Detecting Spoofed Time in NTP Traffic

Tomas Cejka[1], Alejandro Robledo[2]
[1]CESNET, a.l.e.
Zikova 4, 160 00 Prague 6
Czech Republic
[2]FIT, CTU in Prague
Thakurova 9, 160 00 Prague 6
Czech Republic
cejkat@cesnet.cz, robleale@fit.cvut.cz

Abstract. Almost every device connected into a computer network uses its own system time. In order to maintain precise system time, various time synchronization protocols are used. Such protocols allow for automatic adaptation of system time to keep it precise as much as possible. This paper deals with detection of possible exploit of vulnerability of the mostly used Network Time Protocol (NTP). Using spoofed NTP messages, an attacker is able to modify the system time of victims. Bad system time might lead to crucial security threats such as usage of already-expired certificated or cache poisoning or clearing.

Keywords. NTP, flow-based analysis, anomaly detection, NEMEA, network security

1 Introduction

Time synchronization is important task to keep infrastructure of network devices working. Currently, in the Internet of Things era, there are lots of devices connected into network and therefore it is impossible to maintain system time manually. As a solution, there are protocols for automatic adjusting of system time of devices.

One of the mostly used protocols for time synchronization is Network Time Protocol (NTP) [3]. Unfortunately, NTP is usually used in an insecure way — without any authentication or encryption. That makes NTP vulnerable and as it was published in [2], this vulnerability can be exploited by attackers.

Fig. 1 shows a typical example of an NTP attack. The blue points show the current victim's system time (measured in discrete timestamps — x-axis). The red points show observed offsets of NTP time that is received by a victim. When an attack starts, the offset gets significantly higher, however, it takes a few NTP exchanges before a victim adapts its system time. When the attack is successful, the victim's system time jumps to the time proposed by the attacker and the offset returns to normal (low) values, i.e. time is synchronized.

Goals of this work are a verification that attacks presented in [2] are possible and a proposal of a detection module that could report such activities. Using virtual machines, the attacks were repeated and we can approve that the victim's system time can be modified by an attacker under special conditions. Having the traffic from the simulation, we have designed a detection method that can be inserted and run in a monitoring infrastructure.

A typical monitoring system infrastructure is shown in Fig. 2. It is based on Monitoring probes which are exporting flow records, i.e. aggregated information about network traffic. Flow records are passed for storage (Collector) and analysis (Network Measurements Analysis

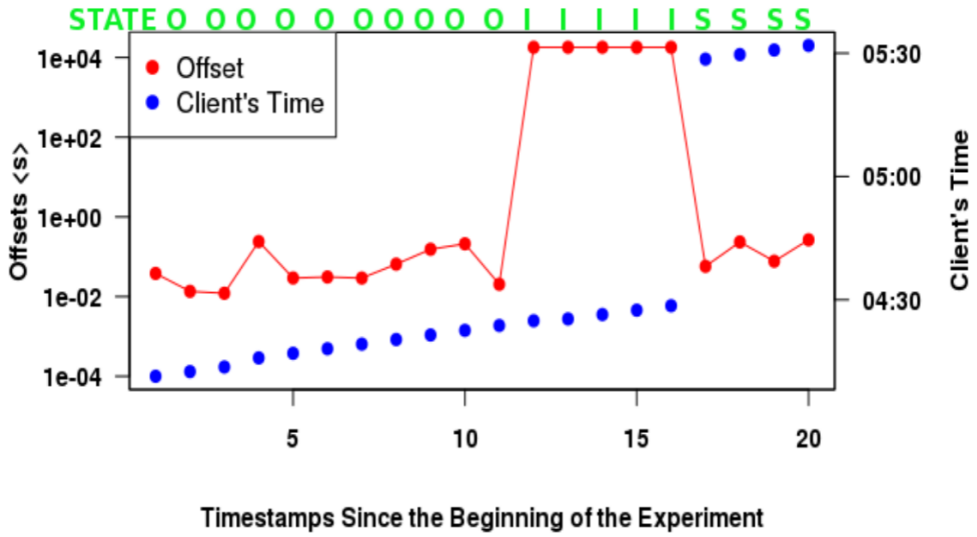

Figure 1: The example of an attack that manipulates with victim's system time.

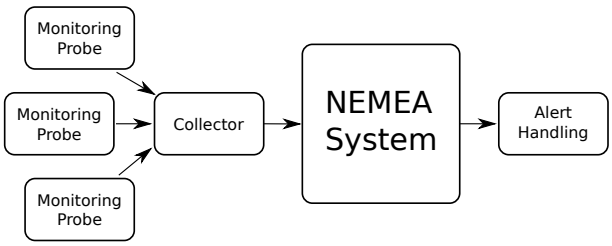

Figure 2: Monitoring system infrastructure.

(NEMEA) [1]). Detection modules of NEMEA produce alerts in a unified format suitable for subsequent processing and storage.

In our work, a new detection module `NTP_Attack_Detector.py` was added into NEMEA. The detection module requires additional information in exported flow records. Therefore, the experimental flow exporter called `flow_meter`[1] was extended by an NTP plugin. The plugin exports new fields that are listed in Tab. 1.

`Flow_meter` can be used on local networks, since it is not optimized for processing huge traffic. However, we can assume a small volume of NTP traffic and therefore the detector can be deployed with `flow_meter` in practice. Exported flow records containing NTP information are analyzed by the detector.

2 Detection Module

The detection module implements two approaches of data analysis.

2.1 Offset Analysis

The first approach is based on observing an offset that can be computed from timestamps of sent and received messages by server and client. The offset is computed using the following

[1] `https://github.com/CESNET/Nemea-Modules/tree/master/flow_meter`

Table 1: List of new NTP fields exported by `flow_meter` plugin.

UniRec Fields for NTP	**Description**
NTP_LEAP	Leap Indicator
NTP_VERSION	NTP Version
NTP_MODE	Mode 3 Request/Mode 4 Response
NTP_STRATUM	Stratum
NTP_POLL	Poll Interval
NTP_REF_ID	Reference ID
NTP_ORIG	Origin Timestamp
NTP_RECV	Receive Timestamp
NTP_SENT	Transmit Timestamp

equation:

$$\theta = \frac{1}{2}\Big[(T_2 - T_1) + (T_3 - T_4)\Big] \tag{1}$$

where the timestamps T_n are defined as follows: in T_1 client sends mode 3 request, in T_2 server receives mode 3 request, in T_3 server sends mode 4 response, in T_4 client receives mode 4 response. The equation is explained in more detail in [4].

Having a sequence of offsets computed with (1), the detection algorithm uses a threshold. When an offset reaches the threshold, an alert is reported. The pseudocode of timestamps capture, offset computation and checking the threshold is shown in Algorithm 1, where E is one NTP exchange of messages.

Algorithm 1 NTP_Offset (**in**: E[1,...,n],**out**:$\theta[1,...,n]$)

Require: $NTPExchangesE_i = \{PacketMode3[i], PacketMode4[i]\}, i\epsilon\{1,...,n\}$
  for all E_i i:=[1,...,n] **do**
    if $i > 1$ **then**
      $T4 \leftarrow PacketMode3[i].ReceiveTimestamp$
      $\theta[i-1] \leftarrow CalculateOFFSET(T1, T2, T3, T4)$
      if $\theta[i-1] > Threshold$ **then**
        $GenerateALERT()$
      end if
    else
      $T1 \leftarrow PacketMode3[i].TransmitTimestamp$
      $T2 \leftarrow PacketMode4[i].ReceiveTimestamp$
      $T3 \leftarrow PacketMode4[i].TransmitTimestamp$
    end if
    $i \leftarrow i + 1$
  end for

2.2 NTP State Transition Analysis

Another complementary detection approach is based on modelling observed traffic as a finite state machine. Each type of an NTP exchange (according to the reference ID of every NTP packet) is represented as a state. The algorithm uses the following states that are referred by a letter in bracket: OK (O), STEP (S), RATE (R), INIT (I), DENY (D), OTHER (B).

The observed NTP traffic can be represented as a sequence of characters, i.e. for instance *OOIIISSSSO* stands for a sequence with 2 OK, 3 INIT, 4 STEP and 1 OK.

Determining a state depends on the reference ID and stratum fields of NTP. For packets with stratum 0 (unspecified or invalid), the state can be read directly from the packet and it is called the kiss code. For stratum 1 or higher, the reference ID identifies the server which the time information is taken from.

The second detection method is based on analysing sequences of states. The patterns of suspicious traffic are transitions (sequence of two states) or set of transitions of the states. This approach helps to reduce the number of false positives of both on-line and off-line attacks.

3 Conclusion

The algorithms that were shown in the previous section were implemented as a detection module of the NEMEA system. The module was tested using traffic traces from small local computer network as well as backbone national academic network. The strategy for detecting on-line attacks is based on computing the offset of an NTP packet exchange and compare that value with a threshold value. The threshold is defined by an upper-bound and a lower-bound value that satisfies a confidence interval based on Chebishev's Inequality. Using a training dataset that was created manually from the traffic traces, the threshold and confidence interval were estimated.

In order to perform our tests, a virtual environment was created consisting of three virtual machines, i.e. NTP server, NTP client and an attacker. The virtual machines were used to demonstrate that the NTP attacks can be done quite easily. The implemented detection module was tested to detect the running attacks.

Since the generated attacks were detected by the module, it seems to be a promising security tool. However, from the nature of the detection algorithm, the module is sensitive to missing data. As a result, the detection performance is highly disrupted when it does not see both directions of NTP exchanges or when the monitoring infrastructure uses data sampling. In addition, the module expects ordered flow records for correct functionality.

Acknowledgments

This work was partially supported by the "CESNET E-Infrastructure" (LM2015042) and CTU grant No. SGS16/124/OHK3/1T/18 both funded by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] Bartos, V., Zadnik, M., Cejka, T.: Nemea: Framework for stream-wise analysis of network traffic. Tech. rep., CESNET (2013)

[2] Malhotra, A., Cohen, I.E., Brakke, E., Goldberg, S.: Attacking the network time protocol. NDSS'16 (2016)

[3] Mills, D., Martin, J., Burbank, J., Kasch, W.: Rfc 5905: Network time protocol version 4: Protocol and algorithms specification. Internet Engineering Task Force (2010)

[4] Robledo, A.U.: Network Time Protocol Attacks Detection. Master's thesis, Czech Technical University in Prague, Faculty of Information Technology (2016)