



INTERNET
SECURITY
SYSTEMS™

RPC Signature Quality

June 2002

Introduction

There is a significant difference in the quality of signatures shipped with popular Intrusion Detection Systems. This paper highlights these by focusing on signatures developed for the Sun Microsystem RPC (Remote Procedure Call) system. Overall, Internet Security Systems' **RealSecure® Protection System**, particularly the **RealSecure® Network Sensor**, has a significant advantage in the effectiveness and quality of these signatures over competing Intrusion Detection Systems and their RPC signatures.

Why the Snort Analysis

RealSecure Network Sensor supports the importing of Snort signatures, which allows RealSecure to become the superset of signatures for BlackICE™ Sentry (now integrated into RealSecure Network Sensor 7.0), RealSecure Network Sensor, and Snort. RealSecure Network Sensor users can utilize Snort rules to quickly update the program. Although users familiar with Snort can benefit from this feature, there is a risk involved with using the open source signature rules, including many false positives and false negatives. While researching the RPC attacks, which are very popular on Unix and Linux systems, researchers discovered many of the IDS signature errors presented in this paper.

RPC Scans

Many of Snort's signatures are written with pattern matching techniques, which result in many false positives and negatives. In comparison, the protocol analysis techniques used by a few commercial IDS products result in virtually no false positives or false negatives.

RPC has some characteristics that make pattern matching difficult. RPC operates using both the TCP and UDP transports and runs on arbitrary ports. This forces the pattern-matching system to examine all packets rather than focus on patterns specific to certain ports.

Instead of detecting the actual intrusion, most of Snort's signatures are in the form of:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (\
msg:"RPC portmap request status"; \
content:"|01 86 B8 00 00|"; \
offset:40;depth:8; \
reference:arachnids,15; \
classtype:rpc-portmap-decode; sid:587; rev:2;)
```

This signature examines incoming Internet traffic for Portmap requests, specifically searching for 0x0186B8, which is the rpc.statd service. These Portmap requests are usually the first stage of communication with the service. Because the RPC service can use any port, a remote client must first query Portmap to discover which port the service is currently using.

These Portmap signatures correlate poorly with intrusions. For example, if a network is using rpc.statd (part of the NFS suite of protocols), a large amount of normal requests for rpc.statd will be sent across the network. If the signature detects all of this traffic, it would be difficult to ascertain whether the traffic is normal or malicious.

Snort attempts to address this situation by adding the \$EXTERNAL_NET -> \$HOME_NET restriction to the rule. This option restricts the signature to examine inbound traffic from the Internet. Normally, NFS and rpc.statd are only used within the internal network and are unlikely to be used for Internet communications. However, while this option solves the problem of too many false positives, it significantly degrades the performance of the IDS. Snort does not seem suitable for detecting internal traffic from internal attackers or intruders who are using an internal system to attack other systems.

Portmap's port 111 is one of the most common ports filtered at firewalls. This forces attackers to use other ways of finding RPC services. Rather than query Portmap, they might portscan a

system to figure out where RPC services are running. Once attackers find an RPC service, they can insert the port number directly into the exploit script. This means the script won't query Portmap, and therefore won't be detected by Snort.

Thus, Snort has an extremely high incidence of false positives and false negatives using this approach. The Cisco Secure IDS (formerly WheelGroup NetRanger) uses the same technique for some of its signatures.

RealSecure Network Sensor uses a different approach that relies upon application layer protocol analysis. Rather than detecting individual `rpc.statd` lookups using Portmap, RealSecure Network Sensor analyzes overall `rpc.statd` traffic. Snort and Cisco detect an early sign that an attack might occur, while RealSecure detects the actual attack.

RPC Exploits

Snort contains a few signatures that detect actual RPC attacks. An example for `rpc.statd` is shown below:

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (\
msg:"RPC EXPLOIT statdx";\
content: "/bin|c74604|/sh";\
reference:arachnids,442;\
classtype:attempted-admin; sid:1282; rev:1;)
```

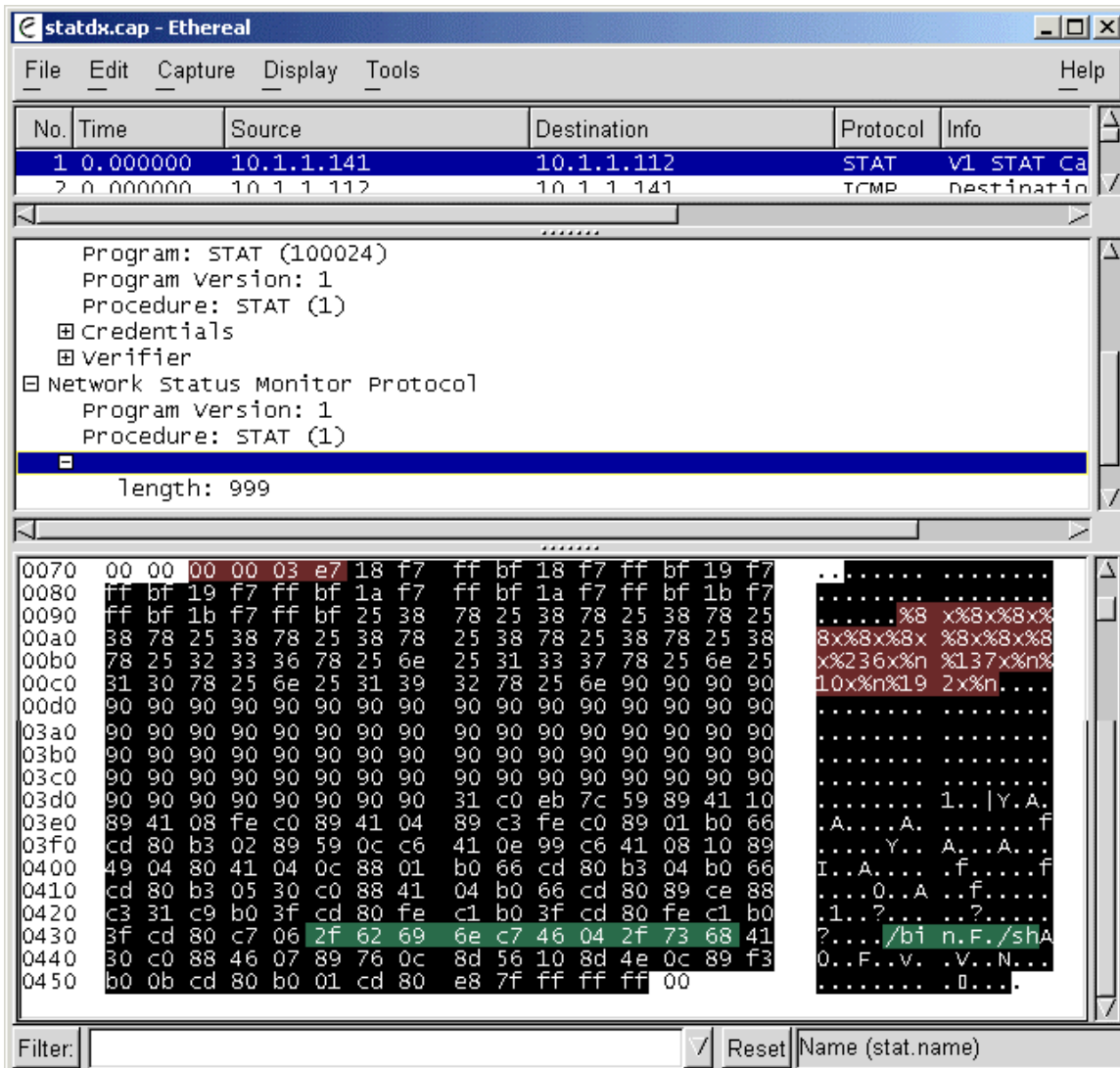
This signature attempts to detect the `statdx.c` exploit, which takes advantage of the “`rpc.statd` format-string vulnerability.” This signature searches for the pattern within the network traffic that was generated by the attack script.

However, `statdx.c` isn't the only script targeting the `rpc.statd` format-string vulnerability. Another script uses different source code and will not be detected by this signature. Yet another `rpc.statd` exploit has been written for Linux on the PowerPC, which cannot be detected by any signature searching for Intel x86 machine code.

In addition, a utility called `ADMutate` can be used to encrypt the part of network traffic that Snort examines. If `ADMutate` is combined with the `statdx.c` script, Snort's signature will be unable to detect this attack.

RealSecure Network Sensor uses a different technique for detecting the `rpc.statd` exploit. The original vulnerability that these scripts exploit is a format-string bug in certain `rpc.statd` fields. RealSecure Network Sensor decodes the `rpc.statd` protocol, breaks apart the individual fields, and analyzes them for format-string contents. All attack scripts (including those encrypted by `ADMutate`) are detected by this protocol-analysis signature.

For example, the following screenshot of the Ethereal packet analyzer shows a protocol decode of the packet generated by the `statdx.c` exploit.



Ethereal is a typical packet analyzer. The top pane shows a packet-by-packet summary. The second pane shows the detail of each field in the packet. The bottom pane shows a hex (hexadecimal) dump of the packet contents. Because of the size of this packet, the data between offsets 0x0d0 through 0x03a0 has been edited for brevity.

The protocol analysis technique used by RealSecure yields a field-by-field decode of the packet using the same methods as Ethereal uses to display the contents in the details pane. The last field of the packet is the 999-byte Filename field. RealSecure analyzes this field and searches its contents for format-string specifiers. If the field is longer than 100 bytes and has more than one format-string specifier, RealSecure triggers a Statd_Format_String alert. The two portions of the string that RealSecure examines are highlighted in red.

In contrast, Snort's signature searches the packet contents within the UDP packet. It treats everything following the UDP header as generic data and doesn't interpret any of the fields. The portion found by Snort is highlighted in green in the above picture.

Note that ADMutate would encrypt the portion highlighted in green, but cannot encrypt the portion highlighted in red.

Snort has only two signatures for rpc.statd: the Portmap lookup described in the previous section and the statdx.c exploit script. RealSecure includes detection for a Solaris buffer overflow vulnerability, a dot-dot directory climbing vulnerability, fingerprinting of popular vulnerability scanners, and detection of a subtle vulnerability that passes rpc.automount requests through rpc.statd. RealSecure also has generic Portmap lookup signatures that detect failed attempts, indicating hacker sweeps.

Like Snort, most of Cisco Secure IDS signatures focus on Portmap lookups rather than actual intrusions. However, it detects sweeps, which are hacker scans of multiple computers for a vulnerable RPC service. Cisco decodes the RPC header but not the application layer. This allows it to work more like RealSecure by detecting intrusions caused by abnormal lengths. Unfortunately, it checks the length of the entire RPC request rather than the length of application layer fields. This can lead to false positives since RPC packets can be very long. In other words, when a certain rpc.statd field exceeds 255 bytes, it is ALWAYS an intrusion. However, when an rpc.statd packet itself exceeds 255 bytes, it is only USUALLY an intrusion.

RPC State-Based Signatures

Systems like Snort and Cisco are stateless. This means they analyze single packets without correlating information among multiple packets. In contrast, RealSecure is a stateful system.

For example, the RealSecure NfsGuess signature detects when an attacker tries many different file handles. Much of NFS security is based on file handles being hard to guess. If an attacker is given enough time, he or she may be able to guess file handles and will be able to access the rest of the system. To detect file handle guessing, RealSecure maintains a list of the file handles that are in use. When enough failed guesses occur, RealSecure can then generate an alert.

Another example is RealSecure's inclusion of returned data. When an attacker scans the system with a Portmap DUMP, BlackICE waits for the outbound response from the system to the attacker. RealSecure then includes the response from the server (success or fail), as well as the Portmap information revealed to the attacker. If the information includes sensitive services (like rpc.statd), then RealSecure will display it to the IDS analyst.

RPC Bugs

After analyzing RealSecure and Snort signatures, Internet Security Systems has a good understanding about the quality of signatures. A careful attacker could evade some Snort signatures, including those that are defective or non-working. Some defective Snort signatures are described in this section.

Example 1: The Snort "RPC NFS Showmount" signature triggers on procedure 5 (EXPORT) sent to the MOUNT service across a TCP port. It has the following three problems:

- The showmount function can be executed over UDP as well as TCP.
- The showmount function can be executed with procedure 6 (EXPORTALL) as well as 5 (EXPORT).
- The Snort signature only looks in the port range used by Solaris (32771 and higher), but many other systems (e.g. Linux) run the MOUNT service at lower ports.

The following packet demonstrates all three of these points.

```

00000000 00 02 7E 26 6A 50 00 03 47 05 09 BC 08 00 45 00 ..~&jP..G.....E.
00000010 00 6C D7 E4 00 00 80 11 27 EF D1 86 AC 8A AC 10 .l.....'.....
00000020 10 8C 02 C5 02 8A 00 58 BD FE 00 00 7A 6A 00 00 .....X....zj..
00000030 00 00 00 00 00 02 00 01 86 A5 00 00 00 02 00 00 .....
00000040 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000050 00 28 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .(.....
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

The significant fields in this packet capture are highlighted in bold: UDP proto=0x11, port hex 0x028A=decimal 650, and procedure EXPORTALL = 0x06.

Example 2: The snort “RPC AMD Overflow” signature has a 1 in 4-billion chance of triggering on a real AMD overflow. The signature will most likely never trigger, and if it does, it is still probably a false positive.

Example 3: The “RPC snmpXdmi overflow attempt” signature has almost the same problem. It isn’t completely wrong, but is unlikely to work well in the real world.

Example 4: The meaning of the following Snort signature is unclear:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (\
msg:"RPC portmap request tooltalk"; flags:A+; \
rpc:100083,*,*; \
reference:cve,CAN-2001-0717; \
classtype:rpc-portmap-decode; sid:1298; rev:2;)
```

It appears that this signature’s intent is to detect the use of the Portmap protocol on port 111 with a ToolTalk lookup (this explanation would be consistent with all the other signatures with the name “RPC portmap request XXX”). However, the signature detects ToolTalk running on port 111 instead of Portmap – a completely different signature for which no known advisory exists.

Conclusion

A comparison of IDS products usually starts by asking, “How many signatures do you support?” As you can see in the Appendices, Snort supports more RPC rules than any of the three commercial products listed here – 59 signatures in total. However, when duplicates and incorrect signatures are removed, that number drops to 26, which is the least of the products listed. Furthermore, Snort only has 3 signatures that detect remote root exploits against RPC services, compared to at least 20 remote root exploits detected by RealSecure, and 6 for Cisco. This is why Internet Security Systems claims to support more attack signatures than any other IDS with RealSecure – it depends upon how you count the signatures.

Many comparisons of IDS products start by putting them in a lab, running exploits, and seeing which IDS signatures trigger. In a lab with no background traffic, the Portmap lookups in Cisco and Snort won’t trigger except when an exploit is run. However, when these systems are placed in a real-world RPC environment, these Portmap signatures will essentially go blind (the Cisco sweep variants should still detect scanning, but exploits won’t be discovered). In contrast, there is little difference between how the RealSecure signatures perform in the lab and in real-world RPC environments.

In real-world RPC environments, you’ll occasionally see “2001724 RPC CALLIT unknown” and “2001721 RPC getport probe” false positives with RealSecure. People who want to get the fewest false positives should disable them, but there are good reasons to leave them enabled. Specifically, the “CALLIT unknown” can be tuned by adding services to those that should be known.

Many comparisons of IDSeS run popular exploits downloaded from Web sites. Reviewers don’t take the trouble of running rarer scripts, running against ports without a Portmap lookup, hiding the code with ADMutate, filtering the RPC traffic with SideStep, or attacking non-x86/non-SPARC platforms. If reviewers went through these steps, then virtually all of the pattern searching signatures would fail to trigger. However, virtually all of the protocol analysis signatures would be unaffected and would trigger as normal. Both Snort and Cisco Secure IDS are affected by this deficiency.

By comparison, Internet Security Systems invests heavily in tracking down rare exploit scripts and testing them with our products. Internet Security Systems also develops its own exploit scripts to test theoretical aspects that don't appear in the wild, e.g. against alternative CPUs like PowerPC or Alpha.

One final note. Many people using IDSeS falsely believe that intrusion detection systems run purely off of the pattern match system that Snort exemplifies. Several of the Snort signatures in the appendix use the technique of finding a unique string in the exploit, and then scanning for that pattern as part of a signature. However, if you look at the two commercial products in the appendix (Cisco Secure IDS and RealSecure), *none* of them use that technique for their RPC signatures. All three products decode up to the basic RPC layers, and RealSecure performs extensive application layer decoding.

Appendix: RPC Signatures Of Various Products

Snort

The following are the Snort signatures at the time this document was written (v1.8.3, January 22, 2002):

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC snmpXdmi overflow attempt";
flags:a+; content:"|0000 0f9c|"; offset:0; depth:4; content:"|00018799|"; offset: 16; depth:4;
reference:bugtraq,2417; reference:cve,CAN-2001-0236; classtype:attempted-admin; sid:569;
rev:2)
```

After running many of the snmpXdmi exploits, none of them have triggered this rule. An intruder can purposely evade this signature by using the SideStep program, changing the length, or inserting empty fields into the RPC header to cause the offsets to vary.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 32771:34000 (msg:"RPC EXPLOIT ttdbserve
solaris overflow"; content: "|C0 22 3F FC A2 02 20 09 C0 2C 7F FF E2 22 3F F4|"; flags: A+;
dsiz e: >999; reference:bugtraq,122; reference:cve,CVE-1999-0003; reference:arachnids,242;
classtype:attempted-admin; sid:570; rev:2;)
```

This signature works by detecting exploit code. Evasion is easy by changing the shellcode. Also, the exploit only works for attacks against Solaris/SPARC; exploits against Solaris/X86, AIX, HP-UX, Irix, etc. will not be detected, and neither will variants over UDP. There are also several versions of the exploit that attack Solaris/SPARC.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 32771:34000 (msg:"RPC EXPLOIT ttdbserve
Solaris overflow"; flags: A+; dsiz e: >999; content: "|00 01 86 F3 00 00 00 01 00 00 00 0F 00 00
00 01|"; reference:bugtraq,122; reference:cve,CVE-1999-0003; reference:arachnids,242;
classtype:attempted-admin; sid:571; rev:1;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 32771:34000 (msg:"RPC DOS ttdbserve solaris";
flags: A+; content: "|00 01 86 F3 00 00 00 01 00 00 00 0F 00 00 00 01|";offset: 16; depth: 32;
reference:bugtraq,122; reference:arachnids,241; reference:cve,CVE-1999-0003;
classtype:attempted-dos; sid:572; rev:2;)
```

This signature triggers on any RPC call to ToolTalk procedure 15, which will produce false positives in environments using ToolTalk.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 634:1400 (msg:"RPC AMD Overflow"; flags:
A+; content: "|80 00 04 2C 4C 15 75 5B 00 00 00 00 00 00 02|";depth: 32;
reference:arachnids,217; classtype:attempted-admin; sid:573; rev:1;)
```

This signature won't work. See comments in the Bugs section earlier in this paper.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 32771: (msg:"RPC NFS Showmount"; flags:A+; content:"|00 01 86 A5 00 00 00 01 00 00 00 05 00 00 00 01|"; offset: 16; depth: 32; reference:arachnids,26; classtype:attempted-recon; sid:574; rev:1;)
```

Triggers on procedure #5, however, procedure #6 provides equivalent output. This can also be done over UDP. See comments in the Bugs section earlier in this paper.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request admin"; content:"|01 86 F7 00 00|";offset:40;depth:8; reference:arachnids,18; classtype:rpc-portmap-decode; sid:575; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request admin"; content:"|01 86 F7 00 00|";offset:40;depth:8; reference:arachnids,18; classtype:rpc-portmap-decode; flags:A+; sid:1262; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request amount"; content:"|01 87 03 00 00|";offset:40;depth:8; reference:arachnids,19;classtype:rpc-portmap-decode; sid:576; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request amount"; content:"|01 87 03 00 00|";offset:40;depth:8; reference:arachnids,19; classtype:rpc-portmap-decode; flags:A+; sid:1263; rev:3;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request bootparam"; content:"|01 86 BA 00 00|";offset:40;depth:8; reference:arachnids,16; classtype:rpc-portmap-decode; sid:577; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request bootparam"; content:"|01 86 BA 00 00|";offset:40;depth:8; reference:arachnids,16; classtype:rpc-portmap-decode; flags:A+; sid:1264; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request cmsd"; content:"|01 86 E4 00 00|";offset:40;depth:8; reference:arachnids,17; classtype:rpc-portmap-decode; sid:578; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request cmsd"; content:"|01 86 E4 00 00|";offset:40;depth:8; reference:arachnids,17; classtype:rpc-portmap-decode; flags:A+; sid:1265; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request mount"; content:"|01 86 A5 00 00|";offset:40;depth:8; reference:arachnids,13; classtype:rpc-portmap-decode; sid:579; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request mount"; content:"|01 86 A5 00 00|";offset:40;depth:8; reference:arachnids,13; classtype:rpc-portmap-decode; flags:A+; sid:1266; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request nisd"; content:"|01 87 cc 00 00|";offset:40;depth:8; reference:arachnids,21; classtype:rpc-portmap-decode; sid:580; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request nisd"; content:"|01 87 cc 00 00|";offset:40;depth:8; reference:arachnids,21; classtype:rpc-portmap-decode; flags:A+; sid:1267; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request pcnfsd";
content:"|02 49 f1 00 00|";offset:40;depth:8; reference:arachnids,22; classtype:rpc-portmap-
decode; sid:581; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request pcnfsd";
content:"|02 49 f1 00 00|";offset:40;depth:8; reference:arachnids,22; classtype:rpc-portmap-
decode; flags:A+; sid:1268; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request
rexnd";content:"|01 86 B1 00 00|";offset:40;depth:8; reference:arachnids,23; classtype:rpc-
portmap-decode; sid:582; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request
rexnd";content:"|01 86 B1 00 00|";offset:40;depth:8; reference:arachnids,23; classtype:rpc-
portmap-decode; flags:A+; sid:1269; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request rstatd";
content: "|01 86 A1 00 00|"; reference:arachnids,10; classtype:rpc-portmap-decode; sid:583;
rev:3;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request rstatd";
content: "|01 86 A1 00 00|"; reference:arachnids,10; classtype:rpc-portmap-decode; flags:A+;
sid:1270; rev:3;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request rusers";
content:"|01 86 A2 00 00|";offset:40;depth:8; reference:arachnids,133; classtype:rpc-portmap-
decode; flags:A+; sid:1271; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request rusers";
content:"|01 86 A2 00 00|";offset:40;depth:8; reference:arachnids,133; classtype:rpc-portmap-
decode; sid:584; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request sadmind";
content:"|01 87 88 00 00|";offset:40;depth:8; reference:arachnids,20; classtype:rpc-portmap-
decode; flags:A+; sid:1272; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request sadmind";
content:"|01 87 88 00 00|";offset:40;depth:8; reference:arachnids,20; classtype:rpc-portmap-
decode; sid:585; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request
selection_svc"; content:"|01 86 AF 00 00|";offset:40;depth:8; reference:arachnids,25;
classtype:rpc-portmap-decode; sid:586; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request
selection_svc"; content:"|01 86 AF 00 00|";offset:40;depth:8; reference:arachnids,25;
classtype:rpc-portmap-decode; flags:A+; sid:1273; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request status";
content:"|01 86 B8 00 00|";offset:40;depth:8; reference:arachnids,15; classtype:rpc-portmap-
decode; sid:587; rev:2;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request ttdbserve";
content:"|01 86 F3 00 00|";offset:40;depth:8; reference:arachnids,24; classtype:rpc-portmap-
decode; sid:588; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request ttdbserve";
content:"|01 86 F3 00 00|";offset:40;depth:8; reference:arachnids,24; classtype:rpc-portmap-
decode; flags:A+; sid:1274; rev:2;)

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request yppasswd";
content:"|01 86 A9 00 00|";offset:40;depth:8; reference:arachnids,14; classtype:rpc-portmap-
decode; sid:589; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request yppasswd";
content:"|01 86 A9 00 00|";offset:40;depth:8; reference:arachnids,14; classtype:rpc-portmap-
decode; flags:A+; sid:1275; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request ypserv";
content:"|01 86 A4 00 00|";offset:40;depth:8; reference:arachnids,12; classtype:rpc-portmap-
decode; flags:A+; sid:1276; rev:2;)

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request ypserv";
content:"|01 86 A4 00 00|";offset:40;depth:8; reference:arachnids,12; classtype:rpc-portmap-
decode; sid:590; rev:2;)

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request ypupdated";
content:"|01 86 BC 00 00|";offset:40;depth:8; reference:arachnids,125; classtype:rpc-portmap-
decode; sid:1277; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request ypupdated";
flags:A+; content:"|01 86 BC 00 00|";offset:40;depth:8; reference:arachnids,125; classtype:rpc-
portmap-decode; sid:591; rev:3;)

alert udp $EXTERNAL_NET any -> $HOME_NET 32770: (msg:"RPC rstatd query"; content:"|00
00 00 00 00 00 02 00 01 86 A1|";offset:5; reference:arachnids,9;classtype:attempted-recon;
sid:592; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 32770: (msg:"RPC rstatd query"; flags:A+;
content:"|00 00 00 00 00 00 02 00 01 86 A1|";offset:5;
reference:arachnids,9;classtype:attempted-recon; sid:1278; rev:1;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request tooltalk";
flags:A+; rpc:100083,*,*; reference:cve,CAN-2001-0717; classtype:rpc-portmap-decode;
sid:1298; rev:2;)
See the Bugs section.

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request tooltalk";
rpc:100083,*,*; reference:cve,CAN-2001-0717; classtype:rpc-portmap-decode; sid:1299; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request snmpXdmi";
flags:A+; rpc:100249,*,*; reference:bugtraq,2417; classtype:rpc-portmap-decode; sid:593; rev:4;)

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request snmpXdmi";
rpc:100249,*,*; reference:bugtraq,2417; classtype:rpc-portmap-decode; sid:1279; rev:3;)

alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request espd";
rpc:391029,*,*; reference:cve,CAN-2001-0331; classtype:rpc-portmap-decode; sid:594; rev:3;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request espd";
rpc:391029,*,*; flags:A+; reference:cve,CAN-2001-0331; classtype:rpc-portmap-decode; sid:595;
rev:4;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request yppasswdd";
rpc:100009,*,*; reference:bugtraq,2763; classtype:rpc-portmap-decode; sid:1296; rev:3;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request yppasswdd";
rpc:100009,*,*; flags:A+; reference:bugtraq,2763; classtype:rpc-portmap-decode; sid:1297;
rev:4;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap listing"; flags: A+;
rpc: 100000,*,*;reference:arachnids,429; classtype:rpc-portmap-decode; sid:596; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 32771 (msg:"RPC portmap listing"; flags: A+;
rpc: 100000,*,*;reference:arachnids,429; classtype:rpc-portmap-decode; sid:597; rev:2;)
```

See comments in the Bugs section earlier in this paper.

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap listing"; content: "|00
01 86 A0 00 00 00 02 00 00 00 04|"; reference:arachnids,429; classtype:rpc-portmap-decode;
sid:1280; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap listing"; flags:A+;
content: "|00 01 86 A0 00 00 00 02 00 00 00 04|"; reference:arachnids,429; classtype:rpc-
portmap-decode; sid:598; rev:3;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 32771 (msg:"RPC portmap listing"; flags:A+;
content: "|00 01 86 A0 00 00 00 02 00 00 00 04|"; reference:arachnids,429; classtype:rpc-
portmap-decode; sid:599; rev:3;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 32771 (msg:"RPC portmap listing"; content:
"|00 01 86 A0 00 00 00 02 00 00 00 04|"; reference:arachnids,429; classtype:rpc-portmap-
decode; sid:1281; rev:2;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC EXPLOIT statdx"; flags: A+;
content: "/bin|c74604|/sh";reference:arachnids,442; classtype:attempted-admin; sid:600; rev:1;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC EXPLOIT statdx"; content:
"/bin|c74604|/sh";reference:arachnids,442; classtype:attempted-admin; sid:1282; rev:1;)
```

```
alert udp $EXTERNAL_NET any -> $HOME_NET 32770: (msg:"RPC rusers query";
content:"|00000000000000002000186A2|";offset:5; reference:cve,CVE-1999-0626;
reference:arachnids,136; classtype:attempted-recon; sid:612; rev:1;)
```

Cisco

The following are the latest Cisco Secure IDS v3.x signatures..

6100 - RPC Port Registration

6101 - RPC Port Unregistration

6102 - RPC Dump

Same SET, UNSET, and DUMP signatures described above.

6103 - Proxied RPC Request

Similar to the three CALLIT signatures in RealSecure, except it doesn't differentiate between hostile and non-hostile requests.

6104 - RPC Set Spoof, 6105 - RPC Unset Spoof

Same as the SET, UNSET, but from spoofed IPs (same as RealSecure signatures of the same name).

6110 - RPC RSTATD Sweep

6111 - RPC RUSERSD Sweep

6112 - RPC NFS Sweep

6113 - RPC MOUNTD Sweep

6114 - RPC YPPASSWDD Sweep

6115 - RPC SELECTION_SVC Sweep

6116 - RPC REXD Sweep

6117 - RPC STATUS Sweep

6118 - RPC ttdb Sweep

Same as the portmap request signatures, except detected when it hits multiple devices in a clear sweep (state-based signature that correlates better with hostile scans than individual GETPORT lookups).

6150 - ypserv Portmap Request, 6151 - ypbind Portmap Request, 6152 - yppasswdd Portmap Request, 6153 - ypupdated Portmap Request, 6154 - ypxfrd Portmap Request, 6155 - mountd Portmap Request

6175 - rexd Portmap Request

6180 - rexd Attempt

Same as the Snort signatures.

6190 - statd Buffer Overflow

6191 - RPC.tooltalk buffer overflow

6192 - RPC mountd Buffer Overflow

6193 - RPC CMSD Buffer Overflow

6194 - sadmind RPC Buffer Overflow

6195 - RPC amd Buffer Overflow

Similar to the RealSecure decodes. The description of the AMD Buffer Overflow simply looks to see if the protocol is AMD and if the packet is very long. This correlates less well with intrusions that break down the individual fields, potentially allowing lots of events in the console triggered by normal AMD (sadmind, cmsd, mountd, tooltalk, and statd) traffic. The RealSecure statd buffer overflow signature triggers on many of the statd format-string attacks, and the Cisco signature will most likely do the same.

RealSecure Sentry (subsumed into RealSecure Network Sensor 7.0)

The following are the latest RealSecure Sentry v3.0 signatures, now subsumed into RealSecure Network Sensor v7.0.

.

2001701 rpc.automountd overflow

Decodes automountd and triggers an overflow in a particular field.

2001702 rpc.statd overflow

Decodes statd and triggers an overflow in a particular field.

2001703 rpc.tooltalkd overflow

Decodes tooltalk and triggers an overflow in a particular field.

2001704 rpc.admind auth

Decodes admind and triggers when bad authentication information is entered.

2001705 rpc.portmap dump

Decodes Portmap and triggers on a DUMP command, equivalent to the 6 Snort "RPC portmap

listing” signatures. Note that RPC signatures run over both TCP and UDP, and will run on any port, so they work on both port 111 and the ephemeral 32771 port from some versions of Solaris.

2001706 rpc.mountd overflow

Decodes mountd and triggers on an overflow in a particular field. Note that a common Linux worm attacked port 635 without doing a Portmap lookup – this signature still catches that worm, though it evades the Snort GETPORT signature.

2001707 RPC nfs/lockd attack

Triggers on NFS requests sent to the lockd service.

2001708 rpc.portmap.set

Triggers on any SET request – these commands should be internal to a machine but never seen remotely.

2001709 rpc.portmap.unset

Same trigger as SET.

2001710 rpc.pcnfs backdoor

Triggers on a certain string contained within a certain field within PCNFS that allows root control over the computer (not a buffer-overflow or format-string attack, something else).

2001711 rpc.statd dotdot file create

Triggers on a directory-climbing bug in one of the file names passed to statd.

2001712 rpc.yppupdated command

Triggers on a certain string contained within a certain field within YPUPDATE.

2001713 rpc.nfs uid is zero

Triggers on a certain RPC credentials sent along with NFS requests.

2001714 rpc.nfs mknod

Triggers on certain values passed within the MKNOD command on NFS.

2001715 rpc.nisd long name

Triggers on a certain field within NIS exceeding a certain length (i.e. a buffer overflow).

2001716 rpc.statd with automount

Triggers on a certain value within rpc.statd protocol that results in an automountd request.

2001717 rpc.cmsd overflow

Triggers on a certain field within rpc.cmsd that exceeds a certain length.

2001718 rpc.amd overflow

Triggers on a certain field within amd that exceeds a certain length.

2001719 RPC bad credentials

Decodes RPC credentials field (part of all RPC requests) and triggers on a couple of bad formats.

2001720 RPC suspicious credentials

Decodes RPC credentials and triggers on values that are not illegal (bad credentials), but are at least unusual.

2001721 RPC getport probe

Triggers upon an incoming getport request that fails, either because the service responds with an error, or because portmap isn't running at all (actually several underlying signatures).

2001722 rpc.sadmind overflow

Triggers on a certain field within sadmind exceeding a certain length.

2001723 rpc.SGI FAM access

Triggers on a certain field within FAM exceeding a certain length.

2001724 RPC CALLIT unknown

One of the most complicated signatures. Attempts to discover attempts at forwarding requests through the CALLIT service. This signature is a frequent source of false positives. Customers report this attack to us, we determine if this is a known use of CALLIT, and then add that number to our list so it won't trigger in the next version.

2001725 RPC CALLIT attack

A known CALLIT that indicates an attack. The list of known and unknown CALLITs within the system is about 100 signatures total.

2001726 RPC CALLIT mount

The most common RPC CALLIT attack.

2001727 rpc.bootparam whoami mismatch

A decode for hostile use of the bootparam protocol.

2001728 RPC prog grind

A state-based signature that detects multiple different RPC program numbers sent to the same port. Some scanners do this to detect which RPC services are running on which ports, either to avoid detection by the Snort GETPORT/listing signatures above, or to get around firewalls that filter port 111.

2001729 RPC high-port portmap

Access to the Portmap at a port higher than 32768. Certain versions of Solaris accidentally put Portmap at these high-ports – it isn't supposed to be used at all. Similar to the Snort signature, but allows a range of ports.

2001730 RPC ypbind directory climb

Triggers on a certain field within ypbind containing a ../.. directory climbing pattern.

2001731 RPC showmount exports

Similar to the Snort signatures, triggers on the MOUNT EXPORT command (proc=5). It also triggers on EXPORTALL (proc=6). The current Snort signature can be evaded by using proc=6 rather than proc=5.

2001732 RPC selection_svc hold file

Triggers on a certain command within this protocol.

2001733 RPC suspicious lookup

Equivalent of all the Snort GETPORT lookups. This doesn't do anything by default, but if the user really likes the idea of triggering on GETPORTs, they can list all the RPC program numbers that they think are suspicious.

2001734 RPC SNMPXDMID overflow

Triggers on a certain field exceeding a certain length within this protocol.

2001735 RPC CALLIT ping

A specific abnormal use of CALLIT used by some attack scripts as a discovery process.

2001736 RPC yppasswdd overflow

Triggers on a certain field exceeding a certain length within this protocol.

2001737 rpc.statd Format Attack

Triggers on a certain field exceeding a certain (short) length and containing format-string specifiers.

2001738 RPC Sadmin Ping

A certain command in Sadmin that indicate hostile intentions.

2001739 RPC Amd Version

Execution of a certain command within AMD that gets version information. Technically, allowed by the protocol and legal, but indicates hostile intent.

2001740 RPC Amd Pid

Execution of a certain command within AMD that gets process information. Technically legal, but indicates hostile intent.

RealSecure

The following are RealSecure v6.0 Signatures (subsumed into RealSecure v7.0):

Admin – same as BlackICE signature.

Amd_Overflow – same as BlackICE signature.

Amd_Pid – Same as BlackICE signature.

Amd_Version – Same as BlackICE signature.

Bootparam – Triggers on any call to BOOTPARAM, not as good as BlackICE signature.

MountdExport – Same as BlackICE signature.

MountdMnt – A decode signature that allows all NFS mount requests to be audited.

NIS_Overflow – Same as BlackICE signature.

NfsGuess – A state-based signature that decodes NFS failed file access attempts to detect hackers attempting to grind through NFS file-handles.

NfsMknod – Same as BlackICE signature.

NfsUid – Same as BlackICE signature.

PcnfsdExec – Same as BlackICE signature.

PmapDump – Same as BlackICE signature.

PmapMnt – Same as BlackICE signature (CALLIT proxied mount request).

PmapProxy – Same as the Cisco signature.

PmapUnset, PmapSet – Same as the Cisco/BlackICE signature.

PmapUnsetSpoof, PmapSetSpoof – Same as Cisco signature.

RPC_Cmsd_Overflow – Same as BlackICE signature.

RPC_snmpXdmid_Overflow – Same as BlackICE signature.

Rexd – Triggers on any usage of the rpc.rexd service, which is so horribly insecure it should never be enabled.

Sadmin_Amslverify_Overflow – Same as BlackICE signature.

Sadmin_Ping – Same as BlackICE signature.

SeISvcH – Same as BlackICE signature.

Snoop_GetQuota_Overflow – Triggers on a certain field in a certain RPC protocol that when sniffed, will cause the snoop packet sniffer to buffer overflow.

Statd_Automount_Exec – Same as BlackICE signature.

Statd_DotDot – Same as BlackICE signature.

Statd_Format_Attack – Same as BlackICE signature.

Statd_Overflow – Same as BlackICE signature.

ToolTalk_Overflow – Same as BlackICE signature.

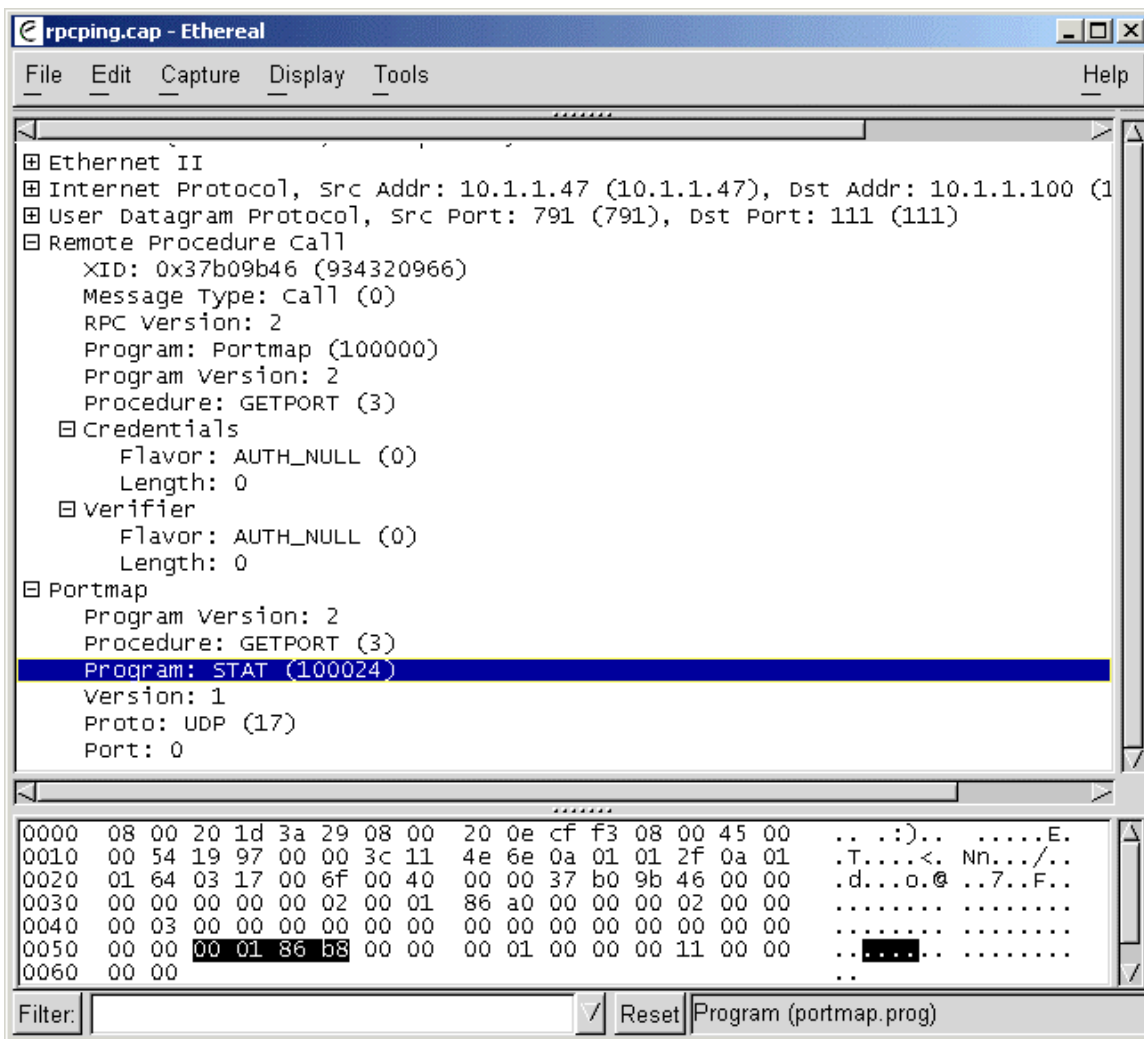
Ypupdate_Exec – Same as BlackICE signature.

Appendix: How Signatures Are Written

Using pattern match technology, signatures are written by extracting unique fingerprints from known exploit scripts. These are unique patterns generated by these exploit scripts that aren't likely to be seen in normal traffic. If these patterns are seen within network traffic, there is a high probability an attacker is running that exploit script against you.

Using protocol analysis technology, signatures are written to analyze or decode the network traffic. Rather than attempting to identify the exploit script, these signatures analyze what the traffic is doing, looking for the electronic equivalent of carrying a loaded gun into a bank.

The best way to understand the difference is by looking at a screen shot from the Ethereal packet sniffer.



Ethereal displays the contents of a packet in two ways. The bottom pane shows a raw hex (hexadecimal) dump of the binary data. The top pane shows a detailed field-by-field analysis of the raw data. The two panes are associated with each other – clicking a field in the upper pane causes the equivalent raw bytes in the lower pane to be highlighted. Clicking in the lower pane causes the equivalent field in the upper pane to be highlighted.

The packet being analyzed in this Ethereal screen shot is a Portmap GETPORT lookup of the rpc.statd service. (Note: whenever two RPC programs interact, they must first discover which port

they are running on through the Portmap service.) Since there are several known vulnerabilities in rpc.statd, such incoming requests are of interest to IDS products.

Ethereal shows that the STAT (rpc.statd) program has the decimal value of 100024, which maps to a hexadecimal value of 0x000186B8 (which corresponds to the bytes 00 01 86 B8).

There are two ways an IDS can trigger on this packet. The pattern match method is to search for the pattern 00 01 86 b8 in the raw data in the lower pane. In the Snort IDS, this would be specified using the following signature:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (\
msg:"RPC portmap request status"; \
content:"|00 01 86 B8|";\
offset:40;depth:8; \
reference:arachnids,15; \
classtype:rpc-portmap-decode; sid:587; rev:2;)
```

The protocol analysis method would be to do a field-by-field analysis of the packet in much the same manner as Ethereal. The Portmap Program field value is compared against the decimal value 100024. If the values match, then the signature triggers.

About Internet Security Systems (ISS)

Founded in 1994, Internet Security Systems (ISS) (Nasdaq: ISSX) is a world leader in software and services that protect critical online resources from attack and misuse. ISS is headquartered in Atlanta, GA, with additional operations throughout the United States and in Asia, Australia, Europe, Latin America and the Middle East.

Copyright © 2002, Internet Security Systems, Inc. All rights reserved worldwide.

Internet Security Systems, the Internet Security Systems logo, and X-Force are trademarks, and RealSecure a registered trademark, of Internet Security Systems, Inc. BlackICE is a licensed trademark, of Network ICE Corporation, a wholly owned subsidiary of Internet Security Systems, Inc. Other marks and trade names mentioned are marks and names of their owners as indicated. All marks are the property of their respective owners and used in an editorial context without intent of infringement. Specifications and content are subject to change without notice.