“NIDS And Dealing With Encryption”

By Patrik Sternudd

Written during Spring 2004, as a part of my GCIA Practical Assignment.

Table of Contents

1. Introduction

Sometimes, I get the feeling the attitude towards NIDS (Network Intrusion Detection Systems) and encrypted data is that as soon as you encounter it, you are out of luck. But is that really the case? Should we just shrug and say *"there's nothing for it"* and go for a coffee? Personally, I do not think so.

In this paper, I will take a look on some common methods for encrypting network data and attempt to answer two primary questions:

1. How do we know we are dealing with encryption?
2. Can we write rules for Snort [0], allowing us to detect it?

My goal is to show that even though most of the data is encrypted, there is still some information available which we can use to enhance our security.

In order to limit the scope, there are some issues I will not cover:

- Loki and other innovate backdoor tunnelling which may or may not be encrypted.
- Strategic placement of NIDS sensors to sidestep the problem (it can be argued this does not help – what if encrypted traffic appears nevertheless?).

Neither will I attempt to cover all possible ways to do encryption; I will only select a few to show that it is, in fact, possible to detect this kind of data when desirable.

This paper is highly technical. The reader should be familiar with TCP/IP concepts and terminology. Generally accepted acronyms may be used without further explanations. A basic understanding of cryptography may be useful, but is not required. Familiarity with Snort is likely to be helpful.

All IP addresses and host names has been obfuscated; any semblance with real networks or domains are entirely coincidental, and should be regarded as such. Network traces has been arbitrarily truncated to allow for easier reading.

2. Why should we care at all?

As security professionals, why should we care? Is not data encryption a good thing, something that adds security to our network?

Well, it is, but only when we are in control of it. Encryption could be used against us as well. It is referred to within that context in several papers dealing with NIDS evasion techniques.

It could also be used in various backdoors and worms calling home.

Another thing to worry about is if - and, in particular, how - it is utilised internally. A lot of encrypted sessions on our internal networks could make us lose the real picture of what is going on. This is particularly true if end users are bringing up encrypted tunnels to the outside. On the other hand, we will certainly want to encrypt our sessions to critical systems such as IDS sensors. System administrators wants (or at least should want) to use SSH to manage servers and network devices. And end users want the privacy offered by encryption services when they access certain resources on the Internet (such as on-line banking).

I will further make three observations that together provide the rationale for the focus of this paper (which is aimed more towards detecting policy violations than the latest worm or hacker attacks):

1. In my experience, people sometimes seem to feel that the IANA[1] assigned numbers[2] are as law. If it comes on port 22, it must be SSH, because it says so, right there in the specification!

2. On a similar note, there are a few ports that almost always seem to be open outbound, even though the firewall is rather restrictive otherwise. Among these, TCP/80 (HTTP) and TCP/443 (HTTPS) are by far the most frequent ones.

3. Security personnel would do well never to underestimate the creativity of end users who feel they are impeded by the firewall or other security measures. They will find ways around it, sooner or later (this is increasingly true for technically savvy people, such as normal system administrators, or simply just people fooling around with computers at home).

The corollary: seeing TCP traffic on ports 22 and 443, we might assume we are seeing SSH and HTTPS. That may unfortunately be an incorrect assumption. What if someone ran a SSH server on port 443 in order to sneak through the firewall?

The issues described above could of course be mitigated by having restrictive policies and firewall rulesets in place, and only allowing traffic to pass through

1 Internet Assigned Numbers Authority
2 As of Jan 2002, the Internet Assigned Numbers is no longer a RFC[1]; the latest version is instead available on the IANA website[2].

application layer proxies. But still, many sites do have quite liberal rulebases (or no firewalls at all, e.g. some universities). In those cases, detecting possibly malicious traffic becomes even more important.

3. What should we look for, then?

While one could theoretically perform some sort of analysis on the encrypted payload in order to pinpoint encryption algorithms, it does not work out very well in reality. Not yet, at least, although anomaly[3] detection systems may be of some use (they do not necessarily need to know the specifics of the payload; instead they may alert on the fact that previously unknown data is flowing from an usually quiet internal system to the Internet).

For one thing, there are numerous algorithms that are commonly used. Then consider that many of these may be run in different modes (for example, SSL v3.0 supports 37 different cipher specifications). It would take an extensive amount of work (and a vast competence in cryptography, which I do not have) to be able to detect all conceivable combinations.

Add to this the fact that this method will require more data to perform the analysis, and we have reached a point where we cannot automate the process without spending huge memory and CPU resources (and perhaps not even then). The old but still excellent treatise by Thomas H. Ptacek and Timothy Newham[4] mentions this in passing.

I will take another approach and take a look at the lower levels of the communication which are supporting the different algorithms. Perhaps we could call them “encryption carriers”. The most common of these are SSL, TLS, SSH and IPSec (there are quite a few of them, the list could definitely be larger).

If we can detect the carriers in a somewhat simple manner, it does not really matter that we do not know the exact algorithm. And unless we want to ensure that we are using secure (in regard to acceptable risk) methods, or want to decrypt the data, we do not even care. If I see a SSL Handshake on the wire, I can then assume that some encryption is going on. The same thing applies for IPSec SA negotiations.

For the remainder of this paper, I will take a closer look at SSL, TLS and SSH in order to describe their characteristics. These characteristics will later be used to write rules for detection. I had originally included an IPSec part as well, but it was removed to make the other sections more in-depth. IPSec is also easier to control at the border – block IP protocols 50, 51 and UDP port 500 and you will probably be safe[4].

3 For more information regarding different types of intrusion detection systems, I recommend the book “Intrusion Detection” by Rebecca Gurley Bace[3].

4 It is always possible to tunnel everything through something else, butmost VPN services (even those that are encapsulated to handle NAT) use IKE on UDP 500.

4. A problem (or two) with Snort

I found that for many detects, it would be useful to look only at specific packets in a TCP session[5]. But how can we tell Snort we are only interested in the first packet after a three-way-handshake, or TWH (this would normally be packet number four in a TCP session)? In theory, it is not has hard as it initially appears; such packets have relative SEQ (sequence) and ACK (acknowledgement) numbers of 1 and a payload greater than zero (if it is zero, it is the ACK completing the handshake).

The problem is that Snort has no capability to inspect *relative* ACK or SEQ numbers, only absolute ones (it is of course much harder to deal with relative ones, as this require awareness of the session state).

To my everlasting gratitude, during the time I considered how to resolve this problem, the Snort Team released Snort 2.1.1 with a new capability called *flowbits*[6], allowing user defined attributes to be set on flows (I do have a hunch that I am not using them quite as the authors imagined, but nevertheless, it solved my problem).

While proceeding with the rule creation, I found another problem. The stream4 preprocessor tags a stream as established as soon the TWH is completed. Well and good, but it then passes it on to the detection module. So, when using the established keyword without any content checks, the first packet it will trigger on is *not* the first data packet after the TWH, but the ACK message completing the TWH. That really messed up my rules.

One workaround would be to use the “flags: A+” check, but getting rid of that was one of the reasons for making the “established” check. Also, even though they are extremely common, PSH (push) and other TCP flags are optional, not required. The workaround I finally decided for was to check for data sizes greater than zero (the ACK completing the TWH should not contain any data).

5. SSH

SSH connections are quite straightforward to detect. Section 4.2 (Protocol Version Exchange) in the Internet-Draft[7] states:

> *When the connection has been established, both sides MUST send an identification string. This identification string MUST be*
>
> *SSH-protoversion-softwareversion SP comments CR LF*

As a quick note: the draft also says this string not necessarily must be the first line sent, but it appears to be the case in most implementations.

5 TCP is described in RFC 793 [5]
6 This is covered in the Snort documentation which is available online [6].
7 Ylonen & Lonvick p.4. [7]

All we need to do is to take a look at the first packet after the completed TWH. If the first four octets contain the ASCII string “SSH-”, chances are good that we are seeing a SSH session. A sample packet is showed in figure 1. The TWH is not included in the sample, nor will it be in subsequent ones.

```
IP server.net.22 > client.org.1536: P 1:21(20) ack 1

0x0000   4500 003c 946c 4000 3906 XXXX XXXX XXXX        E..<.l@.9.......
0x0010   XXXX XXXX 0016 0600 4dfb 7c0c fca7 1442        ........M.|....B
0x0020   5018 c4e0 XXXX 0000 5353 482d 322e 302d        P.......SSH-2.0-
0x0030   5375 6e5f 5353 485f 312e 300a                  Sun_SSH_1.0.
```

Fig 1 SSH Server response.

A snort rule to detect it can be written thus:

```
alert tcp any 22 -> any any (msg: "SSH server response"; \
            flow:established; \
            flowbits:isnotset,FIRST_PACKET_SEEN; \
            content: "SSH-"; depth:4; \
            flowbits:set,FIRST_PACKET_SEEN; )
```

Note that the flowbits label “FIRST_PACKET_SEEN” is entirely arbitrary (it is a convenient one in my opinion, but I am biased). So what does it do? Well, first it checks that the flow is in an established state. Then it proceeds to check that the attribute “FIRST_PACKET_SEEN” is not set (if it is, this flow has been dealt with before). The actual content check follows, and if all condition match, the “FIRST_PACKET_SEEN” attribute is added to the flow.

In order to detect SSH going on non-standard ports, the port number 22 should of course be prefixed with an exclamation mark.

6. SSL and TLS

A quick primer on SSL and TLS has been written by Holly Lynne McKinley as a GSEC practical [8], and is recommended reading for those without prior knowledge of those protocols. For the purpose of this document, the reader should be aware that there are different internal protocols. At the lowest level is the Record Protocol. Go up one level and we find the Handshake Protocol, which is responsible for setting up new sessions. For doing the session initiation, Hello messages are used (the data structures differ between the variants).

> *When a client first connects to a server it is required to send the client hello as its first message.*[8]

Great! Just what we needed. In the next three sections I will go through these variants and show the structures they are using (the specifications are not really quotation-friendly, so I will use my own style).

8 Dierks & Allen, p.33. [9]

Again, (as with SSH), it is the fourth packet in the TCP session that is of interest. Please be aware of the difference between SSH, though: In SSL/TLS, it is the client that will send the first data packet, not the server.

Note that I have written the version as a 16 bit hexadecimal number. In the specifications, it is actually composed of two separate octets, the first being the major number and the second the minor.

6.1. SSL version 2.0

A SSL version 2.0 [10] (I will henceforth refer to it as SSL v2) header is structured in the following way (only interesting portions are showed):

SSL version 2.0 header			
Byte offset	***Field***	***Value***	***Meaning***
0,1	Record Length	<variable>	
2	Message Type	0x01	Client Hello
3,4	Client Version	0x0002	SSL v2.0
5,6	Cipher Specification Length	<variable>	
9..	Cipher Specifications	<variable>	

A sample packet is showed in figure 2 (below).

```
IP client.org.1554 > server.net.443: P 1:46(45) ack 1

0x0000   4500 0055 4ec1 4000 8006 XXXX XXXX XXXX        E..UN.@.........
0x0010   XXXX XXXX 0612 01bb 024e 7541 45c7 9ee3        .........NuAE...
0x0020   5018 fc00 XXXX 0000 802b 0100 0200 1200        P........+......
0x0030   0000 1001 0080 0300 8007 00c0 0600 4002        ..............@.
0x0040   0080 0400 80a4 46ea 981a 400d b785 11c0        ......F...@.....
0x0050   188b 0b9c bf                                  .....
```

Fig 2 SSL v2 Client Hello.

The bytes 5 and 6 (from the beginning of the payload) tells us that the cipher specifications takes up 18 bytes. Each specification consists of three octets of data (for example, "`01 00 80`" equates to "`SSL_CK_RC4_128_WITH_MD5`"), which means that we should have 6 different specifications in this message. At offset 9 and 18 bytes onward, the various specifications included in the message are defined. In version 2, all valid alternatives have the middle octet set to zero. A quick comparison against that format shows the above packet to be correct in this regard. Unfortunately, we are unable to use the specifications for our rule writing, as they are variable (the reason this section is here will become apparent later).

Sadly, the string "`01 00 02`" in that particular position is not too much to go on. We need to make our signature more precise somehow, or we might drown in false positives. This can be achieved by reading through the specification and calculate

the maximum payload size. A specification compliant layer two Client Hello will have a maximum payload of 80 (for this to happen, all possible cipher specifications would have to be enabled, which is not very likely).

The snort rule could go along these lines:

```
alert tcp any any -> any !443 (msg: "SSL v2 client hello"; \
            flow:established; \
            dsize: < 81; \
            flowbits:isnotset,FIRST_PACKET_SEEN; \
            content: "|01 00 02|"; offset:2; depth:3; \
            flowbits:set,FIRST_PACKET_SEEN; )
```

6.2. SSL version 3.0 and TLS version 1.0

SSL version 3.0 [11] and TLS version 1.0 [9] (henceforth referred to as SSL v3 and TLS) both use the same header. The only difference is the version numbers (which is `0x0300` and `0x0301`, respectively).

SSL version 3.0 and TLS 1.0 header			
Byte Offset	***Field***	***Value***	***Meaning***
0	Record Content Type	0x16	Handshake
1,2	Protocol Version	0x0300	SSL v3.0
3,4	Record Length (bytes)	<variable>	
5	Handshake Type	0x01	Client Hello
6,7,8	Message Length (bytes)	<variable>	
9,10	Client Version	0x0300	SSL v3.0

The obligatory packet dump (figure 3):

```
IP client.org.1553 > server.net.443: P 1:121(120) ack 1

0x0000   4500 00a0 4e70 4000 8006 XXXX XXXX XXXX        E...Np@.........
0x0010   XXXX XXXX 0611 01bb 010f 5aa5 4114 af17        ..........Z.A...
0x0020   5018 fc00 XXXX 0000 1603 0000 7301 0000        P...........s...
0x0030   6f03 0000 009a bc55 cacd 611f 5f93 2919        o......U..a._.).
0x0040   4768 87ba 1ac5 4cd2 5b02 4992 c0a1 34aa        Gh....L.[.I...4.
0x0050   8c65 1720 47cb a5e4 5bd0 72ed b3f4 a069        .e..G...[.r....i
0x0060   d719 15e2 2a14 c697 65fa 9d30 d567 3cd2        ....*...e..0.g<.
0x0070   afaf 552c 0028 0039 0038 0035 0033 0032        ..U,.(.9.8.5.3.2
0x0080   0004 0005 002f 0016 0013 feff 000a 0015        ...../..........
0x0090   0012 fefe 0009 0064 0062 0003 0006 0100        .......d.b......
```

Fig 3 SSL v3 Client Hello.

As can be seen, this header is more complex than the one used by the previous version. This requires a more complex snort rule, but with the advantage that we are reducing the likelihood of false positives. Only the SSL v3 sample rule is displayed.

```
alert tcp any any -> any !443 (msg: "SSL v3 client hello"; \
            flow:established; \
            flowbits:isnotset,FIRST_PACKET_SEEN; \
            content: "|16 03 00|"; depth:3; \
            content: "|01|"; offset:5; depth:1; \
            content: "|03 00|"; offset:9; depth:2; \
            flowbits:set,FIRST_PACKET_SEEN; )
```

6.3. SSLv3 or TLS 1.0 in backward compatibility mode

Both SSL v3 and TLS can be backwards compatible with SSL v2 This is done by using a SSL v2 Client Hello message with the version number of `0x0300` or `0x0301`, respectively. The higher number of possible cipher specifications means the maximum payload will be 170 or 161 bytes (TLS has fewer specifications than SSL v3).

A modified snort rule could look thus (this time, the TLS sample is showed):

```
alert tcp any any -> any !443 (msg: "TLS backwards compatible ClientHello"; \
            dsize: < 162; \
            flow:established; \
            flowbits:isnotset,FIRST_PACKET_SEEN; \
            content: "|01 03 01|"; offset:2; depth:3; \
            flowbits:set,FIRST_PACKET_SEEN; )
```

SSL v3 would of course use “dsize: < 171” instead of 162.

7. Running a Snort Ruleset

My theories naturally had to be put to the test. A sniffer running snort was put on a network segment just between the corporate firewall (at an undisclosed site) and the upstream router. The ruleset used is described in the next section (for brevity, the complete ruleset is not included; however, the rules presented in the previous sections are used without other alterations than different port numbers or rule actions).

7.1. The ruleset

First, in order to use pass rules to ignore traffic we know is good (or rather do not care about in this context), it is necessary to reconfigure the action order:

```
config order: pass alert log activation dynamic
```

Then, of course, we need the stateful capability and the preprocessors that use it:

```
config stateful
preprocessor flow
preprocessor stream4
```

As for the rules, it begins with a rule for alerting on SSH traffic coming on non-standard ports (i.e. anything except port 22). A rule for SSH on the standard port is also active, mostly as a reference.

Then there are three rules to alert on SSL versions 2 and 3, plus TLS. Two additional ones handle backward compatible Client Hello's for TLS and SSL v3. These five rules only trigger on ports different from 443.

The following five rules are identical to the previous ones except for the facts they are monitoring port 443 only, and they are passing the traffic instead of alerting (there was a lot of encrypted web traffic on that port, which had to be excluded).

The final rule (shown below) is the reason for the changed config order. It will alert on anything on port 443 that has not been passed by the previous rules. Without being able to lock it down to the first packet after TWH, this rule would have generated too many false positives to have been even remotely useful.

```
alert tcp any any -> any 443 (msg: "Non TLS/SSL on TLS/SSL port"; \
                      !dsize:0; \
                      flow:established,to_server; \
                      flowbits:isnotset,FIRST_PACKET_SEEN; \
                      flowbits:set,FIRST_PACKET_SEEN; )
```

7.2. Some SSL v2 false positives

As I had feared, the static content of the SSL v2 header was not specific enough, the pattern may occur in non-SSL traffic as well.

```
[**] [1:0:0] SSL v2 client hello [**]
03/06-13:00:33.849067 web server.org:80 -> client.com:1432
TCP TTL:254 TOS:0x0 ID:6050 IpLen:20 DgmLen:116 DF
***A**** Seq: 0xAC50B003  Ack: 0x3BE76F  Win: 0x2398  TcpLen: 20
```

The packet triggering the alert is displayed in figure 4.

```
IP web server.org.80 > client.com.1432: . 2890969091:2890969167(76) ack 3925871
win 9112 (DF)

0x0000   4500 0074 17a2 4000 fe06 XXXX XXXX XXXX        E..t..@.........
0x0010   XXXX XXXX 0050 0598 ac50 b003 003b e76f        .....P...P...;.o
0x0020   5010 2398 XXXX 0000 0d61 0100 0211 0304        P.#......a......
0x0030   2112 3105 4151 6113 2271 8132 0614 91a1        !.1.AQa."q.2....
0x0040   b142 2324 1552 6233 34c1 7282 4307 2592        .B#$.Rb34.r.C.%.
0x0050   0853 d1f0 6373 3516 e1a2 f1b2 8326 4493        .S..cs5......&D.
0x0060   5464 45c2 a374 3617 18d2 55e2 65f2 b384        TdE..t6...U.e...
0x0070   c3d3 75e3                                      ..u.
```

Fig 4 False SSLv2 packet.

Well, that packet does not really look like a SSL v2 header. I see no trace of any cipher specifications, and the payload is larger than usual for SSLv2 packets (of course, knowing that web server.org is a legitimate web server also helps the analysis, but the most telling fact is undoubtedly the absence of the cipher specifications). A couple more of these appears in the alert log, but they are not too frequent, just a minor annoyance.

7.3. SSH traffic

A bunch of alerts came within a very short interval (a subset being showed below):

```
[**] [1:0:0] SSH server response [**]
03/08-04:54:44.763433 172.16.1.222:22 -> obvious.scanner:38711

[**] [1:0:0] SSH server response [**]
03/08-04:54:44.815555 172.16.1.231:22 -> obvious.scanner:38720

[**] [1:0:0] SSH server response [**]
03/08-04:54:44.822176 172.16.1.232:22 -> obvious.scanner:38721

[**] [1:0:0] SSH server response [**]
03/08-04:54:44.826248 172.16.1.242:22 -> obvious.scanner:38731
```

Nothing very exciting, seems to be a normal portscan targeting systems at the corporate network. We do have the TWH (it must be completed for this rule to trigger), so it is probably not spoofed. From looking at the source port numbers of the scanner (they are increasing over time), I would hazard a guess that a common TCP Connect scan is being done, targeting port 22 only. Notice that the complete network is being scanned, with destination IP incrementing sequentially. Not too clever, and easy to detect (hey, that signature was not even built for detecting port scans!).

There is an extra bonus, though. While looking at the payload with tools as tcpdump or Ethereal, we can see the SSH server version right after the initial “SSH-” string (please refer to figure 1 for a sample). The four responses yielded two different versions of OpenSSH: 3.4p1 and 3.7.1p2. Why is this interesting? Well, according to the security page on the OpenSSH website [12], all versions prior to 3.7.1p2 has one or more security vulnerabilities. In other words, we have vulnerable remote access protocols running at “our” site, and all inhabitants on the Internet seems to be free to connect to them at will. Not very cool at all.

The next SSH-related alert looked like this:

```
[**] [1:0:0] SSH server response on non-standard port [**]
03/08-09:41:18.394286 large_netblock.isp:2214 -> client.org:1266
TCP TTL:51 TOS:0x0 ID:43435 IpLen:20 DgmLen:65 DF
***AP*** Seq: 0xAF306541  Ack: 0xB6FD07FA  Win: 0x16D0  TcpLen: 20
```

Obviously, someone is running SSH from the internal network to an external SSH server on the Internet, and it is listening on port 2214. That might be worth looking into. It could be a compromise, but it would not surprise me if it is someone connecting to their home computer during work hours.

7.4. Something sneaky

Snort picked up the following.

```
[**] [1:0:0] Non TLS/SSL on TLS/SSL port [**]
03/08-11:00:38.541282 client.com:52374 -> server.net:443
TCP TTL:115 TOS:0x0 ID:13497 IpLen:20 DgmLen:46 DF
***AP*** Seq: 0xDA5B044C  Ack: 0xF82BD099  Win: 0xFBFA  TcpLen: 20
```

The offending packet is showed in figure 5.

```
IP client.com.52374 > server.net.443: P 3663397964:3663397970(6) ack 4163621017
win 64506 (DF)

0x0000   4500 002e 34b9 4000 7306 5891 XXXX XXXX        E...4.@.s.......
0x0010   XXXX XXXX cc96 01bb da5b 044c f82b d099        .........[.L.+..
0x0020   5018 fbfa XXXX 0000 7f7f 4943 4100             P.........ICA.
```

Fig 5 Suspicious packet on port 443 (interesting parts highlighted in red)

Now, that packet really should be SSL or TLS, but obviously it is not. More of these packets are seen throughout the capture (which spanned over a couple of days). They do not occur frequently, just a few times for each source host and day (more sessions were seen from other sources but to the same destination). The payload did always consist of the hexadecimal sequence visible in the capture ("`7f7f 4943 4100`"), so it can probably be used for a new rule to detect this kind of traffic.

The string "`ICA`" corresponding to the hexadecimal value of "`49 43 41`" is a good clue. MetaFrame [13] is quite a nice remote desktop product from Citrix, and the protocol they use is called ICA (Independent Computing Architecture). The port usually associated with this traffic is 1494 (it is registered with IANA), but I would guess there is some kind of firewall-evading business going on here.

8. Conclusion

Encrypted network traffic is a challenge for intrusion detection and should be deployed only in a controlled manner. This is of course a policy issue more than anything else, but if controls are in place, it becomes much easier decide whether encrypted data is acceptable or not ("hey, there should be no SSL going coming from that network!"). It does not help if we can detect SSL unless we also can decide whether the traffic is legitimate or not.

Even though only a specific service is expected (as listed by IANA) on a certain port, there is no guarantee someone does not deploy another service through it. An encrypted service may be used on a port not usually associated with encrypted data or vice versa. A NIDS such as Snort can be used to detect such anomalies.

The "normal" (if there is such a thing) usage for Snort is being a signature based NIDS, or possibly a sniffer. The purpose in these cases is to detect specific attacks. However, if a signature can be written for a protocol, Snort can also easily be used to detect that protocol on unusual ports. It can of course also be used to detect anything except the, for a specific port, expected protocols. This is perhaps more of a network policy appliance than an outright intrusion detection one.

For the really sneaky and patient attacker, the discussed measures will not be enough. If someone did the extreme and tunnelled TCP over HTTP over TLS on port 443, none would be the wiser.

While it is certainly possible to write Snort rules to detect encryption carriers, it is a bit awkward. A specific crypto preprocessor would be really welcome in the future (it would probably be a lot faster than the rules I wrote). Another thing I think Snort could benefit from is the possibility of more complex rules (I would absolutely love to be able to use logic operators). This would eliminate the need for the "pass" rules I used to detect anything except the known good traffic. Oh, and while I am at it. Pure not rules would be even nicer (i.e. to prepend an exclamation mark before the content check to match anything except the specified content).

In his GSEC practical about IDS evasion, Corbin Del Carlo writes:

> *To protect against an attacker encrypting their commands, future NIDSs should alert if they see any outbound encrypted session from any host that does not normally conduct encrypted sessions, as well as any large number of inbound session initiations that would be typical of a brute force password guessing attack over SSH or an SSL VPN.*[9]

A crypto preprocessor could possibly make anomaly detection engines such as SPADE[10] more accurate, if they could be made to cooperate.

9 Del Carlo, p.6. [14]
10 SPADE is a Snort Preprocessor – Statistical Packet Anomaly Detection Engine [15]

To conclude my findings, I would say that it is indeed possible to detect encrypted traffic by looking at the carrying protocols. It is my assumption that other carriers can be identified and put into rules by the same method I used (i.e. by reading the specifications). And as usual, a human mind is still required to tell false positives from real events. There are still no replacements for trained analysts.

9. References

[0] ”Snort”. URL: http://www.snort.org/ (25 Jun 2004)

[1] Reynolds, Joyce K (editor). ”Assigned Numbers: RFC 1700 is replaced by an On-Line Database”. Jan 2002. URL: http://www.ietf.org/rfc/rfc3232.txt (25 Jun 2004)

[2] IANA. ”TCP and UDP Port Numbers” 19 Mar 2004.
URL: http://www.iana.org/assignments/port-numbers (25 Jun 2004)

[3] Bace, Rebecca Gurley. ”Intrusion Detection”. Macmillan Technical Publishing.
2000. ISBN 1-57870-185-6

[4] Ptacek, Thomas H & Newsham, Timothy N. ”Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”. Jan 1998.
URL: http://www.snort.org/docs/idspaper/ (25 Jun 2004)

[5] Postel, Jon (editor). ”Transmission Control Protocol”. Sep 1981.
URL: http://www.ietf.org/rfc/rfc0793.txt (25 Jun 2004)

[6] The Snort Project. ”SnortUsers Manual” 12 May 2004.
URL: http://www.snort.org/docs/snort_manual.pdf (24 Jun 2004)

[7] Ylonen, Tatu & Lonvick, C (editor). ”SSH Transport Layer Protocol”. Jun 2004.
URL: http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-18.txt (25 Jun 2004)

[8] McKinley, Holly Lynne. “SSL and TLS: A beginner's guide”. 2003.
URL: http://www.sans.org/rr/papers/44/1029.pdf (25 Jun 2004)

[9] Dierks, T & Allen, C. ”The TLS Protocol Version 1.0”. Jan 1999.
URL: http://www.ietf.org/rfc/rfc2246.txt (25 Jun 2004)

[10] Hickman, Kipp E.B. ”SSL 2.0 PROTOCOL SPECIFICATION”. 9 Feb 1995.
URL: http://wp.netscape.com/eng/security/SSL_2.html (25 Jun 2004)

[11] Freier, Alan O. et al ”The SSL Protocol Version 3.0”. 18 Nov 1996.
URL: http://wp.netscape.com/eng/ssl3/draft302.txt (25 Jun 2004)

[12] ”OpenSSH Security”. URL: http://www.openssh.org/security.html (25 Jun 2004)

[13] ”Citrix Remote Office Connectivity Solutions for the On-Demand Enterprise”.
URL: http://www.citrix.com/site/PS/solutions/solution.asp?solutionID=1408 (25 Jun 2004)

[14] Del Carlo, Corbin. ”Intrusion detection evasion: How Attackers get past the burglar alarm”.
25 Sep 2003. URL: http://www.sans.org/rr/papers/index.php?id=1284 (25 Jun 2004)

[15] Hoagland, James & Staniford, Stuart. “SPICE / SPADE”.
URL: http://www.silicondefense.com/software/spice/index.htm (23 Mar 2004)