Czech Technical University in Prague

Faculty of Information Technology

ASSIGNMENT OF MASTER’S THESIS

Title:	Network Time Protocol attacks detection
Student:	Mr Alejandro Robledo
Supervisor:	Ing. Tomáš Čejka
Study Programme:	Informatics
Study Branch:	Computer Security
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2016/17

Instructions

Study Network Time Protocol (NTP) that can be used for time synchronization in computer networks.
Study principles of modern network monitoring using network flows.
Study published vulnerabilities in [1] that allow attacks based on manipulation of victim's system time and try to repeat an attack in an experimental/virtual network.
Propose a set of needed information from NTP for detection of the attacks shown in [1].
Develop a detection module for the Nemea system [2] to detect the described attacks.
Evaluate the detection module functionality using an experimental real network infrastructure (in cooperation with the supervisor).

References

[1] Malhotra, Aanchal, et al. "Attacking the Network Time Protocol."
[2] https://github.com/CESNET/Nemea

L.S.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 9, 2015

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Systems

Master's thesis

NETWORK TIME PROTOCOL ATTACKS DETECTION

Bc. Alejandro Robledo Urrea

Supervisor: Ing. Tomáš Čejka

8th May 2016

Acknowledgements

I would first like to thank my thesis supervisor Ing. Tomáš Čejka of the Faculty of Information Technology at Czech Technical University in Prague. He was always open whenever I ran into a trouble spot or had a question about my research or writing. I must express also my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Declaration

Citation of this thesis

Robledo Urrea, Alejandro . *NETWORK TIME PROTOCOL ATTACKS DETECTION* . Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Network Time Protocol (NTP) se v počítačových sítích používá pro synchronizaci času. Nevhodně nastavená NTP infrastruktura umožňuje útočníkovi manipulovat se systémovým časem oběti. Cílem této práce je ověření zranitelnosti používaného protokolu v simulovaném prostředí. Výsledkem práce je experimentální otestování proveditelnosti tohoto typu útoku, návrh funkčního detekčního mechanizmu a nakonec i vlastní implementace detekčního modulu pracujícího se záznamy o tocích rozšířenými o NTP informace. Implementovaný modul byl integrován a testován v rámci existujícího systému NEMEA.

Klíčová slova Síťová bezpečnost, Network Time Protocol, On-line útok, Off-line útok, Kiss of Death,Priming the pump, NEMEA systém, Detekce útoků, Spolehlivostní interval

Abstract

Network Time Protocol (NTP) is used for time synchronization in computer networks. When NTP is insecurely configured, it is possible for attackers to manipulate victim's system time. The aim of this thesis is to verify a vulnerability of the widely-used protocol in a simulation environment. The result of the thesis is a created experimental environment for testing the NTP attacks, design of a working detection mechanism and, finally, an implementation of a detection module that works with flow records that are extended by NTP information. The implemented module was integrated and tested with existing NEMEA system.

Keywords Network Security, Network Time Protocol, On-line attack, Off-line attack, Kiss of Death, Priming the pump, NEMEA system, Detection of attacks, Confidence Interval

Contents

List of Figures

List of Tables

Introduction

Network Time Protocol (NTP) [1], is commonly defined as one of the oldest Internet protocols in current use (in operation since 1985), and is also the most widely used and accepted method for maintaining accurate time across entire data networks (some alternatives are: Precision Time Protocol (PTP) [2], Reference Broadcast Synchronization (RBS) [3], Global Position System (GPS) [4]). Managing time has become an essential ingredient in networking nowadays for different reasons: the most obvious reason is to have configured all the networking devices with accurate time; but, behind this naive reason there are implications directly for Information Technology (IT) application systems which correct and secure functioning relies on the fact that time must be synchronized and accurate on all the elements involved in the system.

If NTP fails on a time dependant system, multiple elements (e.g. applications) on the system could fail also. Some examples (among others), of these kind of systems are: a) Bitcoin is a digital currency system where managing transactions is carried out collectively by the use of peer-to-peer technology to operate with no central authority. One of the most important elements is the block-chain, which is a public ledger[1] of all transactions in the Bitcoin network, and is composed by timestamps "blocks". Each block contains a record of some or all recent transactions, and a reference to the block that came immediately before it. By announcing inaccurate timestamps when connecting to a node, an attacker can alter a node's network time counter and mislead it into accepting an alternate block-chain. This could significantly increase the chances of a successful double-spend, drain a node's computational resources, or simply slow down the transaction confirmation rate [5]; (this is a general problem of all digital currencies). b) HTTP Strict Transport Security (HSTS) RFC-6797, is a web site security mechanism that mitigates Man-In-The-Middle (MITM) attacks by enabling web servers to declare themselves accessible only via Hyper Text Transfer Protocol Secure (HTTPS) connec-

[1]And account book of recorded business transactions

tions [6]; but, in [7] it is described how an attacker who sends forward in time a computer device connected to a web server using HSTS, can force HSTS policies to expire; as a consequence, a web client typing a Hyper Text Transfer Protocol (HTTP) Uniform Resource Locator (URL) in its browser, could get connected in an insecure way, and the attacker could also intercept all the information. c) An attacker could also forward in time a Domain Name Service (DNS) server by a day (typical live-time of a cache entry), causing most of its cache entries expire. As it is stated by [8], a widespread NTP failure could cause multiple resolvers to flush their cache at the same time and flooding the network with DNS queries. d) When working with Transport Layer Security (TLS) certificates, an attacker who sends backward in time the date of a computer system, could cause the host to accept certificates that the attacker acquired fraudulently[2], and have since been revoked; finally, the attacker could make expired certificates to become valid [8]; as a consequence, this allows the attacker to decrypt the data.

NTP is an excellent way to keep a large number of network nodes in close synchronization, requires a minimum of network overhead (one exchange of NTP packets every 64 to 1024 seconds [1]), can also maintain a high level of synchronization accuracy and is easy to implement. On the other hand, NTP is vulnerable to attacks and the most representative is the NTP amplification attack, topic on which researchers focus their attention nowadays. Moreover, some implementations of the protocol present security issues when they are configured without authentication (e.g., Ubuntu ntpd 4.2.5p5). When using some of these NTP implementations, an attacker on-line (attacker with visibility of NTP traffic) and off-line (attacker without visibility of NTP traffic), can change the time of an NTP client or can also perform a Denial-of-Service (DoS) attack by not allowing the client to synchronize to its server, as it was stated by [8]. In order to continue the investigation over the integrity of time information transmitted by NTP protocol, the focus of this thesis will be on on-line and off-line attacks over NTP, and also on methods for their detection.

There exist tools in internet (e.g. Delorean [9], free access), that allows to perform on-line attacks [7]. Some vendors provided patches to mitigate on-line and off-line attacks over NTP (e.g. Cisco Systems, RedHat security team [10]); additionally, Juniper Networks investigated about this topic in order to stablish which of its operative systems and software versions are vulnerable to on-line attacks over NTP [11] (CTPOS/CTPView are vulnerable). There is still a technological gap due to the fact that many other vendors haven't solve this issue; moreover, even counting with some patch, it is required to be applied, and of course, there is no guaranty that all devices with this vulnerability would apply it.

The main objective and contribution of this thesis is to fill that gap in the

[2]This conference by Moxie Marlinspike, exposes a case of stolen certificates: `http://youtu.be/Z7Wl2FW2TcA`

following way: First, it is going to be proved that NTP is vulnerable to on-line and off-line attacks by considering a NTP client-server operation implemented in a simple network using virtual machines (Ubuntu server 14.04.3) in VirtualBox; both client and server will be running the un-authenticated NTP implementation ntpd v4.2.6p5 (which is the second most popular version of ntpd found by [8] after v4.1.1).

Second, it is going to be designed and implemented a strategy for detection of anomaly values of timestamp offsets in the NTP communication, which leads to the detection of on-line attacks. It is also necessary to define a threshold in order to reduce the amount of false positives. As a complement to the previous strategy, the behaviour of the normal NTP traffic is modelled using a state machine where the states are defined by values of the NTP field reference ID. Some transitions between states can be used as evidence of anomaly or suspicious behaviour, and provide information about possible on-line and off-line attacks.

Third, this implementation will be integrated to Network Measurement Analysis (NEMEA) system. NEMEA is a framework for automated analysis of flow records gathered from network monitoring processes in real time [12]. After the integration, the implementation could be used and tested in real environments.

If we want to understand the current state of NTP security, it would be a good idea to take a look to the specification RFC-5905 [1], which describes the implementation of NTPv4. This document bases the security of NTP on the use of authentication. NTPv4 supports both symmetric and asymmetric cryptographic authentication, but, it is rarely used in practice; when using symmetric cryptography it is required manual configuration of a symmetric key in both, the client and server, and this makes symmetric cryptography not very practical when a server must communicate with a big amount of clients. As stated by [8], the use of cryptography combined with additional defence mechanisms already built in the NTP architecture, protocol, and algorithms, makes timestamps exchange scheme resistant to spoofing and packet-loss.

A different approach for securing NTP is documented in [13]. Security is achieved by following two approaches: First, it is recommended the implementation in routers Cisco or Juniper, of BCP38[3], for Network Ingress Filtering to defeat Denial of Service Attacks that employ IP Source Address Spoofing; Second, it is stated a set of recommendations about how to reach secure NTP, by using a better configuration of the ntp.conf file[4] than the default configuration. One example of the recommendations is to configure synchronization with the local operative system, which is particularly useful when there is no communication with the configured NTP servers. However,

[3]Also known as "Network Ingress Filtering", is a security mechanism that filters incoming packets from end customers and allow packets only from IP addresses assigned to them.

[4]configuration file of NTP for Linux operative systems

none of the security mechanisms discussed previously are utilized during the experiments described in this thesis.

As it was already mentioned, the focus of most researchers nowadays, from a security point of view related to NTP protocol, is about NTP amplification attacks. [14] explains this attack in the following way: An NTP amplification attack begins with a computer controlled by an attacker on a network that allows source IP address spoofing. The attacker generates a large number of UDP packets with spoofed source IP address. These UDP packets are sent to NTP servers that support the MONLIST command in order to request a list of the last IP addresses that accessed the NTP server. If an NTP server has its list fully populated, the response to a MONLIST request will be 206-times larger than the request. In this attack, since the source IP address is spoofed and UDP does not require a handshake, the amplified response is sent to the intended target.

This thesis is divided as follows: NTP Fundamentals will be described in Chapter 1, and gives an explanation of how a NTP environment works, considering un-authenticated NTP and using client-server mode of operation. Attacking NTP protocol will be described in Chapter 2 as well as experimental results. In Chapter 3, it is shown the implementation of the detection module of these attacks. The next step is the integration between the detection module of NTP attacks and NEMEA system, which is explained also in Chapter 3. Finally, the whole system (module of detection of NTP attacks + NEMEA system) is evaluated and documented in Chapter 4.

Chapter 1

NTP Fundamentals

The objective of this chapter is to explain the fundamental concepts about NTP protocol, to define some terminology and to describe the exchange of packets between the NTP client and NTP server. It is also focused on concepts and aspects that an attacker must be aware of in order to perform the attacks described in Chapter 2.

1.1 NTP Protocol

NTP is widely used to synchronize system clocks among a set of distributed time servers and clients and it is built on the Internet Protocol (IP) and User Datagram Protocol (UDP) [1]. There are three NTP protocol variants: symmetric, client/server, and broadcast. Client/server mode of operation is the most widely used in Internet, and is the mode on which [8] found vulnerabilities. As a consequence, that's the mode considered in this thesis.

Client/server mode operates in an hierarchical structure where the level of the server defines it's stratum, as it is shown in Figure 1.1 [15]. A client requests time information to a set of servers. The stratum is a number from 0 to 16 where 1 means that the server is at the root of the structure (primary server); a client which synchronizes to a stratum 1 device, will be at stratum 2 and the structure continues this way. Devices at stratum 0 and 16 means that they are unsynchronized. Secondary servers are those at stratum 2 to 15.

1.2 NTP Packet Structure

The attacks described in Chapter 2, and that were investigated by [8] and [7], are based on spoofing NTP packets, so, a full description of the NTP packet is shown in Figure 1.2 and explained as follows[5] :

[5]This is a reduced view of the NTP packet since not all fields are explained and also not all possible values of every field are shown. This explains the fields considered important for

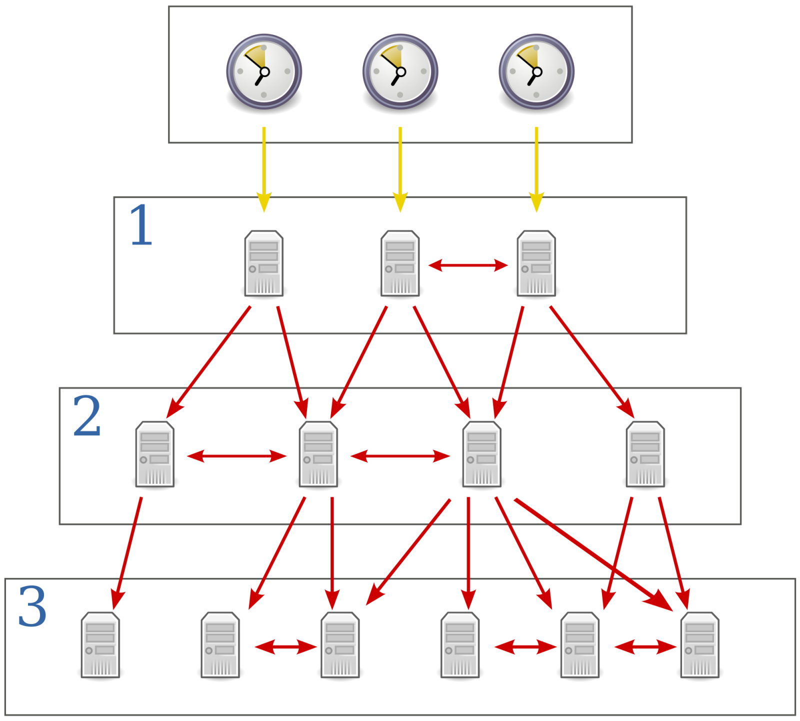
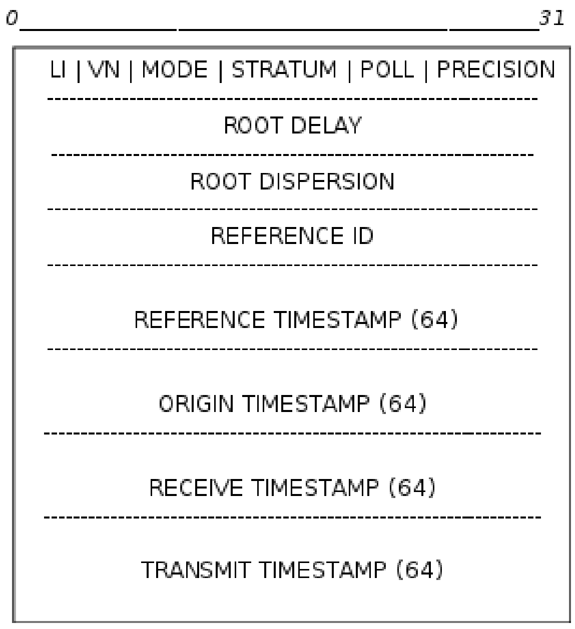


Figure 1.1: NTP Hierarchical Structure



```
0___________________________________________________31
LI | VN | MODE | STRATUM | POLL | PRECISION
---------------------------------------------------------
ROOT DELAY
---------------------------------------------------------
ROOT DISPERSION
---------------------------------------------------------
REFERENCE ID
---------------------------------------------------------

REFERENCE TIMESTAMP (64)
---------------------------------------------------------

ORIGIN TIMESTAMP (64)
---------------------------------------------------------

RECEIVE TIMESTAMP (64)
---------------------------------------------------------

TRANSMIT TIMESTAMP (64)
```

Figure 1.2: NTPv4 Packet Format

Leap Indicator (LI): Warns about one second to be inserted or deleted in the last minute of the current month [1]. The utility for this field during this thesis is by the fact that LI is equal to 3 in case the client's clock is unsynchronized and it is equal to 0, 1, or 2 if synchronized.

Mode: Since the focus is on client/server mode, it is most interesting when Mode is equal to 3, which means that the client sends a request for time information to the server. Additionally, Mode equal to 4 means that the server responds to client with it's time information. This type of packets are explained in detail in Section 1.3.

Stratum: Indicates the level in the hierarchical structure explained in Section 1.1. 1 indicates primary server (e.g., equipped with a GPS receiver); 2-15 are secondary servers synchronized via NTP; 0 invalid and 16 unsynchronized [1].

Poll: Maximum interval between successive messages, in $\log_2$ seconds[1].

Precision: It is the smallest possible increase of time that can be experienced in NTP. NTP precision is determined automatically, and it is measured as a power of two. For example if precision variable is equal to -16, it means that the precision is 2^{-16} seconds [16].

Root Delay, Root Dispersion: Indicates the total round trip delay and total dispersion, respectively to the primary reference source (Stratum 1) [17]. The concept of Delay is described in detail in Section 1.4.

Reference ID: The interpretation depends on the value in the stratum field. For packet stratum 0 (unspecified or invalid), this is a four-character ASCII RFC-1345 string, called the kiss code [1]. Some values that are useful (from an attacker point of view) and discussed in next Chapters are shown in Table 1.1. Moreover, above stratum 1 (secondary servers and clients), represents the

Table 1.1: Examples Reference ID Values

Reference ID	**ASCII Code**	**Observation**
INIT	73.78.73.84	NTP client service initialized
STEP	83.84.69.80	NTP client takes server's time
RATE	82.65.84.69	client must reduce polling interval
DENY	68.69.78.89	stop sending packets to the server

identifier of the server from which the time information is taken. Since we are working with IPv4 RFC-791, the identifier is the four-octet IPv4 address. If using the IPv6 address family, it is the first four octets of the MD5 hash of the IPv6 address [1].

Reference Timestamp: Time when the system clock was last set or corrected [1].

the attacks that are discussed in Chapter 2

Origin Timestamp: Considering a mode 4 response, it is the client's time when the Mode 3 request departed towards the server. As it is explained in Chapter 2, this field is very important since it is used by the client as nonce in order to check if the Mode 4 response from the server is valid.

Receive Timestamp: Considering a mode 4 response, it is the server's time when the Mode 3 request (coming from the client) arrives to the server.

Transmit Timestamp: Local time when client sends Mode 3 request or server sends Mode 4 response packet. Also know as Sent Timestamp.

1.3 NTP Exchange of Packets

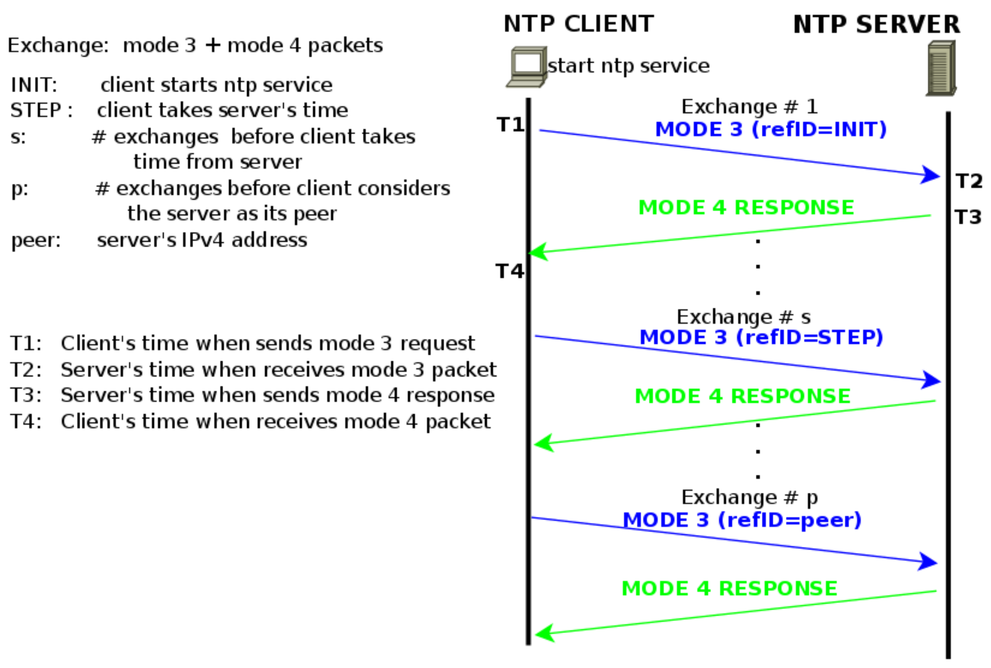
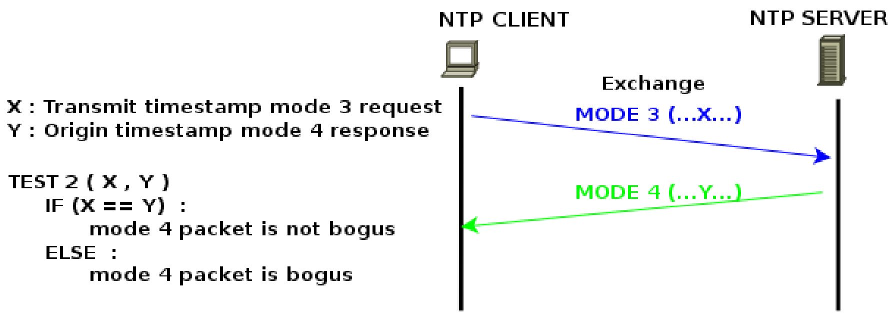

Exchange: mode 3 + mode 4 packets
INIT: client starts ntp service
STEP : client takes server's time
s: # exchanges before client takes time from server
p: # exchanges before client considers the server as its peer
peer: server's IPv4 address
T1: Client's time when sends mode 3 request
T2: Server's time when receives mode 3 packet
T3: Server's time when sends mode 4 response
T4: Client's time when receives mode 4 packet
NTP CLIENT
NTP SERVER
start ntp service
T1
Exchange # 1
MODE 3 (refID=INIT)
T2
MODE 4 RESPONSE
T3
T4
Exchange # s
MODE 3 (refID=STEP)
MODE 4 RESPONSE
Exchange # p
MODE 3 (refID=peer)
MODE 4 RESPONSE

Figure 1.3: NTP Packet Exchange

Figure 1.3 shows the exchange of packets required for an NTP client to select a server as its peer. Normally a client configures more than one server. This example works with just one server for simplicity for explaining the concepts. Lets assume that the NTP server is synchronized (It is taking its time from a selected NTP peer), and NTP service at the client side just starts. The client sends a mode 3 packet to the server; this mode 3 packet represents a request for time information; the main characteristics of this packet are: mode = "3" and referenceID = "INIT". The server receives the mode 3 packet and replies with a mode 4 packet; the main characteristics of a mode 4 packet are: mode = "4", origin timestamp = "Transmit timestamp used by the client in its mode 3 request" and Transmit timestamp = "Server's time when sends mode 4 packet". When the client receives the mode 4 reply from the server, the first packet exchange is finished. The client sets its time to the server's time. Then, the client starts the exchange number s by sending a mode 3 packet with referenceID = "STEP". Additionally, in the packet exchange number p the client notifies the server that it was selected as the client's peer. The possible values for s and p are determined by the poll process and clock discipline algorithm (described in Section 1.5). As it is expressed in Figure 1.4, when the client receives the mode 4 packet response, it checks that the value in the field Origin timestamp corresponds to the value Transmit timestamp in the mode 3 packet request. This is defined by [1] (page 36) as Test 2, and is the mechanism used by the client to know if the response from the server is legitimate.



```
X : Transmit timestamp mode 3 request
Y : Origin timestamp mode 4 response

TEST 2 ( X , Y )
   IF (X == Y)  :
      mode 4 packet is not bogus
   ELSE  :
      mode 4 packet is bogus
```

Figure 1.4: Test 2 by NTP Client

1.4 Peer Process Statistics Variables

Using the definitions of timestamps stated in Figure 1.3, it is possible to define some important variables that the client's NTP daemon uses to select the peer:

Delay δ : Round-trip delay during NTP exchange of packets between client and server [1].

$$\delta = (T4 - T1) - (T3 - T2) \tag{1.1}$$

Offset θ : Offset quantifies the time shift between a client's clock and a server's clock [8]. This is a very important parameter for detection of on-line attacks described in Chapter 2 and implemented in Chapter 3. Lets assume that delays on client to server communication and reverse, server to client, are symmetric and equal to $\frac{\delta}{2}$. Then, the gap between the server and client clock is $T2$–$(T1 + \frac{\delta}{2})$ for the mode 3 request, and $T3$–$(T4 + \frac{\delta}{2})$ for mode 4 response. After averaging this two values we get Equation 1.2.

$$\theta = \frac{1}{2}\Big[(T2 - T1) + (T3 - T4)\Big] \tag{1.2}$$

1.5 How Does NTP Work?

Some of the following concepts may not be necessary at the moment of performing an attack on NTP, but are necessary to understand its behaviour and possible ways to detect the attack. The algorithms and processes described in Table 1.2 , are used by the client using the NTP daemon in the following way: First, it is discovered the available NTP servers and start association for each server found; those servers become candidates for being the peer of the client. Clock Select Algorithm determines from the candidates, the ones that are good timekeepers and bad timekeepers (with offset $\theta \rangle$ 10 seconds [8]). Then, the Clock Cluster Algorithm produces the survivor list by pruning the statistical outliers from the good timekeepers. Finally, the Clock Discipline algorithm uses the survivor statistics in order to discipline the system clock[6] [18].

1.6 Important NTP Thresholds and States

Assume an NTP client is synchronized to an NTP server and they continue with exchange of packets. The client calculates the offset when the mode 4 packet is received from the server using Equation 1.2. Then the Clock Update Routine described in Table 1.2 uses that information, compares with the NTP Thresholds from Table 1.3 , and the possible results are the following [1]:

PANIC: Means the offset is greater than the panic threshold PANICT (1000 s) and should cause the program to exit with a diagnostic message to the system log.

STEP: Means the offset is less than the panic threshold, but greater than the step threshold STEPT (125 ms). In this case, the clock is stepped to the correct offset.

ADJ: Refers to adjustment. Means the offset is less than the step threshold and thus a valid update.

[6]This is an overview of NTP processes and algorithms and not all algorithms are included. In order to have a detail description of them, it is recommended to read [18],[16] and [1]

Table 1.2: NTP Processes and Algorithms

Process / Algorithm	Description
Poll Process	Sends NTP packets at intervals determined by the clock discipline algorithm. The process is designed to provide a sufficient update rate to maximize accuracy while minimizing network overhead. The process is designed to operate over a poll exponent range between 3 (8 seconds) and 17 (36 hours).
Clock Select Algorithm	Determines from a set of servers, which ones are correct (good timekeeper) and which ones are not (bad timekeeper) according to a set of formal correctness assertions.
Clock Cluster Algorithm	Processes the good timekeepers produced by the clock select algorithm to produce a list of survivors by pruning the statistical outliers. These survivors are used by the mitigation algorithms to discipline the system clock.
Clock Discipline Algorithm	It is the heart the NTP [1]. NTP keeps precision time by applying small adjustments to system clock periodically [16].
Clock Update Routine	Each time an update is received from the system peer, the clock update routine is called. It uses different modes (e.g. PANIC, ADJ and STEP) to describe the result [1].

Table 1.3: NTP Thresholds

Variable	Description
PANICT	Panic Threshold (Default value 1000 s)
STEPT	Step Threshold (Default value 125 ms)
step-out	Minimal interval the client will consider a time step as valid since the last step (Default value 900 s)

1.7 NTP Timestamp

Working with timestamps is very important for NTP protocol. As it was discussed in previous sections, there are fields in NTP header that contains time information such as: origin timestamp, receive timestamp and transmit timestamp. Some important parameters related with NTP are calculated based on timestamps information. Also, in order to parse correctly NTP packet headers related to timestamps, it is important to know its structure and the way to work with them. A timestamp is a 64-bit, unsigned fixed-point number in seconds and fraction [18]. So, the timestamp is composed by 32 bits for representation of seconds relative to 0 hours on 1 January 1900 and 32 bits to represent fractions of seconds, as it is shown in Figure 1.5. The multipliers to the right of the point are 1/2, 1/4, 1/8, 1/16, etc [19].

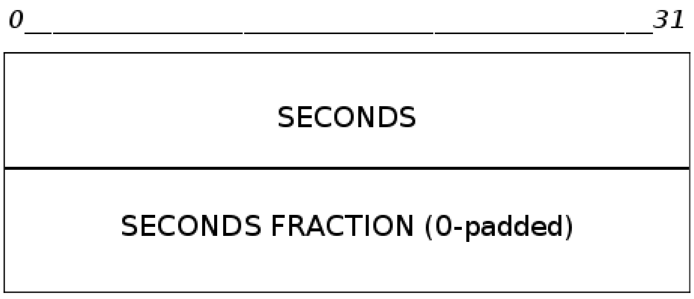


Figure 1.5: Format of NTP Timestamp Field

1.8 NTP Implementation

There exist implementations of NTP protocol for different operative systems like *Linux Systems*, *Microsoft Windows* and *Virtual Memory system.* Lets focus on the implementation of NTP for *Linux Systems* since it is the operative system used during all the experiments explained in this thesis. The software is available as C source and it runs on most UNIX compatible operating systems. This implementation of an NTP client and server is available for free. The software consists of the following components:

ntpd: It is the daemon process that is both, client and server.

ntpdate: A utility to set the time once.

ntpq, ntpdc: These components are used for monitoring and control programs that communicate via UDP with ntpd.

ntptrace: This is an utility to back-trace the current system time, starting from the local server.

The *ntpd* program is an operating system daemon that sets and maintains a computer system's time in synchronization with time servers. It is a complete implementation of the NTP Protocol version 4 (the current version). Additionally, *ntpd* uses a single configuration file to run the daemon in server and client modes. The configuration file, usually named ntp.conf, is located in the /etc directory in Linux systems [20] . Another important characteristic of *ntpd* is its configuration option called -*g*, which allows an NTP client that first initializes to accept any time shift, even one exceeding 1000 seconds [8].

Since the -*g* is the origin of the vulnerability disused in 2.2 it is important to disuse it in detail. Normally, *ntpd* exits with a message to the system log if the offset (difference between server and client time) exceeds a value of 1000 seconds by default. This option allows the time to be set to any value without restriction; however, this can happen only once. If the threshold is exceeded after that, *ntpd* will exit with a message to the system log.

Chapter 2

Attacking NTP Protocol

The objective of this chapter is to describe the attacks over NTP protocol found by [8] and [7]. First it is important to describe the conditions under which the experiments took place. Then, there is a description of On-line and Off-line attacks. At the same time, experimental results are shown and disused.

2.1 Technical Conditions and Assumptions

Figure 2.4 and 2.6 describe the topological diagrams for on-line and off-line attacks respectively. Table 2.1 summarizes the conditions for the experiments. The experiments are performed in a controlled virtual environment using VirtualBox [21] on a Debian [22] host operative system. There are three guests virtual machines using Ubuntu [23] Operative System (one of the most widely used desktop Linux distributions); one is the NTP client, other the NTP server and the last one is the attacker.

It is used the NTP Deamon ntpd 4.2.6p5 as implementation of NTP version 4. The NTP implementation is very important, since [8] and [7] found that ntpd 4.2.6p5 is vulnerable to on-line and off-line attacks[7]. Additionally, this implementation of NTP is the one which is downloaded and installed by default when using Ubuntu operative system[8].

The NTP environment is configured in client/server operation mode. There is one server which is synchronized to its local operative system time in order to avoid dependence with Internet connectivity. Additionally, the client/server communication is un-authenticated. The experiments hold under IP version 4 [24]. The technique for MITM attack (which is one of the ingredients for on-line attacks), is Address Resolution Protocol (ARP) Poisoning. Finally,

[7]Ubuntu and ntpd 4.2.6p5 are not the only systems vulnerable to NTP on-line and off-line attacks. [7] states that these attacks can also be designed against Fedora, Mac OS X Lion, Mac OS X Maverick and Microsoft Windows

[8]apt-get install ntp

Table 2.1: Experimental Conditions for Attacks over NTP

Technical Features of Experiments	**Description**
Virtual Environment	Vitualboxr103443 4.3.32Debian
Host Machine	Debian GNU/Linux 8 (Jessie) 64 bits
Image Guest Virtual Machines	Ubuntu14.04.3server amd64.iso. One is the NTP client, other for NTP server and finally a virtual machine for the attacker
NTP Implementation	ntpd 4.2.6p5. For both, NTP server and client
Scripts for attacks	The programming language for attacks is Python language version 2.7.6 and Scapy version 2.2.0
NTP Configuration	NTP is configured in client/server operation mode un-authenticated. The client is configured with one NTP server. The server is synchronized with its own operative system time.
Internet Protocol Version	IPv4
MITM Technique	ARP Poisoning

the programming language used for the implementation of scripts for attacks is Python [25]. Scapy [26], is a tool for packets manipulation based on Python programming language [9]. This tool is very used for attacking techniques that involve spoofing of packets. Scapy has a little different terminology for the fields of the NTPv4 packet, and it is important to take them into account: *LI* is ***leap*** in Scapy, *Reference ID* is ***id***, *Reference timestamp* is ***ref***, *Origin timestamp* is ***orig***, *Receive timestamp* is ***recv*** and *Transmit timestamp* is ***sent***.

2.2 On-line Attack

[8] describes the way to perform on-line attacks and it is registered in CVE-2015-7704 [10]. The expression on-line attack refers to the fact that the attacker

[9]Python `https://www.python.org/`

[10]Common Vulnerabilities and Exposures (CVE) is a database of publicly known information security vulnerabilities and exposures sponsored by the office of Cybersecurity and Communications at the United States Department of Homeland Security [27]

most have visibility over the traffic between the server and the client.

As it is described in Figure 2.1 and defined by [1], a client will accept a time step from its server if it is greater than the *step threshold* (125 ms by default) and smaller than the *panic threshold* (1,000 s). It is true just if the last clock update is greater than the *step-out* value, which for ntpd is 900 seconds by default.

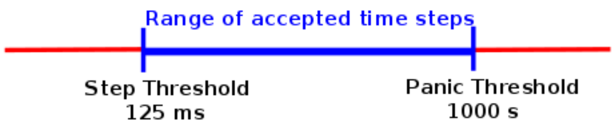


Figure 2.1: Rage for Accepted Time Steps for ntpd v4.2.6

From the previous definition, [7] designed what is called the time skimming attack where an attacker on-line can spoof the mode 4 responses from server to client and generate a time step in the range between the *step threshold* and the *panic threshold*, and this can be performed with a periodicity greater than the *step-out* value. [8] states that this approach is very inefficient since the attack could last at least 114 days in order to produce one year step[11].

In order to quickly shift a client's time, [8] proposes another approach which ingredients are described in Table 2.2: First, some MITM technique is necessary in order to have visibility of the traffic between client and server. ARP Poisoning[12], is the technique used during the experiments in order to perform MITM attack. Second, there exists an option in the ntpd implementation called the *-g* option. The *-g* function works in the following way: The variable *allow_panic* equal to **TRUE** is the responsible for not allowing time steps bigger than the *panic threshold*. But, when the ntpd service is started, *allow_panic* variable is set temporarily to **FALSE**, which allows time steps bigger than the *panic threshold*. The *-g* option of ntpd must be enabled (it is enabled by default) for on-line attacks to be successful. Third, it is necessary a reboot of the ntpd service in order to take advantage of the *-g* configuration. [8] arguments that an attacker can take advantage of some maintenance or use a packet-of-death technique to deliberately reboot the OS, and cause ntpd to restart. Finally, the attacker must spoof mode 4 packets coming from the server to the client by modifying the *Transmit timestamp*.

[11]Shifting the client back one year using steps of size 16 minute each requires $\frac{1x365x24x60}{16} = 32850$ steps in total; with a 5 minute *step-out* value, this attack takes at least 114 days

[12]In ARP Poisoning an attacker sends spoofed ARP messages onto a local area network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host, causing any traffic meant for that IP address to be sent to the attacker instead [28]

Table 2.2: Ingredients for On-line Attack over NTP

Element	Description
Man-in-the-Middle	What is important is that the attacker most have visibility of the exchange of packets between the client and the server. In order to achieve this task the attacker could use whatever technique for hijacking, for instance: ARP poissoning, DNS spoofing, DHCP spoofing
-g option of ntpd	The *-g* is an option of the implementation ntpd that allows time steps greater that panic threshold when there is a reboot of the ntpd service
Reboot	Reboot of ntpd service
Spoof NTP packets	Spoof of NTP mode 4 packets from server to client. The bogus packet determines the time step desired by the attacker

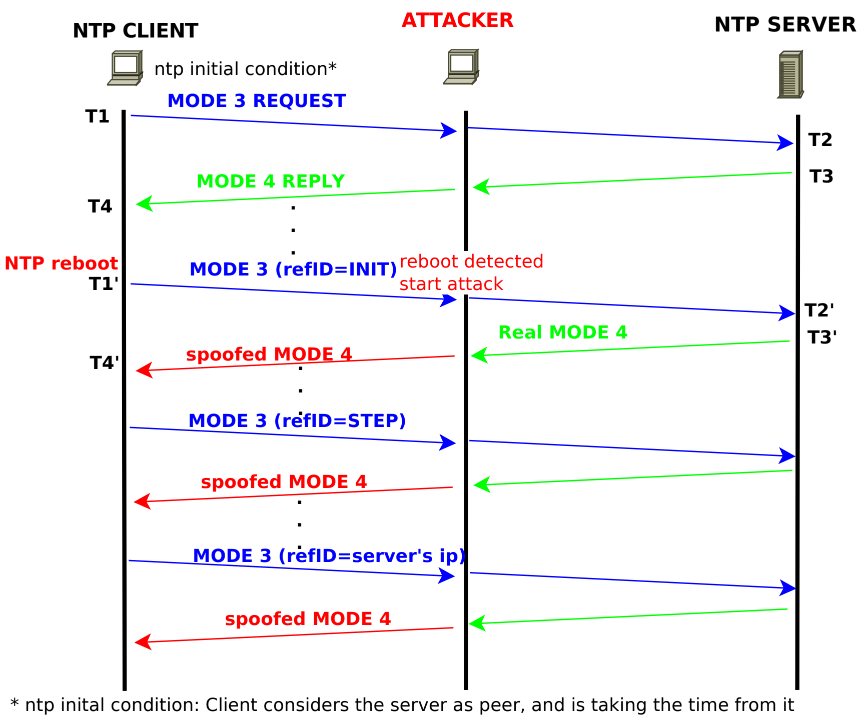


Figure 2.2: NTP Packet Exchange, on-line attack

The sequence of packet exchanges for on-line attack is described by Figure 2.2. Lets assume that the client is synchronized with its server and there is a normal exchange of NTP packets between them. The MITM attack starts and the attacker forwards all packets without any alteration. The ntpd service is restarted, and the attacker detects it when receives the mode 3 packet with *reference ID* equal to **INIT**. Then, the attacker starts to spoof every mode 4 packet coming from the server by manipulating the *Transmit timestamp* value (bogus time) that allows the time step. The exchange of packets continues until the client accepts the bogus time and sends a mode 3 packet with *reference ID* equal to **STEP**. After some packet exchanges the client will consider the server as its peer, the mode 3 packets in the *reference ID* will be equal to the server's IPv4 address and the attack can be considered successful.

Lets focus now on how to spoof mode 4 packets. As it is described in Figure 2.3, the attacker modifies the *Transmit timestamp* by a value (a specific date forward or backward in time) we can call Time attack (Ta). In order to set Ta it is necessary to add or subtract some Attack offset (Ao) to the *Transmit timestamp* of the real mode 4 packet.

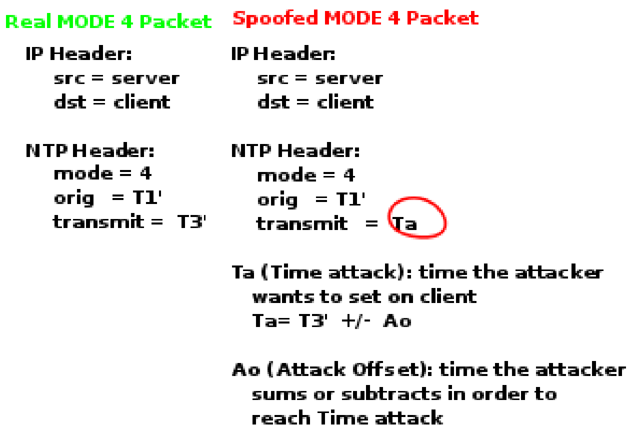


Figure 2.3: Spoofed Mode 4 response, on-line attack

The experiments were executed under a controlled environment described by Figure 2.4. The Figure 2.5 describes the script which was used to execute the on-line attack during the experiments. The experiments confirm that it is possible to perform an on-line attack by sending the client's time forward and backward in time using time greater than the *panic threshold.*

A detail experiment of on-line attack is described as follows and some packets are shown in Figure 2.5, the client is synchronized to its server with current date and time Monday 29 February 2016 15:22:36(3665748156.0). The client ntp service is restarted. The attacker detects the reboot (mode 3 packet *reference ID* is equal to **INIT**) and starts to spoof mode 4 packets using Ao = -215000000.0 seconds which means the Ta is Friday, 08 May 2009 05:09:16 (3450748156.0). After 5 packet exchanges the client **STEP** into the

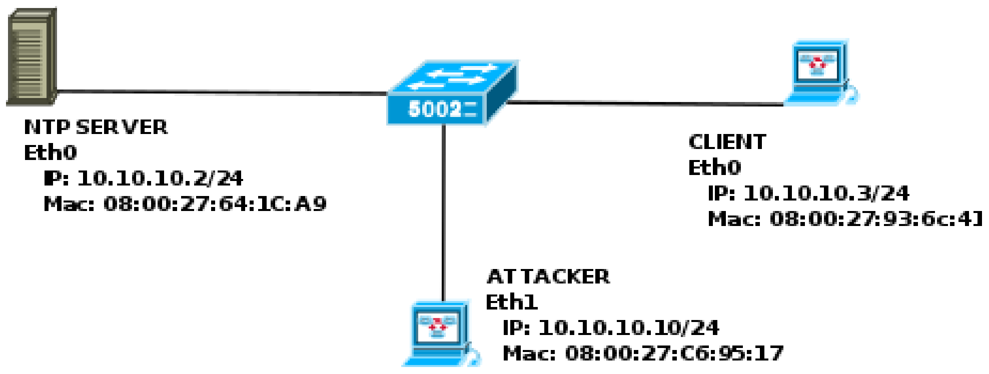


Figure 2.4: Topological Diagram for On-line Attack

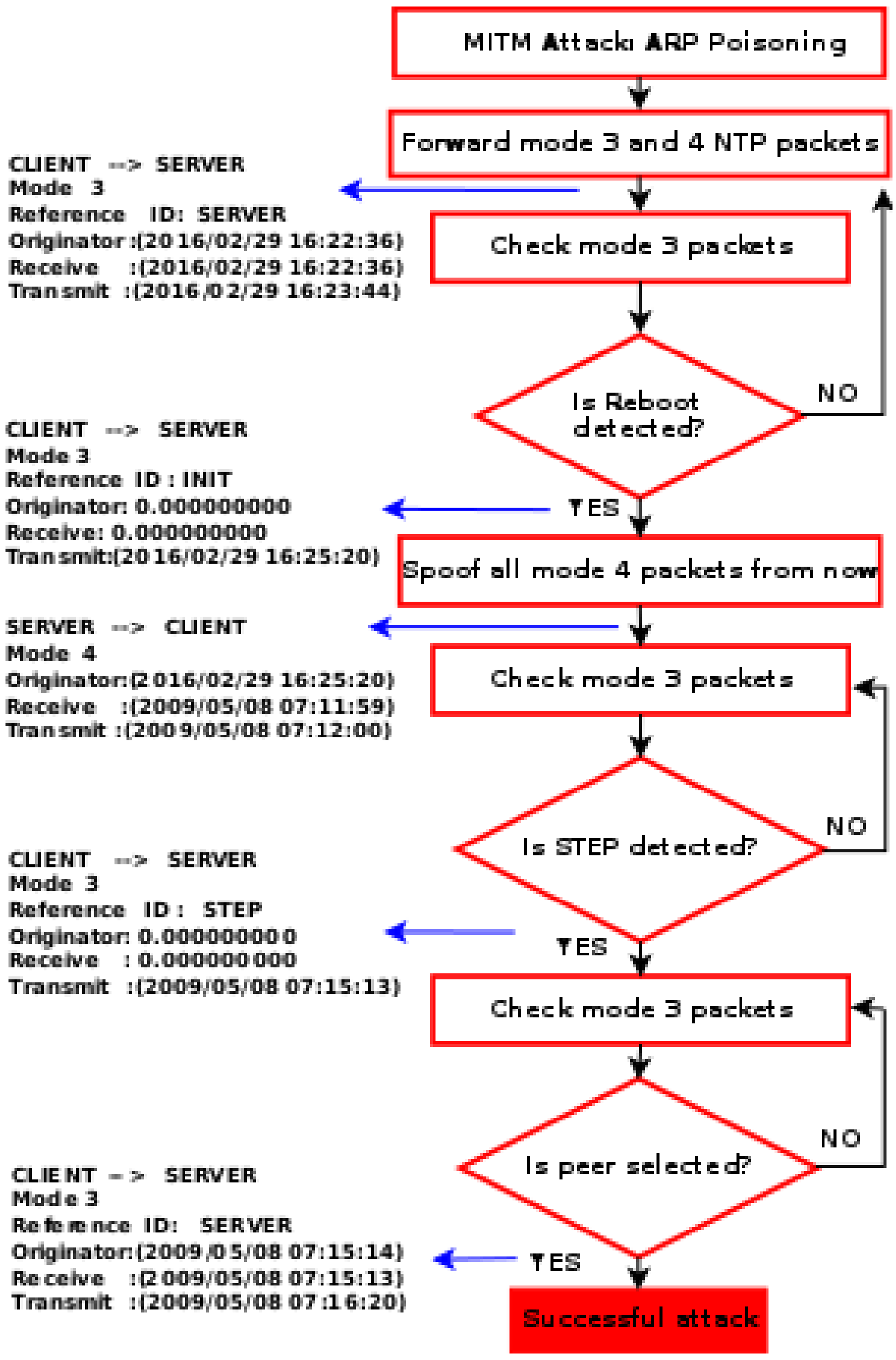
bogus time. At packet exchange number 8 the client considers the server as its peer and the attack is considered successful.

The scripts listed in Table 2.3 were implemented in Python Language[13], and are utilized to perform on-line attacks. *arpPoisoning.py* is used first to guaranty the attacker visibility of the NTP traffic between client and server. Then, the script *attackOnLine.py* is in charge of sending the client forward or backward in time.

Table 2.3: Software Implemented for On-line Attacks

Script	**Description**
arpPoisoning.py	Script used to perform MITM attack using the strategy of ARP poisoning. Implemented in Python language
attackOnLine.py	Script used to perform the on-line attack

[13]Python language `https://www.python.org/`



```
CLIENT  --> SERVER
Mode  3
Reference  ID: SERVER
Originator :(2016/02/29 16:22:36)
Receive    :(2016/02/29 16:22:36)
Transmit   :(2016/02/29 16:23:44)
```

```
CLIENT  --> SERVER
Mode 3
Reference ID : INIT
Originator: 0.000000000
Receive: 0.000000000
Transmit:(2016/02/29 16:25:20)
```

```
SERVER  --> CLIENT
Mode  4
Originator:(2016/02/29 16:25:20)
Receive    :(2009/05/08 07:11:59)
Transmit :(2009/05/08 07:12:00)
```

```
CLIENT  --> SERVER
Mode  3
Reference  ID :  STEP
Originator: 0.000000000
Receive    : 0.000000000
Transmit   :(2009/05/08 07:15:13)
```

```
CLIENT  --> SERVER
Mode 3
Reference ID:  SERVER
Originator:(2009/05/08 07:15:14)
Receive    :(2009/05/08 07:15:13)
Transmit   :(2009/05/08 07:16:20)
```

Figure 2.5: Flow Diagram for On-line Attack Script

2.3 Off-line Attack, Kiss-of-Dead

An off-line attack means that the attacker doesn't have visibility over the exchange of packets between the client and server. It was found by [8] that it is possible to perform a DoS attack on a NTP client by an off-line attacker who sends spoof NTP packets. It is a DoS attack because the client (the victim) will be banned by the server to send mode 3 packets. This behaviour is documented in CVE-2015-7704.

There are two approaches in this case: spoofing a *Kiss-of-Death (KoD)* packet and the technique called by [8] as *priming-the-pump*. As described in Figure 2.6, there is an off-line attacker who has connectivity with both, the server and the client, which means he must be able to send packets to the client and/or server. During the experiments it was used the topology described in Figure 2.4 for simplicity.

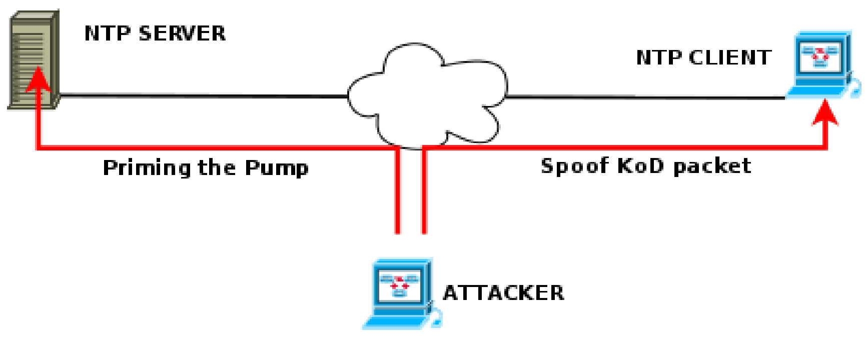


Figure 2.6: Topological Diagram for Off-Line Attacks

An off-line attacker with connectivity to the client can spoof one KoD packet and send it to the client. This situation is shown in Figure 2.7. The KoD packet has the following characteristics: *mode* 4, *LI* 3, some value for *polling interval* $\tau\epsilon\{1, 2, ..., 17\}$, *stratum* 0 and *reference ID* equal to *RATE* or *DENY*. *RATE* is the kiss code and is an order to the client to reduce immediately its polling interval [1]. As a consequence, the client will stop sending mode 3 queries to the server for at least 2^τ seconds (if $\tau = 17$, 2^τ will be around 36 hours). The previous value is just the lower bound, in this case the attacker is not able to control exact values since it is made by poll process algorithm. According to [1], the kiss code *DENY* can completely disconnect the client from its server. According to [8], when using ntpd v4.2.6, an off-line attacker can trivially send the client a KoD that is spoofed to look like it came from its server; the only information the attacker needs is the IP addresses of the relevant client and server; in other words, the *TEST 2* explained in Section 1.3, doesn't apply to KoD packets. During the experiments using ntpd 4.2.6p5 on Ubuntu14.04.3, and under the conditions described in Table

2.1, off-line KoD was not successful since ntpd 4.2.6p5 utilizes Test 2 for KoD packets.

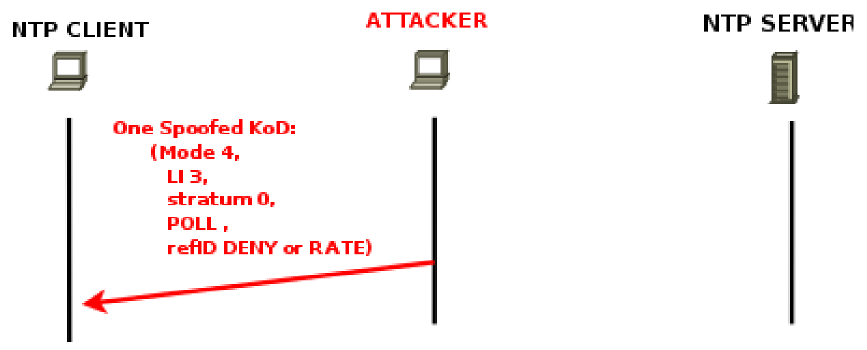


Figure 2.7: Off-line DoS Attack by Spoofing KoD

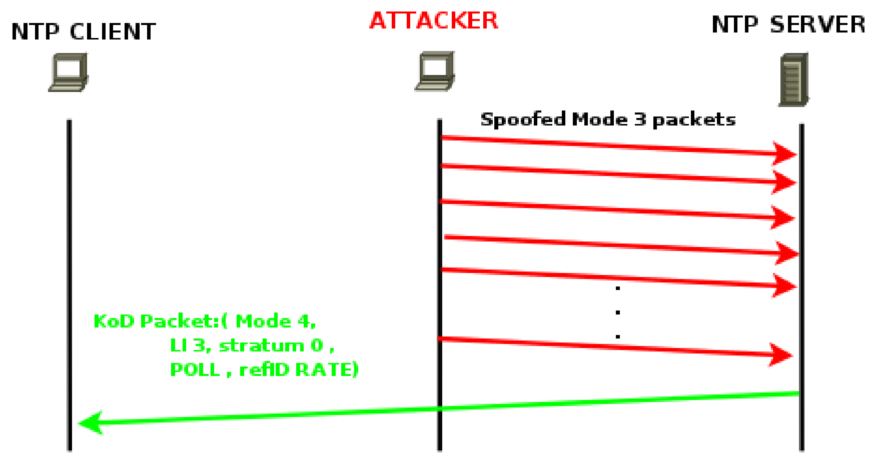


Figure 2.8: Off-line DoS Attack by Priming the Pump

In *priming-the-pump* approach, and attacker spoofs multiple mode 3 requests and sends them to the server. [1] defines that a NTP server will send a KoD packet to a client that queries the server many times during within a specified time interval. In order to test this vulnerability, the exact number of spoofed mode 3 packets that are sent to the server and their time intervals are not relevant. In order to perform the experiment a couple of scapy commands were used. It is sent to the server 90 mode 3 queries in rapid succession (every 1 second), each of which was spoofed with the source IP of the victim client and origin timestamp equal to the current time on the attacker's machine.

```
packet = IP(src="clientIP",dst="serverIP")/UDP()/NTP(mode=3)
sendp(packet, iface="eth1", inter=1 , count=90 )
```

As it is described in Figure 2.8, the server will send the client a mode 4 response which is called Kiss-of-Death. During the experiment, the client stopped sending mode 3 queries during 30 hours.

2.4 Discussion about On-line and Off-line Attacks

The condition for off-line attacks based in KoD is based on network connectivity. Once the attacker has connectivity with the server or the client, the attack is reduced to spoofing NTP packets. Talking about on-line attacks, the conditions are more difficult to achieve since the main ingredient is the reboot of NTP client. The attacker could wait until the client reboots due to software updates or power cycling. Additionally, the attacker could use some packet-of-death technique to force reboot at operative system level, for instance using Teardrop attack[14].

About the possibility of performing on-line attacks over NTP, [30] states that a computer device with a clock that was set to a date months or years in the past could probably trigger errors from the operating system or other applications. On the other hand, it is a real vulnerability and potentially harmful.

At first glance, off-line attacks by spoofing KoD packets or by priming the pump seems not very harmful, but [8] explains an scenario that describes the consequences of these attacks: An off-line attacker could turn off NTP at that client's side by preventing it from synchronizing to any of its preconfigured servers. Some implications could be :

1. The client will rely on its own local clock for the time. If the client has accurate local time, then this attack is unlikely to cause much damage.

2. The client machine could be incapable of keeping time for itself, e.g., because it is in a virtual machine [31]. In this case, client's clock wont be corrected by NTP.

[14]Teardrop is a program that sends IP fragments to a machine connected to the Internet or a network. The bug causes the TCP/IP fragmentation re-assembly code to improperly handle overlapping IP fragments. As a consequence, when a Teardrop attack is run against a vulnerable machine, it will crash or reboot [29].

Chapter 3

Detection Module of NTP Attacks

This chapter explains NEMEA system and the way it is used to achieve the task of detection of NTP attacks described in Chapter 2. It starts with an overview of NEMEA system functionality. There is a description of the way the plug-in for parsing NTP packets was developed. Then, it is explained how offset θ can be used as a parameter for detecting anomaly behaviour and the procedure to establish a correct threshold in order to reduce the amount of false positives. As a complement to the previous strategy, the behaviour of the normal NTP traffic is modelled using a state machine where the states are defined by values of the NTP field reference ID. Some transitions between states can be used as evidence of anomaly or suspicious behaviour and give information about possible on-line and off-line attacks.

3.1 NEMEA System

Figure 3.1 shows the monitoring infrastructure an organization can follow by using NEMEA system for detection of NTP attacks. First, the NTP traffic between the server and the client is captured by the monitoring probe[15]. Second, the collector receives the network traffic from multiple probes. Third, the traffic is received by NEMEA system, which performs data analysis that generates alerts in case an NTP attack is detected. A security analyst who reads the received alert, can handle a security incident according to security policies of the specific organization.

The NEMEA system is a modular system for network traffic analysis and anomaly detection[16]. It is composed by independent modules developed using

[15]An example of probe used by other projects which involves NEMEA system, are COMBO cards `http://www.liberouter.org/technologies/cards/`

[16]`https://github.com/CESNET/Nemea`

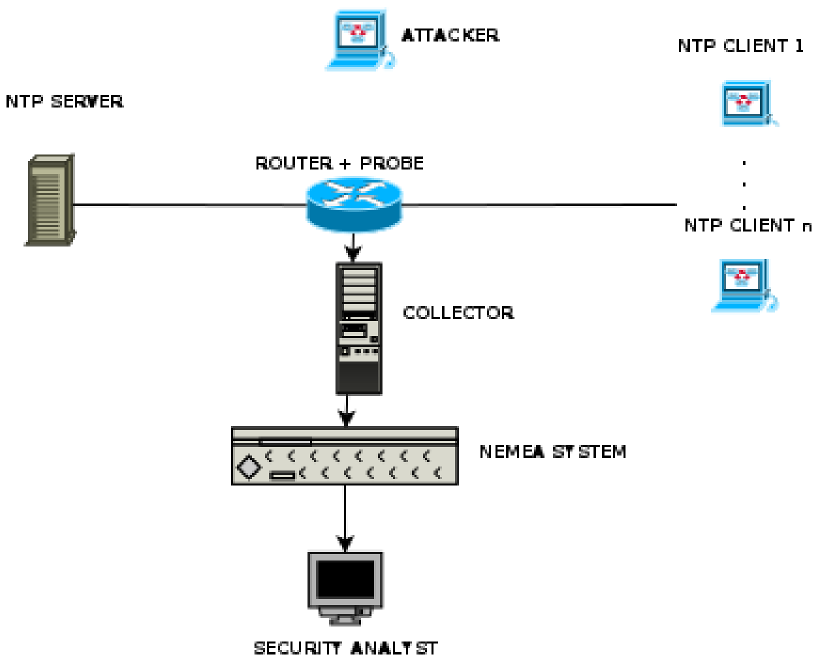


Figure 3.1: Monitoring Infrastructure of NTP Traffic

the NEMEA framework. The framework makes the development of NEMEA modules by implementing common tasks in form of shared libraries. Figure 3.2 shows an example of interconnected modules of NEMEA system.

The way flow records are handled is based on IP Flow Information Export (IPFIX)[17], however, NEMEA uses its own binary data format called Unified Records (UniRec) [12]. Contrary to IPFIX, UniRec was designed for efficient access to all data fields and therefore it can be used for real-time continuous processing of flow records.

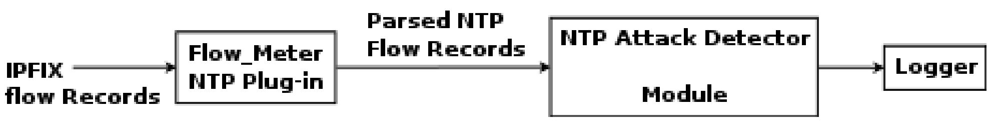


Figure 3.2: NEMEA System for NTP attacks Detection

[17]In paper [32] there is a complete explanation of how flow records work, and it is focused on IPFIX

Table 3.1: NTP Plug-in Fields

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

Table 3.2: Software Implemented for NTP $Plug - in$

Program	**Description**
ntpplugin.cpp	Implementation of the NTP plug-in in C++ language
ntpplugin.h	Implementation of the NTP plug-in in C++ language
flowifc.h, flow_meter.cpp	These files were modify in order to work with ntpplugin.cpp and ntpplugin.h

3.2 NTP Plugin

Since there was no existing monitoring element that could export NTP information needed for this thesis, a `flow_meter` module was extended. It is a simple flow exporter implemented as part of the NEMEA system. The `flow_meter` module can be extended by an implementation of a plug-in that is capable of exporting application specific information. For the purposes of this thesis, NTP plug-in for `flow_meter` was developed.

The plugin was implemented in the C++ Language[18] and the list of created or extended files can be seen in Table 3.2. Using the plugin, NTP related fields were added into flow records. Table 3.1 shows the added NTP fields used during the parsing process. The resulting records are expected and processed by the proposed `NTP_Attack_Detector` module.

[18] C++ Programming Language `http://www.cplusplus.com`

3.3 Detection On-line Attacks Based on Offset

$$\theta = \frac{1}{2}\Big[(T2 - T1) + (T3 - T4)\Big] \tag{3.1}$$

The detection of on-line attacks is based on offset computation. The objective of this approach is to compute the offset θ of every packet exchange using Equation 3.1, as a parameter to model the behaviour of the NTP traffic. Using this technique, it is possible to build a traffic profile where the symptom-specific feature vector is the offset of every NTP packet exchange. The sign of the offset helps to identify if one step in time is forwards (positive) or backwards (negative). Once this parameter is calculated, it is compared with a threshold in order to determine if there is an attack in process. The information about T4 (client's time when receives mode 4 packet), is not present in the mode 3 request neither is in the mode 4 response. In order to have this information, it is necessary to wait until the next mode 3 request and take this value from the field Received timestamp.

Figure 3.3 shows the results of the calculation of offsets for 100 NTP packet exchanges under normal conditions (there is no attack). It is very useful in order to determine and define normal behaviour of the NTP traffic, and to define some threshold during the detection of the attack.

The process of how to calculate the offset for one NTP exchange based on Equation 3.1 requires some additional explanation and also requires to review Figure 1.3. In Figure 1.3 it is defined T1 as the client's time at the moment it sends the mode 3 request to the server and it is in the NTP field Transmit timestamp. Additionally, T2 is server's time at the moment it receives the mode 3 request and this information can be taken from the mode 4 response in the field Receive timestamp. T3 is server's time at the moment it sends the mode 4 response and this information can also be taken from the mode 4 response in the field Transmit timestamp. Finally, T4 is the client's time when it receives the mode 4 response, but, it is not present in the mode 4 response. In order to know the information about T4 it is necessary to wait until the next mode 3 request and take it from the field Receive timestamp. As a consequence, to have all the information required to calculate the offset of one NTP exchange E_i it is necessary to wait until the beginning of the next NTP exchange E_{i+1}.

Data from Figure 3.3 and 3.4 was measured in virtual environment during real experiments. The attacker will send the client's time one hour forward in time. The X-axis represents the timestamps since the beginning of the experiment (one timestamp represents one NTP packet exchange). In red it is represented the calculation of the offset and its value in the Y-axis at the left side (in logarithmic scale). Additionally, in blue it is the client's time and its value is at the Y-axis at the right side. At the beginning of the experiment the client's time is 04:11:23 (using format Hours:Minutes:Seconds). From timestamps 1 to 11 it is shown a normal behaviour of NTP where the

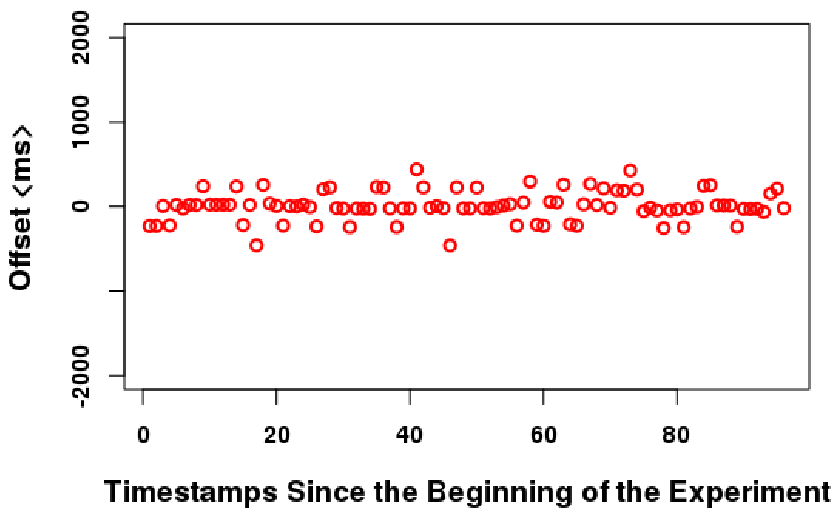


Figure 3.3: Offsets NTP Exchanges (Normal Traffic)

offset is at the order of milliseconds and client's time is increasing normally. The attack starts at timestamp 12 where the server is proposing the client to shift its time one hour to the future. At this timestamp number 12, it is possible to see the evidence of the attack since the offset is about 1,800 seconds, but, it is not evident at the client's time (its value is normal). At timestamp number 17, it is evident that the client accepted the time and it is now 05:28:33 and the attack is considered to be successful. The attack remains active (MITM is active and also the spoofing of mode 4 responses) during the time determined by the attacker. Once the attack is successful, offset is not useful for attack detection any more since it behaves in a normal way again.

The Algorithm 1 describes the way the offset θ for every NTP packet exchanged is calculated. It is assumed there is a client/server NTP communication. Every exchange of packets is represented by E which is composed by two types of packets, one mode 3 request and one mode 4 response. We have n number of packet exchanges E, and for each of them the offset θ is calculated using Equation 3.1. To calculate offset θ[i] of the NTP exchange E_i, it is necessary to wait until the beginning of the next NTP exchange E_{i+1}

The detection module needs to be designed in a way that allows to detect NTP attacks not just for one NTP conversation, but for many. One NTP conversation is defined as the exchange of NTP packets between one client and one server. In order to store information about past exchanges, ten hash-maps were implemented[19]. In these hash-maps, one key is mapped to one value.

[19]The data structured used to build the hash-map in Python is the *Dictionary*. The

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
```

Table 3.3: Pros and Cons of Using Attack Detection Based on Offsets

Pros	**Cons**
Offset of one NTP packet exchange is simple to compute.	In order to have all the information to compute the offset of NTP packet exchange E_i, it is necessary to wait until packet exchange E_{i+1}.
Detection of an anomaly based on offset is simple to implement by comparison with a threshold.	Wrong selection of the threshold can produce false positives or undetected attacks.
If offset is greater that 1,000 seconds. It is an anomaly. This value of offset is not allowed by th the specification [1].	When values of offsets are less than 1,000 seconds, this technique of detection is prone to false positives and or undetected attacks.
The amount of false positives can be reduced by determining experimentally the value of the threshold.	The process of determining experimentally the threshold doesn't have a fixed time.
It is a simple implementation that considers each client/server NTP communication. It is designed to support detection of NTP attacks for multiple NTP conversations (multiple client/server communications) at the same time.	

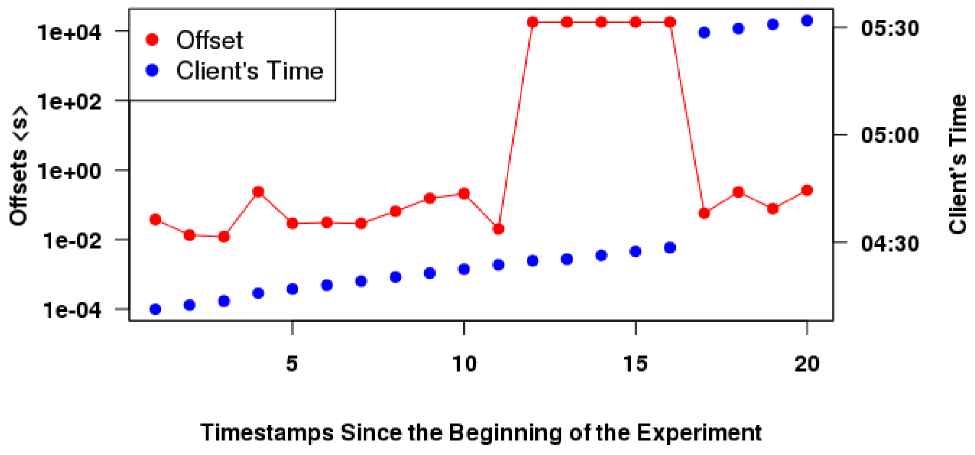


Figure 3.4: Offsets During On-line Attack (1 Hour Forwarding)

Two examples of how the dictionaries are used: The hashmap *hashmap_mode3* stores the information of transmit timestamp for mode 3 packets, and the key is the identification of the conversation that follows the format *clientIP-serverIP*. For instance *10.10.10.3-192.168.1.1* where 10.10.10.3 is the client and 192.168.1.1 is the server. In the same way, there is another hash-map *hashmap_mode4* that stores the information of transmit timestamp and receive timestamp for mode 4 packets, and also the key is the identification of the conversation that follows the format *clientIP-serverIP*. This technique allows the module to handle the detection of NTP attacks for multiple NTP conversations at the same time.

dictionary is defined as an unordered set of key:value pairs, with the requirement that the keys are unique (within one dictionary). `https://docs.python.org/2/tutorial/datastructures.html`

3.4 Detection On-line Attack Threshold Estimation

As it was stated in the previous section, one important step in the detection process is to compare the offset to a threshold that must be selected wisely to reduce the number of false positives. The proposed strategy for calculation of the threshold involves experimental analysis of normal NTP traffic. A template of the *mean* and *standard deviation* of offsets is then used to detect anomalous behaviour, as it is recommended by [33]. In this approach, online learning is used to build a traffic profile for a given network. When newly acquired data fails to fit within some confidence interval of the developed profiles then an anomaly is declared. Normal behavior of time series data is captured as templates and tolerance limits are set based on different levels of standard deviation. The metric is a random variable θ (offset of NTP Exchange) representing a quantitative measure accumulated over a period. This statistical model makes no assumptions about the underlying distribution of θ; all knowledge about θ is obtained from observations [34]. This model is based on the assumption that all we know about θ_1,...,θ_n, are mean and standard deviation using the equations from Algorithm 2.

Algorithm 2 Offset_Threshold (**in**: i, θ_i, sum_{i-1}, ss_{i-1}; **out**: Threshold)

```
Require: i                        # Identifier current NTP exchange
  i-1                             # Identifier previous NTP exchange
  $\theta_i$                             # Offset current exchange
  $sum_{i-1}$                        # Sum of offsets until previous exchange
  $sum_i$                           # Sum of offsets until current exchange
  $ss_{i-1}$                         # Sum of offset squares until previous exchange
  $ss_i$                            # Sum of offset squares until current exchange
  sq                              # Square of offset
  sd                              # Standard Deviation
  d                               # Multiplier standard deviation

  if $i > 1$ then
    $sq_i \leftarrow \theta_i^2$                    # Square of offset
    $sum_i \leftarrow \theta_i + sum_{i-1}$          # Sum all offsets
    $ss_i \leftarrow sq_i + ss_{i-1}$              # Sum all offset squares
    $mean \leftarrow \frac{sum_i}{i}$                # Mean all offsets
    $sd \leftarrow \sqrt{\frac{ss_i}{i-1} - mean^2}$   # Standard Deviation Calculation
    $Threshold \leftarrow d$ x sd
  end if
```

The statistical tool to determine the *Confidence Interval* is the*Chebishev's Inequality*, which is very useful in this case since the random variable (offset) doesn't follow any specific probability distribution.

Every time there is a new NTP packet exchange E_i, its value of offset θ_i is compared with a value of *Threshold* in order to determine if the offset belongs to the *confidence interval* using Algorithm 3. The confidence interval is defined by *mean* ± *Threshold*, where *Threshold = d x Standar Deviation.* If the offset is outside the *confidence interval* an anomaly is detected. By *Chebyshev's inequality* [35], the probability of a value falling outside this interval is at most $1/d^2$; for d = 4, for example, it is at most 0.0625. For this problem, *Chevishev's inequality* is useful since it guarantees that for any probability distribution (nearly all) values are close to the *mean* and no more than $1/d^2$ of the distribution's values can be more than *d standard deviations* away from the *mean.* Additional advantages of using this technique are described in Table 4.2.

Algorithm 3 Confidence_Interval_Threshold (**in**: θ_i, Threshold, mean)

```
if  (mean - Threshold) ⟨ $\theta_i$ ⟩ (mean + Threshold) then
  return TRUE          # No Anomaly
else
  return FALSE         # Anomaly
end if
```

3.5 Detection On-line and Off-line Attack by Modelling Behaviour using States of NTP Exchanges

Another technique suggested in [33], is to model the traffic using a finite state machine. For the case of NTP traffic every NTP packet exchange will receive a state. The NTP traffic is modelled by strings of codes (one letter) that represents the states. The patters of anomaly traffic are transitions (sequence of two letters) or set of transitions of the states. The use of this technique works as a complement to the detection based on offsets, and could help to reduce the amount of false positives. Additionally, it is useful for detection of both on-line and off-line attacks. Another advantage about this approach, is that during a forensic analysis of traffic, it is easier for a security analyst to understand the behaviour of the NTP traffic. It is due to the fact that the analysis can be done by reading sequences or letters. The definitions of the states are based mainly on the analysis of NTP field reference ID of every packet. The interpretation of the reference ID depends on the value in the stratum field. For packets with stratum 0 (unspecified or invalid), this is a four-character ASCII RFC-1345 string, called the kiss code [1]. Moreover,

Table 3.4: States for Modelling NTP traffic

Name of State	Code and Description
OK	**O** - Normal NTP traffic Exchange Figure 1.4. Stratum 1-15.
STEP	**S** - A step in system time has occurred. The value of NTP's field Stratum is 0.
RATE	**R** - The server has temporarily denied access because the client exceeded the rate threshold. Useful state to detect off-line attacks. The value of NTP's field Stratum is 0.
INIT	**I** - The association has not yet synchronized for the first time. Implies reboot. Useful to detect on-line attacks. The value of NTP's field Stratum is 0.
DENY	**D** - Access denied by remote server. Useful state to detect off-line attacks. The value of NTP's field Stratum is 0.
OTHER	**B** - Other type of kiss code. The value of NTP's field Stratum is 0.

Table 3.5: Definition of Anomaly States

Anomaly	Description
n number of Is = one S	Suggests on-line attack by the detection of a reboot at the client's side and then one step.
D	Implies off-line attack by kiss of death spoofing or primping the pump.
R	Implies off-line attack by kiss of death spoofing or primping the pump.

above stratum 1 (secondary servers and clients), represents the identifier of the server from which the time information is taken. Table 3.4 shows the defined states using definitions from [1].

Table 3.5 lists the definition of anomalies to detect off-line attacks and possible on-line attacks.

For instance, the attack described by Figure 3.4, can be modelled as a sequence of states in the following way: *OOOOOOOOOOOIIIIISSSSOO*. The first 11 *Os* are normal exchanges. Then, the ntpd service at the client side reboots and the next 5 exchanges represents *INIT-I* state (the association has not yet synchronized for the first time). After that, there is one exchange in *STEP-S* state when the client accepts the time from the attacker and exchanges continues normally.

Algorithm 4 NTP_States (**in**: E[1,...,n],**out**: S[1,...,n])

```
Require: $NTPExchangesE_i = \{PacketMode3[i], PacketMode4[i]\}, i\epsilon\{1,...,n\}$
  int flag = 0
  for all $E_i$ i:=[1,...,n] do
    Switch ($PacketMode3[i].ReferenceID$)
    Case "$INIT$":
      $S[i] \leftarrow$ "$I$"          # State INIT
      $flag \leftarrow 0$              # Don't Analyse mode 4 packet
    Case "$STEP$":
      $S[i] \leftarrow$ "$S$"          # State STEP
      $flag \leftarrow 0$
    Default:
      $flag \leftarrow 1$              # Analyze mode 4 packet
    end Switch

    if flag == 1 then
      if PacketMode4[i].Stratum == 0 then
        Switch ($PacketMode4[i].ReferenceID$)
        Case "$DENY$":
          $S[i] \leftarrow$ "$D$"      # State DENY
        Case "$RATE$":
          $S[i] \leftarrow$ "$R$"      # State RATE
        Default:
          $S[i] \leftarrow$ "$B$"      # State OTHER
      else
        $S[i] \leftarrow$ "$O$"        # State OK
      end if
    end if
    end Switch
  end for
```

In Algorithm 4, it is explained the way how the states for every packet exchange are defined. The state of one NTP exchange is described as follows: First, it is analysed the mode 3 packet, and it is checked if its *reference ID* (stratum = 0) is equal to *INIT* or *STEP*; in this case the state is **I** or **S** respectively and the mode 4 packet is not analyzed. Otherwise, the mode 4 packet is analysed and it is checked if its *reference ID*(stratum = 0) is equal to *RATE* or *DENY*; in this case the state is **R** or **D** respectively. If stratum is different than 0 the state is **O**. There is one additional state (**B**), and is the case of an mode 4 packet with stratum 0, but *reference ID* is different to *RATE* or *DENY*. The same is done for every packet exchange in order to have the set of states that model the behaviour of the conversation.

The idea of states was extended in order to have more information about

Table 3.6: Pros and Cons of Using States of NTP Exchanges

Pros	**Cons**
Simple way to define states of NTP packet exchanges.	This strategy does not consider the time between NTP packet exchanges.
Works as a complement to the detection of attacks based on anomalies in offsets.	Requires the user to specify the NTP conversation over which the analysis will be performed.
Allows to detect typical packets involved in on-line (Mode 3: reference ID equal to INIT or STEP) and off-line attacks (Mode 4: reference ID equal to RATE or DENY) in real time.	Most be executed together with the detection strategy based on anomalies of offsets.
Helps to identify a pattern for on-line attacks: *OOOOOOOOOOOIIIIISSSSOO*.	Requires off-line learning of malicious patterns before the scheme can be deployed on the network.
Useful to identify new anomalies in the behaviour of NTP packet exchanges.	
The use of characters to define states makes it easy to read by a security analyst during network forensic analysis process.	

one NTP packet exchange, and also this analysis could be useful for future investigations about the behaviour of NTP. After the extension, one NTP will be described by 4 characters $X_1X_2X_3X_4$. X_1 represents the state of reference Id field, as it was explained previously in Table 3.4 . X_2 represents time between 2 consecutive packet exchanges. With X_3 it is possible to know if the client is backward or forward in time with reference to the server. X_4 gives information about how big is the offset. The different values of states are illustrated in Figure 3.5.

In order to understand the utility of this approach, in Table 3.7 it is explained a summary (not all of packet exchanges are shown) of the states during the attack described by Figure 3.4.

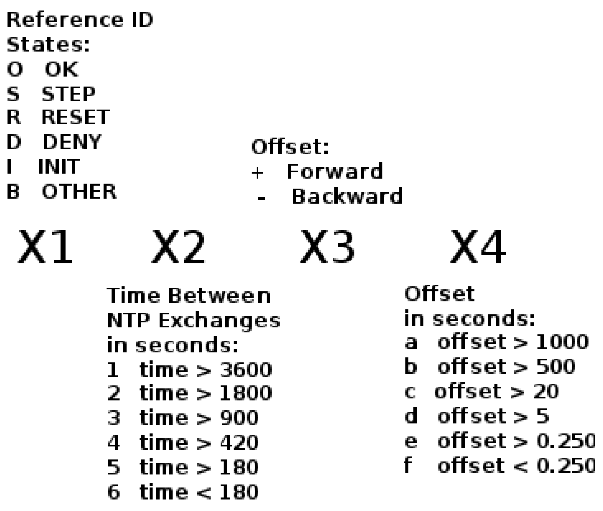


Figure 3.5: Definition of States for one NTP Exchange

Table 3.7: Examples of Extended States

NTP Exchange State $X_1X_2X_3X_4$	Analysis of NTP packet Exchange
O 6 - f	X_1=O: The exchange was OK according to definition in Table 3.4. X_2=6: The previous NTP exchange was less than 3 minutes ago. X_3=-: The client's time is backward in time with respect to the server. X_4=f: The value of offset is less than 250 ms.
I 6 - a	X_1=I: There was a reboot of ntp service at client's side. X_2=6: The previous NTP exchange was less than 3 minutes ago. X_3=-: The client's time is backward in time with respect to the server. X_4=a: The value of offset is more that 1000 seconds. This state is considered dangerous since represents an attempt of on-line attack.
S 6 - f	X_1=S: The client steps in time. X_4=f: The value of offset is less than 250 ms.

3.6 Detection Module Implementation

The module implemented using NEMEA system, is *"NTP_Attack_Detector.py"*. This module implements the detection strategies described in the previous sections (Section 3.3 , Section 3.4 and Section 3.5).

The detection module receives NTP flow records in the input interface. The input interface, as well as the output interface, can be a TCP interface[20] (for remote communication between modules) or a UNIX socket[21] (for local communication). Every packet is processed by Algorithm 1 (to calculate the offset of the exchange) and then by Algorithm 4 (to set the estate of the exchange). In case an alert is generated, it is sent to the output interface.

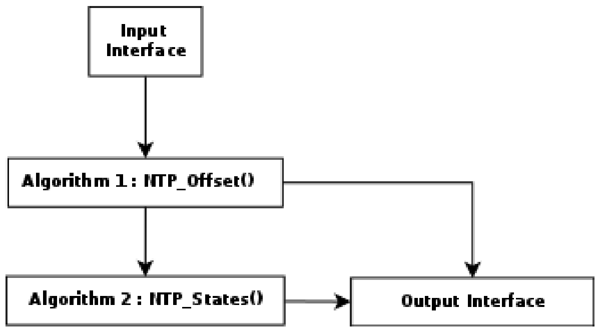


Figure 3.6: Detection Module Implementation

Table 3.8: Program Implemented for Detection Module

Program	Description
NTP_Attack_Detector.py	Program that implements the strategies of detection described in Chapter 1refchp:detectionModule

The functionalities and modes of operations of the detection module are described as follows:

1. Learning Mode: Learns normal behaviour of offsets. The outputs are values of *lower bound* and *upper bound* (offset threshold) for every NTP communication. Doesn't generate alerts of attacks. The outputs are stored in a configuration file *config.ini.*

2. Alert Mode: Uses the results of thresholds from *Learning Mode.* In case there is a new NTP conversation which wasn't learned previously, the

[20]TCP interface format t:⟨hostname or ip⟩, ⟨ port⟩
[21]UNIX socket format u:⟨socket identifier⟩

default threshold is : lower bound = -1,000 seconds and upper bound = 1,000 seconds. It is used the value of 1,000 seconds because not even the NTP will allow steps bigger that this value, as it was explain in Section 2.2 and Figure 2.1.

3. Step Analysis Mode: Generates a report with all states for every NTP packet exchange.

4. Focus Mode: The module can focus the analysis to a specific NTP communication (one client and one sever)

Chapter 4

Evaluation of Detection Module

This chapter describes the way how the module of detection was tested using traffic from real networks. NTP traffic was captured from different sources: from the laboratory described by Figure 2.4, also from NEMEA office at Faculty of Information Technology and finally from CESNET network. From CESNET network [22], it was captured 85.648.459 NTP flow records stored in a file which size was 17 GB. First, it is described three experimental cases about calculation of threshold. Second, NTP module is tested during on-line attack. Third, it is discussed the module's throughput. Finally, it is described some difficulties experienced during this evaluation. This experiments involve all the programs implemented: NTP plug-in, detection module and also scripts for attacks.

4.1 Evaluation of Thresholds

The strategy proposed in this thesis for detection of on-line attacks is based on the idea of computing the offset of a NTP packet exchange and compare that value with a threshold value defined by an *upper-bound* and a *lower-bound* that satisfy a *confidence interval* based on Chebishev's Inequality. It is important to remember from Section 2.2, that the maximum possible value of offset allowed by NTP protocol is 1,000 seconds. Table 4.1 shows some examples of *Confidence intervals* found experimentally using the detection module. It is important to notice that every case is independent and the results can vary. A good threshold for one specific case, can be very different to the results found for another case. Another important observation is that the calculation of *lower* and *upper-bound*, could be affected by the way the NTP flow records are capture, as it will be discussed in Section 4.4.

[22]CESNET: is a developer and operator of national e-infrastructure for science, research, development and education in Czech Republic `https://www.cesnet.cz/?lang=en`

Table 4.1: Experiments Confidence Interval

Experiment	Lower Bound (sec)	Mean (sec)	Upper Bound (sec)
1	-0.68	-0.008	0.69
2	-26.22	-1.18	23.85
3	-0.27	-0.002	0.27
4	-65.28	-6.49	52.28
5	-77.97	-12.33	53.31
6	-59.51	-5.57	48.37
7	-79.91	-10.89	58.11

During this section it is going to be shown a set of cases that illustrates the way the learning mode works using the detection module. It is important to remember that the purpose of this mode is to acquire knowledge about offsets of normal NTP traffic:

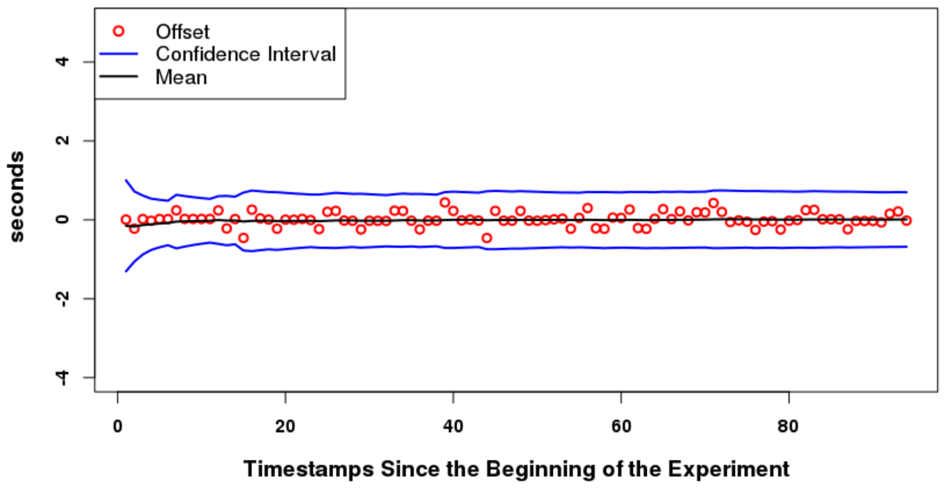


Figure 4.1: Case 1 Confidence Interval

Case 1 "Regular Offsets, Same Network": Figure 4.1, shows an example of calculation of confidence interval for the case described in Section 2.2 and Figure 2.4. During this experimental test of learning process; it was used real traffic from a laboratory where the client and server are in the same network (low delay traffic). Both the client and server are synchronized. This figure illustrates the way the *confidence interval* of 93.75% is calculated while NTP traffic is detected. It was calculated the *threshold* for 100 NTP packet exchanges. Since the values of NTP *offsets* are very regular, the *mean* and *confidence interval* are regular as well.

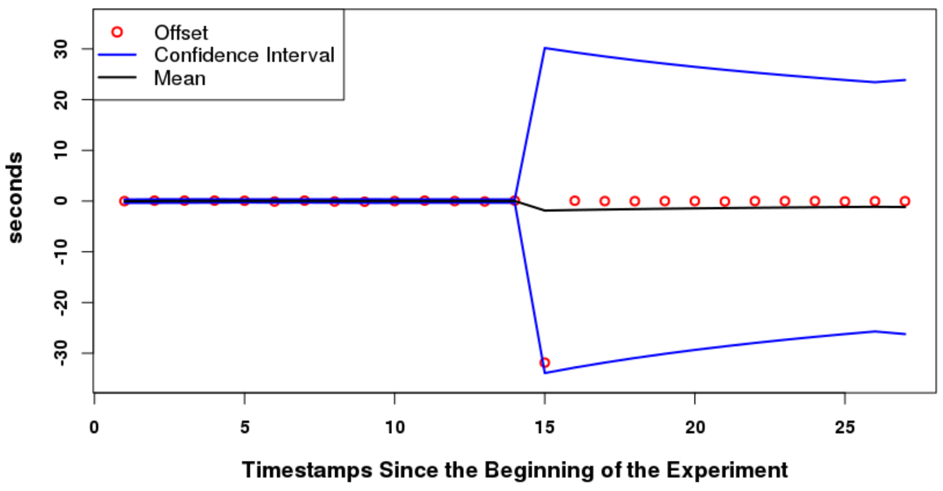


Figure 4.2: Case 2 Confidence Interval

Case 2 "Non-Regular Offsets, Server in Internet": For this experiment, the server is in internet (which increases the delay), and the client and server are synchronized. The results in Figure 4.2 illustrates very well the process of learning done by the detection module. The first packet exchanges present very regular offsets (order of millisecond); then, there is one offset of about -30 seconds. After this, the interval grows considerably. Finally, the values of offsets are regular again (order of millisecond), and the interval starts to decrease gradually.

Case 3 "CESNET Network, Long Experiment": This experiment is interesting since the detection module was tested using NTP data captured during 3 days. This test was done using 2,831 NTP packet exchanges (5,662 NTP packets). The results of this experiment are shown in Figure 4.3. It can be seen how the confidence interval adapts to the new results during a long term experiment. It is important to mention that some of the offsets were calculated incorrectly by the Algorithm 1, due to the fact that some of the NTP packets were not recorded in correct sequence by the probes; this difficulty is going to be discussed in detail in Section 4.4.

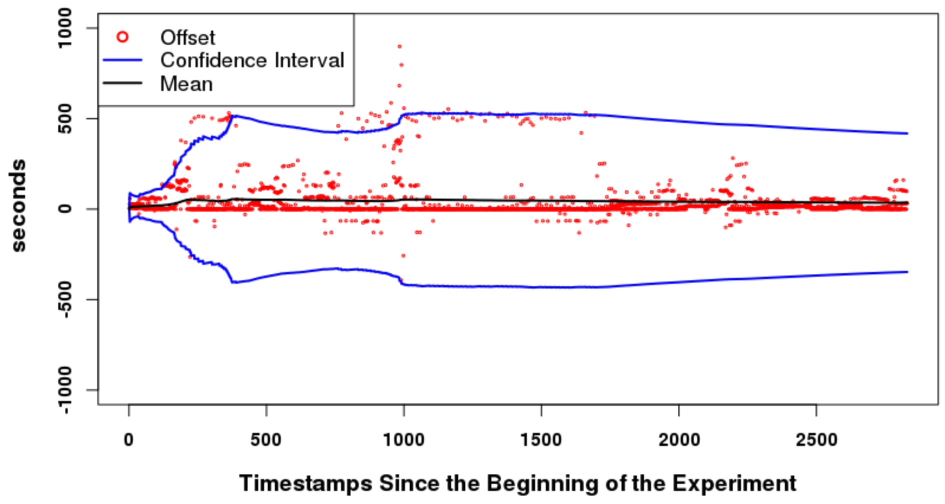


Figure 4.3: Case 3 Confidence Interval

4.2 Test During Attack

The on-line attack described by Figure 2.2, was replicated but this time the traffic was analysed by the detection module. The detection module detected 5 consecutive ALERTS of on-line attack when it was detected offsets of about 1,800 seconds. It is important to notice that once the attack is successful, the alerts stop because the offsets are normal. The detection module is capable of detecting on-line attacks while the attack is in process, but doesn't detect the attack when it happened in the past, even if the MITM attack is still in process. Additionally, it was tested that the module generates alerts when a *kiss of death* packet i detected.

4.3 Throughput Calculation

The throughput of the detection module was tested in the following way: 85.648.459 flow records captured during 3 days from two probes located in a CESNET network (the records were anonymized, which means that the IP addresses don't correspond to the real ones). These records were used as input for the detection module. It was measured the time at the beginning of the computation of the flow records and at the end. It was found that the detection module can compute 8,196 NTP records per second. During the capture of flow records using the monitoring probes, it was observed 350 NTP flow records per second on average. As a consequence, the detection module is capable of computing NTP records faster than the average value of received records. The testing machine specifications are described as follows:

Operative System: Fedora 21 GNU/Linux

CPU: Intel i7-4785T @ 2.2GHz

RAM: 16 GB

Cores: 8

The memory required to store the data related to one NTP conversation using Python dictionaries is low. Every one will use one entry of the 10 dictionaries implemented for different purposes. The memory starts to increase while the number of NTP conversations increases. During the test using the 85.648.459, it was detected 27,321 NTP communications.

Table 4.2, discusses the pros and cons about the proposed method for threshold calculation.

4.4 Difficulties During Evaluation

During the test of the detection module there were some difficulties that are described as follows. This section also works as lessons learned about the conditions on which the detection module works correctly:

1. There was a test where the detection module didn't produce any result. It was due to the fact that the file with flow records only contained records for NTP request. So, it is necessary for the correct functionality of the module, to see both directions of the communication (request + response).

2. There were other tests where the results from the detection module were not correct, due to the fact that the file with flow records contained duplicate records. So, it is necessary for the correct functionality of the module, to count with not duplicate records. It was probably caused by the nature of configuration of monitoring infrastructure, where the main router mirrors all packets to local monitoring probe.

3. There were also cases where some records were not sorted by start time of the flow record, causing incorrect calculations of offsets.

4. The quality of results from the detection module depends on a correct set of records: Sorted by start time of the flow, not duplicate records. The set of flow records must follow the order of the NTP communication between client and server.

5. The reason of these problems with flow records was mainly because in CESNET backbone network it is not possible to see all packets from traffic due to routing policy, so, it is possible that packets of the same NTP conversation travel via different paths. Another identified problem appear when the device from which the flow records were taken, uses two interfaces for the capture, which can produce duplicate records.

Table 4.2: Pros and Cons of Threshold Definition by using Mean an Standard Deviation

Pros	Cons
Working with standard deviation guaranties that Threshold will be in the same units as the measured variable (seconds). This values are simple to calculate. Doesn't require to store all previous values of offsets, it is just necessary to store the total sum of offsets and sum of square values of offsets.	Online learning is used to build a traffic profile for a given network.
This strategy also allows to apply Chebishev's Inequality, which is very useful for cases of random variables without specific probability distribution.	The efficiency of this approach depends on the accuracy of the traffic profile generated.
This method is used to learn what constitutes normal activity from its observations, and the confidence intervals automatically reflect this increased knowledge.	The Algorithm 1, requires that the flow records are received by the detection module in the same sequence that the NTP communication traffic, which implies that the monitoring strategy and infrastructure must be taken into account.
Because the confidence intervals depend on observed data, what is considered to be normal for one user can be considerably different for another.	The learned information from one NTP communication is useless for other NTP communications.
The learning process for multiple NTP communications can be done simultaneously.	The transition from *learning mode* to *alert mode* is not automatic and there is not a strategy that determines when the learning is over.

6. The memory usage of the module could be improved if it is implemented a time-out for idle NTP communications that clears the entries on the Python dictionaries related with these conversations.

4.5 Future Work

After the end of this thesis, some ideas for future work in this area are explained:

1. It is possible to focus security of NTP by testing and detecting off-line attacks based on fragmentation of NTP packets.

2. Some alternatives for detection of on-line attacks could be designed. For instance, the strategy of states definition explained in Section 3.5, could be combined with Markov's Chain analysis and define the probability of occurrence for each state. This idea could be also used to detected new anomalous behaviour of NTP traffic.

3. In 2016 it was found new vulnerabilities on NTP protocol. One of them is described by *CVE-2016-1550 "Improper Input Validation"*: ntpd does not use a constant-time memory comparison function when validating the authentication digest on incoming packets. In some situations this may allow an attacker to conduct a timing attack to compute the value of the valid authentication digest causing forged packets to be accepted by ntpd. *CVE-2016-1548 "Authentication Bypass by Spoofing"*: By spoofing packets from a legitimate server, an attacker can change the time of an ntpd client or deny service to an ntpd client by forcing it to change from basic client/server mode to interleaved symmetric mode[23].

4. Implementation in C language of the algorithms exposed in this thesis in order to have better performance.

5. It would be interesting and useful to design a technique that allows the transitions from *learning mode* to *alert mode* automatically when an algorithm considers that the traffic is stable and the values of offset don't differ too much among them.

6. In the introduction of this thesis it was explained some theoretical scenarios where some applications or protocols (Bitcoin, HSTS protocol, DNS protocol, TLS certificates), could be affected if the time integrity is not guaranteed. It would be worthy to test this potential consequences in order to find new ways to protect these applications and protocols.

[23]CERT, Vulnerability Notes Database `https://www.kb.cert.org/vuls/id/718152`

Conclusion

The implementation of NTP protocol ntpd v4.2.6 is vulnerable to on-line attacks when the -g option is enabled, since it is possible for an attacker to send spoof timestamps just after the ntpd service is rebooted on the client side. As a consequence, the on-line attacker can send the client forwards or backwards in time.

Additionally, the implementation ntpd v4.2.6 is vulnerable to off-line attacks by priming the pump, where the attacker spoofs and sends many mode 3 bogus packets to the server; consequently, the server will send a Kiss-of-Death packet to the client, notifying that the client must reduce the rate of queries. The off-line attack by spoofing KoD packet was tested for ntpd 4.2.6p5 on Ubuntu14.04.3, and it was found that this particular scenario is not vulnerable due to the fact that ntpd applies Test 2 to KoD packets (a KoD is no longer trivially spoofable with ntpd v4.2.6p5). Finally, the best option to perform off-line attack is by using priming the pump technique.

The security of NTP protocol matters since it lies in the background of many other systems, which means that there are applications and protocols whose correctness and security relies on the correctness of local clocks. Some consequences of attacking NTP could be: Bitcoin transactions that are composed by blocks of timestamps could be altered; HSTS protocol could be disabled and web traffic wouldn't be encrypted and vulnerable to eavesdropping; DNS protocol could be affected by making expire cache on DNS servers and possible flooding of DNS queries; TLS certificates could be affected by forcing the host to accept certificates that the attacker acquired fraudulently and make the communication vulnerable to eavesdropping.

Strategies for detection of on-line and off-line attacks were designed: Threshold definition by calculation of a confidence interval during learning mode of the detection module. Detection of on-line attacks by comparison of offsets with threshold boundaries using alert mode. Finally, detection of on-line and off-line attacks by states definition of NTP exchanges. This last strategy reduces the possibility of false positives when on-line attacks are detected, it is designed to be easy readable by security analysts and useful for network forensic analysis.

Calculation of offsets of NTP packet exchanges is useful when on-line attacks are detected. Detection of an anomaly based on offsets is simple to implement by comparison with a threshold that must be determined experimentally. Using the mean and standard deviation of a set of measured offsets in NTP communications, works as a source of knowledge about the behaviour of the NTP traffic; this allows to set the confidence interval by using Chebishev's Inequality. This strategy works well in cases where it is guaranteed that the flow records reflects the real sequence of NTP traffic.

The contribution of this thesis starts by the design of a virtual environment for proving attack strategies over NTP. Also, the implementations of scripts to perform the attacks. Additionally, the implementation of a plug-in for parsing NTP flow records that works for NEMEA systems. Moreover, the implementation of a detection module according to the designed strategies for detection.

Bibliography

[1] Mills, D.; Martin, J.; Burbank, J.; et al. RFC 5905: Network time protocol version 4: Protocol and algorithms specification. *Internet Engineering Task Force*, 2010.

[2] IEEE. IEEE Standard for a Precision Clock Synchronization Protocol forvNetworked Measurement and Control Systems. *IEEE Std. 1588*, 2008: pp. 1–269.

[3] Elson, J.; Girod, L.; Estrin, D. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, volume 36, no. SI, 2002: pp. 147–163.

[4] Parkinson, B. W. *Progress in Astronautics and Aeronautics: Global Positioning System: Theory and Applications*, volume 2. Aiaa, 1996.

[5] Corbixgwel. Timejacking and bitcoin: The global time agreement puzzle. `http://culubas.blogspot.cz/2011/05/timejacking-bitcoin_802.html`, 2011, accessed February 2016.

[6] Mattsson, J.; Näslund, M. Detection and Mitigation of HTTPS Man–in–the–Middles and Impersonators.

[7] Selvi, J. Bypassing HTTP Strict Transport Security. *Black Hat Europe*, 2014.

[8] Malhotra, A.; Cohen, I. E.; Brakke, E.; et al. Attacking the Network Time Protocol. *NDSS'16*, 2016.

[9] Selvi, J. Delorean. `https://github.com/PentesterES/Delorean`, accessed February 2016.

[10] RedHat. CVE-2015-5300. `https://access.redhat.com/security/cve/cve-2015-5300`, 2015, accessed February 2016.

[11] RedHat. Security Announcement cve-2015-5300. https://kb.juniper.net/InfoCenter/index?page=content&id=JSA10711&actp=search, 2015, accessed February 2016.

[12] Bartos, V.; Zadnik, M.; Cejka, T. Nemea: Framework for stream-wise analysis of network traffic. *CESNET, Technical Report*, 2013.

[13] Stenn, H. Securing the Network Time Protocol. *Queue*, volume 13, no. 1, 2014: p. 20.

[14] Graham, J. Understanding and Mitigating NTP-based DDoS Attacks. https://blog.cloudflare.com/understanding-and-mitigating-ntp-based-ddos-attacks/, 9 January 2014, accessed February 2016.

[15] Wikimedia. Network Time Protocol. https://en.wikipedia.org/wiki/Network_Time_Protocol, Last modified on 13 April 2016, accessed February 2016.

[16] NTP.org. Clock Quality (FAQ). http://www.ntp.org/ntpfaq/NTP-s-sw-clocks-quality.htm, 2010, accessed February 2016.

[17] Mills, D. L. Internet Time Synchronization: The Network Time Protocol. *Internet Engineering Task Force*, 1994.

[18] NTP.org. How NTP Works. https://www.eecis.udel.edu/~mills/ntp/html/warp.html, 2014, accessed February 2016.

[19] NTP.org. How does it work? http://www.ntp.org/ntpfaq/NTP-s-algo.htm, 2010, accessed February 2016.

[20] Baum, F. ntpd - Network Time Protocol (NTP) Daemon. https://www.eecis.udel.edu/~mills/ntp/html/ntpd.html, 2014, accessed February 2016.

[21] Oracle. VirtualBox. https://www.virtualbox.org, 2016, accessed February 2016.

[22] Debian.org. Debian The universal operative system. https://www.debian.org, 2016, accessed February 2016.

[23] Canonical. Ubuntu Operative System. http://www.ubuntu.com, 2016, accessed February 2016.

[24] RFC-791. Internet Protocol Version 4 RFC-791. https://tools.ietf.org/pdf/rfc791.pdf, 1981, accessed February 2016.

[25] Python. https://www.python.org/, 2001, accessed February 2 016.

[26] Secdev.org. Scapy Usage. `http://www.secdev.org/projects/scapy/doc/usage.html`, 2008, accessed February 2016.

[27] CVE.org. CVE. `https://cve.mitre.org/`, 1999, accessed February 2016.

[28] Jajodia, S.; Mazumdar, C. *Information Systems Security: First International conference, ICISS 2005, Kolkata, India, December 19-21, 2005, Proceedings*, volume 3803. Springer, 2005.

[29] Teardrop. `http://www3.physnet.uni-hamburg.de/physnet/security/vulnerability/teardrop.html`, 2010, accessed February 2016.

[30] Dan, G. New attacks on Network Time Protocol can defeat HTTPS and create chaos. `http://arstechnica.com/security/2015/10/new-attacks-on-network-time-protocol-can-defeat-https-and-create-chaos/`, 2015, accessed February 2016.

[31] VMware. Timekeeping in vmware virtual machines: vsphere 5.0, workstation 8.0, fusion 4.0. `http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf`, 2011, accessed February 2016.

[32] Hofstede, R.; Celeda, P.; Trammell, B.; et al. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *Communications Surveys & Tutorials, IEEE*, volume 16, no. 4, 2014: pp. 2037–2064.

[33] Thottan, M.; Ji, C. Anomaly detection in IP networks. *Signal Processing, IEEE Transactions on*, volume 51, no. 8, 2003: pp. 2191–2204.

[34] Denning, D. E. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, , no. 2, 1987: pp. 222–232.

[35] Toader, G. On Chebyshev's inequality for sequences. *Discrete Mathematics*, volume 161, no. 1, 1996: pp. 317–322.

Appendix A
Acronyms

NTP	Network Time Protocol
PTP	Precision Time Protocol
GPS	Global Position System
RBS	Reference Broadcast Synchronization
HSTS	HTTP Strict Transport Security
MITM	Man-In-The-Middle
HTTPS	Hyper Text Transfer Protocol Secure
HTTP	Hyper Text Transfer Protocol
URL	Uniform Resource Locator
NEMEA	Network Measurement Analysis
DoS	Denial-of-Service
IP	Internet Protocol
UDP	User Datagram Protocol
DNS	Domain Name Service
TLS	Transport Layer Security
LI	Leap Indicator
IT	Information Technology
ARP	Address Resolution Protocol
IP	Internet Protocol

Ta	Time attack
Ao	Attack offset
KoD	Kiss-of-Death
IPFIX	IP Flow Information Export
UniRec	Unified Records

Appendix B

Contents of Enclosed CD

readme.txt	the file with CD contents description
virtual machines	the directory with virtual machines
src	the directory of source codes
module_detection	dyrectory module detection
NTP_Attack_Detector.py	detection module python
ntp_plug-in	directory ntp plug-in
ntpplugin.h	ntp plug-in .h file
ntpplugin.cpp	ntp plug-in .cpp file
scripts_attacks	directory scripts
arpPoisoning.py	arp poisoning attack in python
attackOnLine.py	on-line attack in python
thesis	the directory of LaTeX source codes of the thesis
text	the thesis text directory
thesis.pdf	the thesis text in PDF format
turorials	directory of tutorials