uni-stuttgart.de/archive/bugtraq/2002/08/msg00125.html

Exploit

http://www.scan-associates.net/papers/sql2kx2.txt

It shows that the vulnerability was discovered by Dave Aitel from http://www.immunitysec.com/，and from http://cert.uni-stuttgart.de/archive/bugtraq/2002/08/msg00125.html, we can read the MS SQL Server Hello Overflow NASL script and the key is following:

```
<snip>
pkt_hdr = raw_string(
0x12 ,0x01 ,0x00 ,0x34 ,0x00 ,0x00 ,0x00 ,0x00  ,0x00 ,0x00 ,0x15 ,0x
00 ,0x06 ,0x01 ,0x00 ,0x1b,0x00 ,0x01 ,0x02 ,0x00 ,0x1c ,0x00 ,0x0c ,
0x03  ,0x00 ,0x28 ,0x00 ,0x04 ,0xff ,0x08 ,0x00 ,0x02,0x10 ,0x00 ,0x0
0 ,0x00
);

pkt_tail = raw_string (
0x00 ,0x24 ,0x01 ,0x00 ,0x00
);
```

<snip>

…..

```
if(get_port_state(port))
{
    soc = open_sock_tcp(port);

    if(soc)
    {

        attack_string=crap(560);
        sql_packet = pkt_hdr+attack_string+pkt_tail;
        send(socket:soc, data:sql_packet);

            r  = recv(socket:soc, length:4096);
            close(soc);
        display ("Result:",r,"\n");
```

```
                if(!r)
                 {
                  display("Security Hole in MSSQL\n");
                 security_hole(port:port, data:report);
                 }
      }
```

pkt_hdr is the package header according TDS, there are 560 Xs in the package ,，pkt_tail is the tail of TDS package。

Now we can see that NFR hadn't researched this vulnerability, but take eleven characters from the scripts as their signature .I think they were irresponsible. It is laughable that the description was "False positives are unlikely due to the nature of the attack ".

Is it NFR who was rated best in class by objective third parties? When I talk about it with my friends Stardust(Stardust@xfocus.org) and Flashsky(FlashSky@xfocus.org), we think relate with that evaluations regard False Negatives but neglect False Positives, because all evaluations are black testing, to evaluate False Negatives only need find some exploits, but to evaluate False Positives want to analyze the system and produce correct behave. In many occasions it is difficult to produce the correct behave without the deeply research. In order to reduce the cost ,many manufacturer of IDS don't research the vulnerability and system, but collect the exploits. That is why there are many False Positives in IDS and why they do well in many evaluations.

Now let us analyze the vulnerability, then write a better signature.

6. Analysis of Hello Buffer Overflow

Unassemble the code in SSlibnet.dll with the tool IDA，and we can see :

```
.text:42CF6F49 loc_42CF6F49:                ; CODE XREF:
sub_42CF6E4F+EA   j
.text:42CF6F49                     mov      eax, [ebp+0xc]
.text:42CF6F4C                     add      eax, [ebp-0x218]
.text:42CF6F52                     push     eax
.text:42CF6F53                     lea      ecx, [ebp-0x214]
.text:42CF6F59                     push     ecx
.text:42CF6F5A                     call     strcpy
.text:42CF6F5F                     add      esp, 8
.text:42CF6F62                     push     offset unk_42D01104
.text:42CF6F67                     lea      edx, [ebp-0x214]
.text:42CF6F6D                     push     edx
.text:42CF6F6E                     call     strcmp
.text:42CF6F73                     add      esp, 8
```

The problem exist in the strcpy when the source strings is more than 0x214(512) bytes. If the source strings is longer than 512,then they will overlay the return address. Here I don't supply the exploits for the vulnerability is too old and almost every system has patch it.

According to the protocol analysis ,the length of package will be calculated in the program, and it can't be "\x00\x34" to avoid to be detected by SQL Server(there is no check in SQL Server 2000).At the same time ,Attacker can divide one package to many package , the "status" may be 0x00 besides 0x1,So there are False Negatives and False Positives at the same time.

After analyzing the vulnerability, we can write our own signature of NFR:

```
sqlserv_schema = library_schema:new(1, ["time","ip","int","ip","int", "str"],
      scope());
sqlserv_rec = recorder("bin/list %c", "sqlserv_schema");

HELLO_SIG = "\x12 ";
#removal the other strings for the package length and dividing package.
MIN_LEN =29;
#Include the TDS package header and the length of Field Indicator Header
…….
<snip>
filter hello tcp (client, dport: 1433) {
      declare $Blob inside tcp.connsym;
      if ($Blob == NULL) {
            $Blob = tcp.blob;
      } else {
            $Blob = cat($Blob, tcp.blob);
      }

      if (strlen($Blob) < MIN_LEN)
            return;

      if (prefix($Blob, HELLO_SIG) && strlen($Blob) > 295) {
      # considering the length of instance is shorter than 255, I choose 295 here.
# 40（package header）+255=295
#And we can add variable here to adjust the length
#alarm
….

}
```

7. Summary

Though the analyzing one of the NFR ‘s signature, we think it is not easy to publish a good signature. To do it well ,we should analyze the vulnerability and some protocols, but not to collect the exploits and take the signature from the exploits.

There are many men who discus whether the IDS is useful. I think we should do it than talking about it. Let do more for IDS!