Covering the global threat landscape

ALTERNATIVE COMMUNICATION CHANNEL OVER NTP

Nikolaos Tsapakis
Independent researcher

1. INTRODUCTION

In this article I will explore Network Time Protocol (NTP) as an alternative communication channel. Although there are already some interesting publications on this topic [1, 2], this article will provide practical examples, code, and the basic theory behind the idea.

2. COVERT CHANNELS

In computer security, a covert channel is a type of attack that creates the ability to transfer information objects between processes that, according to the computer security policy, are not supposed to be allowed to communicate [3].

An example of such a communication channel is DNS, which has been abused by malware in order to hide network traffic between client (infected computer) and server (command and control). One example (among many) of malware that uses DNS as a covert channel is the Denis malware family [4].

Another example of such a communication channel is ICMP, which has also been abused in the past by malware [5].

The NTP protocol may also be used to carry data and, as such, it may be open to exploitation in a similar way to DNS and ICMP.

To widen the idea, any protocol that provides space for data may be used as a starting point for similar exploitation as long as both client and server follow the format specification and avoid inconsistencies.

3. BASIC NTP COMMUNICATION

The Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks [6].

I will generate a basic NTP request and response on a WinOS machine and explain what such basic communication represents. Assuming that the clock of the client has a date of 3 March 2019, clicking 'Update Now' in the Internet Time Settings will result in the client sending an NTP request to the time server. Such a default server would be time.windows.com. The server will reply with an NTP message which contains time measurements. Those measurements will drive the clock synchronization.

```
> Request
Network Time Protocol (NTP Version 3, client)
  Flags: 0xdb, Leap Indicator: unknown (clock
unsynchronized),
     Version number: NTP Version 3, Mode: client
  Peer Clock Stratum: unspecified or invalid (0)
  Peer Polling Interval: 10 (1024 sec)
  Peer Clock Precision: 0.015625 sec
  Root Delay: 0.198867797851563 seconds
  Root Dispersion: 8.80982971191406 seconds
  Reference ID: NULL
  Reference Timestamp: Mar 3, 2019 18:54:23.652902999 UTC
  Origin Timestamp: Jan 1, 1970 00:00:00.000000000 UTC
  Receive Timestamp: Jan 1, 1970 00:00:00.000000000 UTC
  Transmit Timestamp: Mar 3, 2019 18:55:13.480902999 UTC

> Request (in hex)
db 00 0a fa 00 00 32 e9 00 08 cf 51 00 00 00 00
e0 26 a1 5f a7 24 a6 a8 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 e0 26 a1 91 7b 1c 75 81

> Response
Network Time Protocol (NTP Version 3, server)
  Flags: 0x1c, Leap Indicator: no warning, Version
number: NTP Version 3,
     Mode: server
     00.. .... = Leap Indicator: no warning (0)
     ..01 1... = Version number: NTP Version 3 (3)
     .... .100 = Mode: server (4)
  Peer Clock Stratum: secondary reference (2)
  Peer Polling Interval: 10 (1024 sec)
  Peer Clock Precision: 0.000000 sec
  Root Delay: 0.131500244140625 seconds
  Root Dispersion: 0.0191650390625 seconds
  Reference ID: 128.138.141.172
  Reference Timestamp: Mar 6, 2019 18:51:40.408170399 UTC
  Origin Timestamp: Mar 3, 2019 18:55:13.480902999 UTC
  Receive Timestamp: Mar 6, 2019 18:55:05.080168299 UTC
  Transmit Timestamp: Mar 6, 2019 18:55:05.080171299 UTC

> Response (in hex)
1c 02 0a e9 00 00 21 aa 00 00 04 e8 80 8a 8d ac
e0 2a 95 3c 68 7d da f7 e0 26 a1 91 7b 1c 75 81
e0 2a 96 09 14 85 e8 e2 e0 2a 96 09 14 86 1b 37
```

In the basic operation of the protocol a client sends a packet to a server and records the time the packet left the client in the Origin Timestamp field (T1).

T1 = Mar 3, 2019 18:55:13.480902999 UTC (as seen in response).

The server records the time the packet was received in the Receive Timestamp (T2).

T2 = Mar 6, 2019 18:55:05.080168299 UTC (as seen in response).

A response packet is then assembled with the original Origin Timestamp and the Receive Timestamp equal to the packet receive time, and then the Transmit Timestamp is set to the time at which the message is passed back toward the client (T3).

T3 = Mar 6, 2019 18:55:05.080168299 UTC (as seen in response).

The client then records the time the packet arrived (T4), giving the client four time measurements. These four parameters are passed into the client's timekeeping function to drive the clock synchronization function [7].

Note that the Destination Timestamp field is not included as a header field; it is determined upon arrival of the packet and made available in the packet buffer data structure [8].

4. NTP V4 MESSAGE FORMAT

Figure 1 shows the packet format for NTP v4 [8].

Figure 2 shows the NTP timestamp format used for Timestamp fields [8].

The format may be compared with the generated messages shown in Section 3.

Notice that the Extension Field is missing from the generated messages in Section 3. This is an interesting field, which is described as follows [8]:

'In NTPv4, one or more extension fields can be inserted after the header and before the MAC, which is always present when an extension field is present.'

This means that I should be able to add an Extension field and place data for tranfer to the server inside the Extension Field. I should also add a message authentication code (MAC) consisting of the Key Identifier field and Message Digest (dgst) field.

Figure 3 shows the Extension Field format [9].

Figure 4 shows the NTP Extension Field Type format and values [9].

Taking into consideration the above format, I will construct an NTP v4 message and send it to a publicly available time server. The server should be able to reply. Both messages should be visible in

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|LI | VN  |Mode |    Stratum     |     Poll      |  Precision   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Root Delay                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Root Dispersion                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Reference ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                     Reference Timestamp (64)                  +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                      Origin Timestamp (64)                    +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                      Receive Timestamp (64)                   +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                      Transmit Timestamp (64)                  +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                    Extension Field 1 (variable)               .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                    Extension Field 2 (variable)               .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Key Identifier                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                            dgst (128)                         |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Packet format for NTP v4.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Seconds                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Fraction                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: NTP timestamp format used for Timestamp fields.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Field Type           |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                                                               .
.                            Value                              .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Padding (as needed)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Extension Field format.

```
0                   1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+---------------+---------------+
|R|E|       Code |        Type   |
+-------------------------------+

+------------+--------------------------------------------+
| Field Type |  Meaning                                   |
+------------+--------------------------------------------+
|   0x0000   | crypto-NAK (with Field Length of 0)        |
|   0x0000   | RESERVED: Permanently Unassigned           |
|   0x0001   | RESERVED: Unassigned                       |
|   0x0002   | Autokey: No-Operation Request              |
|   0x8002   | Autokey: No-Operation Response             |
|   0x0102   | Autokey: Association Message Request       |
|   0x8102   | Autokey: Association Message Response      |
|   0x0202   | Autokey: Certificate Message Request      |
|   0x8202   | Autokey: Certificate Message Response     |
|   0x0302   | Autokey: Cookie Message Request            |
|   0x8302   | Autokey: Cookie Message Response           |
|   0x0402   | Autokey: Autokey Message Request           |
|   0x8402   | Autokey: Autokey Message Response          |
|   0x0502   | Autokey: Leapseconds Value Message Request |
|   0x8502   | Autokey: Leapseconds Value Message Response |
|   0x0602   | Autokey: Sign Message Request              |
|   0x8602   | Autokey: Sign Message Response             |
|   0x0702   | Autokey: IFF Identity Message Request      |
|   0x8702   | Autokey: IFF Identity Message Response     |
|   0x0802   | Autokey: GQ Identity Message Request       |
|   0x8802   | Autokey: GQ Identity Message Response      |
|   0x0902   | Autokey: MV Identity Message Request       |
|   0x8902   | Autokey: MV Identity Message Response      |
|   0x0005   | Checksum Complement                        |
|   0x2005   | Checksum Complement (deprecated flag 0x2000) |
+------------+--------------------------------------------+
```

Figure 4: NTP Extension Field Type format and values.

my *Wireshark* session. Neither message should break the protocol format.

5. CONSTRUCTING NTP V4 REQUEST

Taking into consideration the protocol format and the generated NTP traffic from Section 3, I wrote a Python script which constructs NTP v4 requests and sends them to a publicly available server. After sending a request, the script waits for a reply. After a reply arrives, the script sleeps for a few seconds and then sends the next request. All requests are similar. The script generates random data for the Extension Field and MAC fields. The size of data in the Extension Field is also random from a minimum to a maximum limit. The script is shown in the Appendix. Comments make it easier to understand the script.

6. TESTING MULTIPLE TIME SERVERS

I tested the script against multiple time servers [10]. Some of them did not respond at all, while others responded. An increase in the data size in the Extension Field (1K and above) may cause some servers to stop responding after sending a few requests. A decrease in sleep among requests may also cause some servers to stop responding after sending a few requests.

Overall, it appears that the messages were arriving at the servers without breaking the protocol and without being blocked, since the servers were responding.

This indicates that such generated network traffic can be used for tranferring data from source (client) to destination (custom server) without being blocked. And for the case of a custom server, responses would continue to be sent down to the client, since the custom server configuration which serves the purposes of the client would apply.

7. DETECTION

The following is the Snort rule that would detect NTP messages like those generated by the script in Section 5. In case the monitoring system is a gate/proxy, the rule should be modified and applied for both incoming and outgoing traffic. Note that the rule was tested on a VM and the script was generating traffic with bad checksums. I used the following command to start Snort on test system:

```
snort -i 1 -c c:\Snort\etc\snort.conf -A console
-k none
```

Notice the -k option, which means that Snort will process the packets with invalid checksums.

```
alert udp any any -> any 123 (dsize:>68; content:"|01
00|";
depth:2; offset:48; msg:"Suspicious Incoming NTP
packet"; sid:1000005;)
```

- Detects incoming messages to destination port 123.
- The size of the message should be more than 68 bytes. The reason for this is that a standard NTP message is 48 bytes long and in case an extension field is present, MAC (here 20 bytes long) is mandatory as per [8] (section 7.5). This results in a minimum size of 68 bytes.
- Byte values 0x01 and 0x00 should exist in offsets 48, 49. Those are ext_r_e_version and ext_opcode values. That particular WORD combination is defined as 'RESERVED: Unassigned' [9], but it may not really mean anything in the case of a covert channel.

As it may be easily understood, this rule *should* be changed in case different values are used. In such a case an investigation should be performed to make sure traffic does not originate from a network that uses the NTP extension field for legitimate purposes, for example AutoKey [9].

8. A CUSTOM PROTOCOL INSIDE NTP

Taking into consideration that NTP is over UDP [11] and that data should be split into multiple segments (requests), additional rules would apply for proper communication on top of NTP. Some of these rules could be the following:

- Messages of the custom protocol may exist inside the Extension Field or even in MAC.
- There must be an integrity check for Extension Field data.
- There must be a sequence number due to multiple segments.
- Start/End characters must exist for defining the start/end of stream. For example, assume that a stream is a file being uploaded in multiple segments. That is, multiple NTP requests.
- Client would request server to ACK requests.
- Server would request client to ACK responses.
- There must be encryption of Extension Field data and MAC data.

9. CONCLUSION

NTP protocol is an interesting case to use as a communication channel because it is widely used, well documented and allowed to flow through networks without applying strict rules. Once again, any protocol that provides space for data may be used as a starting point for similar exploration as long as both client and server follow the specification and avoid inconsistencies.

10. REFERENCES

[1] https://ricklahaye.info/projects/ntp.pdf.

[2] https://www.researchgate.net/publication/316287783_Covert_Channel_over_Network_Time_Protocol.

[3] https://en.wikipedia.org/wiki/Covert_channel.

[4] https://securelist.com/use-of-dns-tunneling-for-cc-communications.

[5] https://blog.trendmicro.com/trendlabs-security-intelligence/phishing-trojan-uses-icmp-packets-to-send-data/.

[6] https://en.wikipedia.org/wiki/Network_Time_Protocol.

[7] https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-58/154-ntp.html.

[8] https://tools.ietf.org/html/rfc5905.

[9] https://tools.ietf.org/html/draft-stenn-ntp-extension-fields-08.

[10] https://www.ntppool.org/zone.

[11] https://en.wikipedia.org/wiki/User_Datagram_Protocol.

Editor: Martijn Grooten
Head of Testing: Peter Karsai
Security Test Engineers: Gyula Hachbold, Adrian Luca, Csaba Mészáros, Tony Oliveira, Ionuţ Răileanu
Sales Executive: Allison Sketchley
Editorial Assistant: Helen Martin

© 2019 Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England
Tel: +44 (0)1235 555139 Email: editor@virusbulletin.com
Web: https://www.virusbulletin.com/

APPENDIX

```
------#------#------#------<START CODE>------#------#------#------
import socket
import sys
import random
import time
import datetime

def main():

       try:
               s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
       except socket.error:
               print 'Failed to create socket'
               sys.exit()

       host = '2.gr.pool.ntp.org'
       port = 123

       while(True) :
               # 0xE3 = 11 100 011b
               # li   = 11b  = 3 => Clock is unsynchronized
               # vn   = 100b = 4 => Version 4
               # mode = 011b = 3 => Client
               li_vn_mode = chr(0xE3)

               # Unsynchronized
               stratum    = chr(0x10)

               # Suggested default limits for minimum and maximum poll intervals
        # are 6 and 10, respectively [4].
               poll = chr(0x06)

               # Random, selected in such a way so that it has a small
        # value < 1 sec and appears realistic.
               precision = chr(random.randint(236,254))

               # Random, selected in such a way so that it has a small
        # value < 1 sec and appears realistic.
               root_delay = chr(0x00) +\
                                          chr(0x00) +\
                    chr(random.randint(1,9)) +\
                                          chr(random.randint(1,254))

               # Random, selected in such a way so that it has a small
        # value < 1 sec and appears realistic.
               root_dispersion = chr(0x00) +\
                                                  chr(0x00) +\
                         chr(random.randint(1,9)) +\
                         chr(random.randint(1,254))

               # No reference
               reference_id = chr(0x00) +\
                                            chr(0x00) +\
                             chr(0x00) +\
                             chr(0x00)

 # This is a zero NTP timestamp, assigned later.
 zero_timestamp = chr(0x00) +\
                                    chr(0x00) +\
                                    chr(0x00) +\
                                    chr(0x00) +\
                                    chr(0x00) +\
                                    chr(0x00) +\
                                    chr(0x00) +\
                                    chr(0x00)

 # Get current datetime as NTP timestamp
 diff = datetime.datetime.utcnow() -\
datetime.datetime(1900, 1, 1, 0, 0, 0)

 timestamp = diff.days*24*60*60+diff.seconds
 timestamp_1 = timestamp & 0xff000000
 timestamp_1 = timestamp_1 >> 24
 timestamp_2 = timestamp & 0x00ff0000
 timestamp_2 = timestamp_2 >> 16
 timestamp_3 = timestamp & 0x0000ff00
 timestamp_3 = timestamp_3 >> 8
 timestamp_4 = timestamp & 0x000000ff

 # Set to current timestamp minus a small value.
 # Keeping it same among packets.
 reference_timestamp = chr(timestamp_1) +\
                                             chr(timestamp_2) +\
               chr(timestamp_3) +\
                                             chr(0x00) +\
               chr(0x7D) + chr(0x21) +\
                                             chr(0x73) + chr(0x83)

 # Set to zero timestamp, since I notice this value been
 # set by default when I try to sync time from a Win OS.
 originate_timestamp = zero_timestamp

 # Set to zero timestamp, since I notice this value been
 # set by default when I try to sync time from a Win OS.
 receive_timestamp = zero_timestamp

 # Set to current timestamp
 transmit_timestamp = chr(timestamp_1) +\
                                            chr(timestamp_2) +\
                                            chr(timestamp_3) +\
                                            chr(timestamp_4) +\
              chr(random.randint(1,254)) +\
                                            chr(random.randint(1,254)) +\
                                            chr(random.randint(1,254)) +\
                                            chr(random.randint(1,254))

 # Using unassigned value for basic extension field format.
 #
```

```
        # Field Type = [ext_opcode, ext_r_e_version] =>
# [0x00, 0x01] = 0x0001 = RESERVED: Unassigned
        #
        # ext_r_e_version = 00000001b =>
        # R = 0 (Request),
        # E = 0 (OK),
        # Version = 000001
        #
        # ext_opcode = 0x00
        ext_r_e_version = chr(0x01)
        ext_opcode = chr(0x00)

        # Max size of data for extension field
        max_size = 500

        # Min size of data for extension field
        min_size = 300

        size = random.randint(min_size,max_size)
        data = ''

        for x in range(size):
                data = data + chr(random.randint(1,254))

        # Padding extension data
        pad = size % 4
        pad = 4 - pad
        size = size + pad + 4

        padding = ''

        while(pad!=0):
                padding = padding + chr(0x00)
                pad = pad - 1

        a = size & 0x0000ffff
        b = a & 0x000000ff
        c = a & 0x0000ff00
        c = c >> 8

        ext_len  = chr(c) + chr(b)
        ext_data = data + padding

        # Random since I am not actually going to use it.
# Just making sure NTP message format does not break.
        key_id = chr(random.randint(1,254)) +\
                          chr(random.randint(1,254)) +\
                          chr(random.randint(1,254)) +\
                          chr(random.randint(1,254))

        # Random since I am not actually going to use it.
# Just making sure NTP message format does not break.
        digest = chr(random.randint(1,254)) +\
                          chr(random.randint(1,254)) +\
                          chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254)) +\
                              chr(random.randint(1,254))

                ext = ext_r_e_version + ext_opcode + ext_len + ext_data

                # Create the NTP packet
                msg = li_vn_mode +\
                           stratum +\
                           poll +\
                           precision +\
                           root_delay +\
                           root_dispersion +\
                           reference_id +\
                           reference_timestamp +\
                           originate_timestamp +\
                           receive_timestamp +\
                           transmit_timestamp +\
                           ext +\
                           key_id +\
                           digest

                print 'Message is : ' + msg

                try :
                         # Send the NTP packet
                         s.sendto(msg, (host, port))

                         # Wait for reply from sever
                         d = s.recvfrom(1024)
                         reply = d[0]
                         addr = d[1]
                         print 'Server reply : ' + reply
                         time.sleep(random.randint(1,3))

                except socket.error, msg:
                         print 'Error Code : ' + str(msg[0]) + ' Message ' + msg[1]
                         sys.exit()

if __name__ == "__main__":
    main()
------#------#------#------<END   CODE>------#------#------#------
```